Escuela Politécnica Superior de Jaén

Grado en Ingeniería Electrónica Industrial

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Jaén

Trabajo Fin de Grado

APLICACIÓN DE LA VISIÓN POR COMPUTADOR A LA RESOLUCIÓN DE LABERINTOS CON UN ROBOT MÓVIL DE LABORATORIO

Alumno: Francisco Javier Martínez Muñoz

Tutor: Prof. D. Sergio Illana Rico
Tutor: Prof. D. Diego Manuel Martínez Gila
Dpto: Ing. Electrónica y Automática

Septiembre, 2023

Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Ingeniería Electrónica y Automática

Don Sergio Illana Rico y Don Diego Manuel Martínez Gila , tutores del Proyecto Fin de Grado titulado: Aplicación de la visión por computador a la resolución de laberintos con un robot móvil de laboratorio, que presenta Francisco Javier Martínez Muñoz, autorizan su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, septiembre de 2023

El alumno:

Firmado por MARTINEZ MUÑOZ FRANCISCO JAVIER - ***1642** el día 03/09/2023 con un certificado emitido por AC FNMT Usuarios

Los tutores:

Firmado digitalmente por MARTINEZ GILA DIEGO MANUEL - 77338812R
Fecha: 2023.09.04 11:40:45 +02'00'

ILLANA RICO SERGIO - 77364877F

Firmado digitalmente por ILLANA RICO SERGIO - 77364877F
Fecha: 2023.09.04 16:03:42 +02'00'

Resumen

El objetivo de este trabajo es la implementación de los algoritmos de control y navegación necesarios para la resolución de cualquier laberinto, utilizando un robot móvil comercial, en este caso el robot de Micro:bit Maqueen Plus V2. Una vez resuelto este objetivo, se compararán los resultados con los obtenidos al resolver el mismo problema, pero esta vez mediante la integración de una cámara cenital que visualiza en todo momento el laberinto completo mientras el robot navega por el laberinto.

Abstract

The objective of this work is that using a commercial mobile robot, in this case the Micro:bit Maqueen Plus V2 robot, the implementation of the control and navigation algorithms necessary for the resolution of any maze can be varied during the presentation of the solution. Once this objective is solved, the results will be compared with those obtained by solving the same problem, but this time by integrating a zenithal camera that visualizes the entire maze at all times while the robot navigates through the maze.

Índice

Figuras

1 Introducción

En este Trabajo de Fin de Grado, se presenta la implementación de un robot de laboratorio que es capaz de resolver laberintos utilizando dos enfoques diferentes. En primer lugar, se utiliza un enfoque basado en sensores infrarrojos, normalmente usados en robot sigue líneas, para que el robot encuentre la salida del laberinto de forma autónoma. En segundo lugar, se implementa un enfoque basado en visión por computador, en el que se toma una fotografía del laberinto y se implementan los algoritmos necesarios para determinar la ruta óptima a seguir.

Al final se comparan los resultados obtenidos con ambos enfoques y se obtienen conclusiones acerca de cuál de ellos es más óptimo en cuanto a eficiencia de la resolución del laberinto. Esto permite evaluar las ventajas y desventajas de cada uno de los enfoques y determinar en qué situaciones es más adecuado utilizar cada uno de ellos.

1.1 Antecedentes y estado del arte

Un laberinto, de acuerdo con la definición de la Real Academia Española (RAE), es un lugar formado artificiosamente por calles y encrucijadas, para confundir a quien se adentre en él, de modo que no pueda acertar con la salida. (Real Academia Española, 2023).

En inglés se pueden distinguir dos palabras distintas para referirnos a un laberinto, dividiéndose entre dos sustantivos: labyrinth (que en español podemos denominar como laberinto unicursal) y maze (que en español podemos denominar como laberinto multicursal).

Un laberinto unicursal está diseñado sobre un camino que no conlleva una sensación de pérdida. Aunque la estructura del laberinto sea muy compleja, al final siempre se encuentra la salida solo siguiendo el camino, ya que no da pie a la toma de decisiones, sólo hay un camino que podemos tomar y no hay callejones sin salida.

Por el contrario, el laberinto multicursal, tiene muchos caminos o rutas y puede darse la ocasión en que tengamos que elegir entre dos o más caminos distintos, en los que alguno de ellos puede no ser la opción correcta y pueden dar a callejones sin salida.

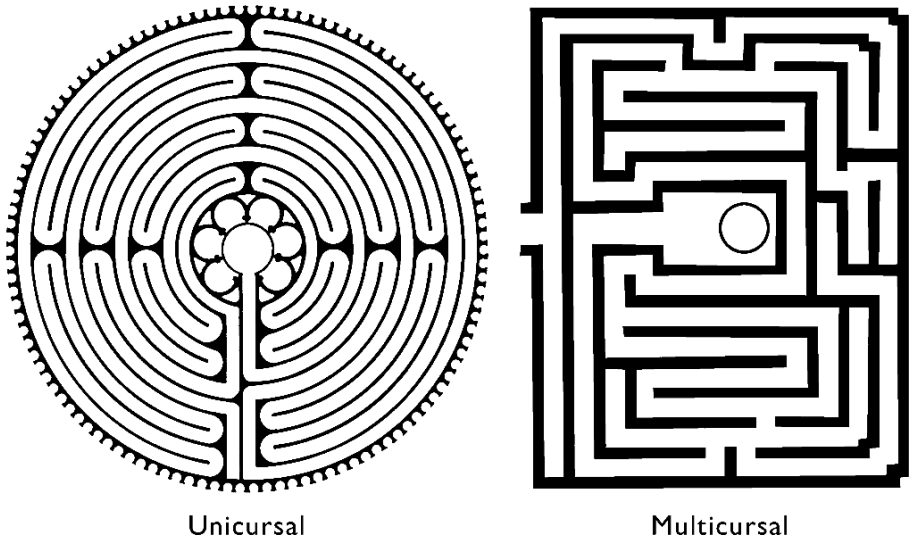


Figura 1.- Ejemplos de laberinto unicursal y laberinto multicursal

Uno de los métodos más sencillos de resolución de laberintos es la regla de la mano derecha o izquierda. Consiste en tocar con la mano derecha o izquierda una de las paredes del laberinto y comenzar a recorrerlo sin despegar la mano de la pared. De esta forma, se asegura que se recorran todas las paredes del laberinto hasta llegar a la salida.

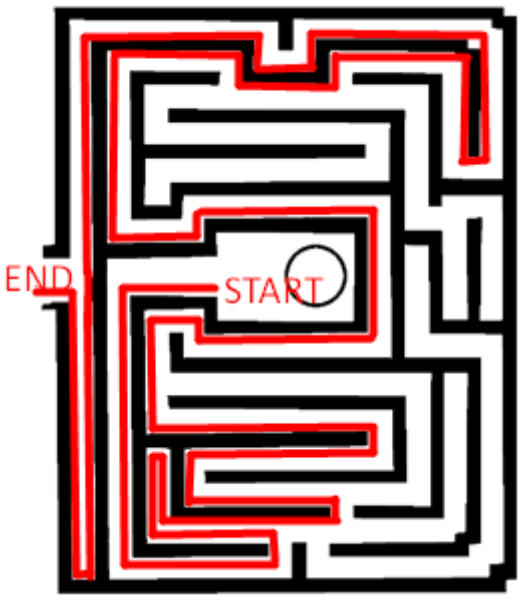


Figura 2.-Ejemplo de resolución con el algoritmo de la mano derecha/izquierda

La ventaja de este algoritmo es que es el más sencillo de aplicar, pero tiene ciertas desventajas, la primera de ellas es que no es el más óptimo, porque para llegar a la salida a veces es necesario tener que recorrer todo el laberinto, por lo que la velocidad de escapatoria del laberinto dependerá de donde se empiece a resolver.

El otro inconveniente, que es el más importante, es que hay modelos de laberinto en el que no es efectivo este método. Por ejemplo, en un laberinto formado por círculos concéntricos, en el que uno se quedaría dando vueltas infinitamente. Para el caso de estudio de este proyecto no es problema, por la forma en la que se construirá el laberinto, nunca se dará este tipo de situación.

Otro algoritmo más complejo que el anterior es el algoritmo de búsqueda en anchura (*BFS – Breadth First Search*). Es un algoritmo de búsqueda utilizado para recorrer o buscar elementos en un grafo.

Un grafo es un conjunto de objetos llamados vértices o nodos unidos por enlaces llamados aristas o arcos, que permiten representar relaciones entre los elementos.

Aunque en este caso de estudio se habla de laberintos y no de grafos, lo cierto es que se puede establecer una relación entre grafos y laberintos. Como se observa en la siguiente Figura 3, se puede establecer un vértice en cada posición fila-columna del laberinto, obteniendo como resultado el grafo de la Figura 4. Los nodos de este grafo se corresponden con los vértices establecidos en el laberinto.

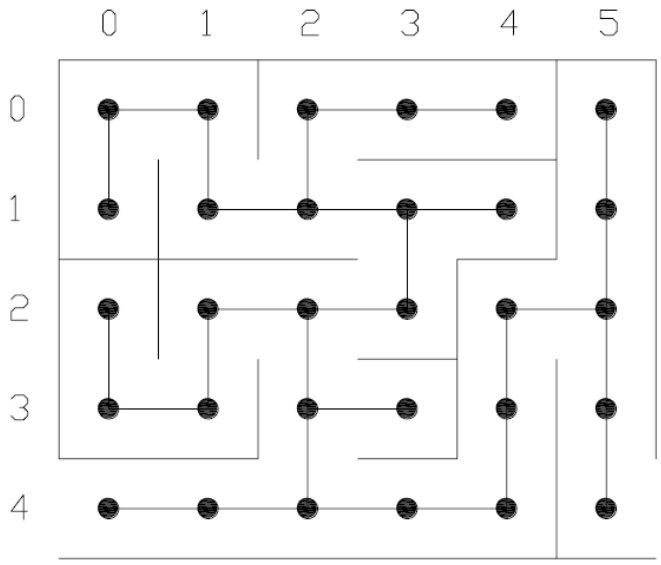


Figura 3.- Grafo interno de un laberinto

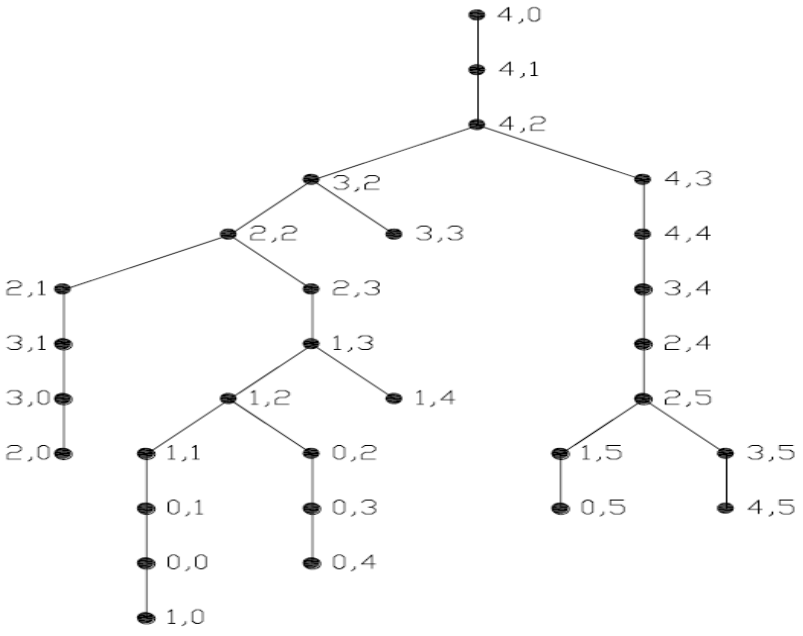


Figura 4.- Representación del grafo que hay dentro del laberinto

De esta manera, si conocemos el punto de entrada y el punto de salida del laberinto, se puede aplicar teoría de grafos y los algoritmos correspondientes para recorrer o buscar elementos en un grafo, como en este caso, el algoritmo de búsqueda en anchura o búsqueda en amplitud.

Para el algoritmo de búsqueda en anchura, se comienza definiendo dos puntos dentro del laberinto: uno para establecer la entrada y otro para determinar la salida que se desea alcanzar. Después, se crea una cola para

almacenar los nodos del laberinto que aún no han sido visitados y se agrega el punto de la entrada a la cola. Mientras la cola no está vacía, se saca el primer nodo de la cola y se examinan sus vecinos, que son los nodos adyacentes al nodo actual. Si se encuentra el nodo objetivo (la salida), se habrá encontrado una ruta hasta ese punto, y se detiene el algoritmo. Sin embargo, si el nodo no ha sido visitado, se marca el nodo como visitado y se agregan los vecinos de ese nodo a la cola para su posterior exploración. Estos pasos se repiten hasta que se encuentre la ruta o hasta que la cola esté vacía.

Para poder acceder después a la ruta desde el nodo de entrada al de salida, se podría también crear, al mismo tiempo en el que se creó la cola para almacenar los nodos del laberinto, una estructura como una matriz. Esta matriz podría almacenar todas las rutas posibles a los vecinos adyacentes, lo que permitiría llevar un registro de los caminos seguidos para alcanzar cada uno de ellos. De esta manera, cuando finalmente se encuentre la salida, tendremos la ruta completa desde la entrada hasta la salida.

Si la cola está vacía y no se ha encontrado una ruta hacia la salida, significa que no hay una ruta desde la entrada a la salida del laberinto.

Por último, otro algoritmo importante utilizado también en grafos es el algoritmo de Dijkstra. Fue desarrollado por el científico Edsger Dijkstra en 1956 y se emplea para encontrar el camino más corto entre un nodo de origen y todos los demás nodos dentro del grafo.

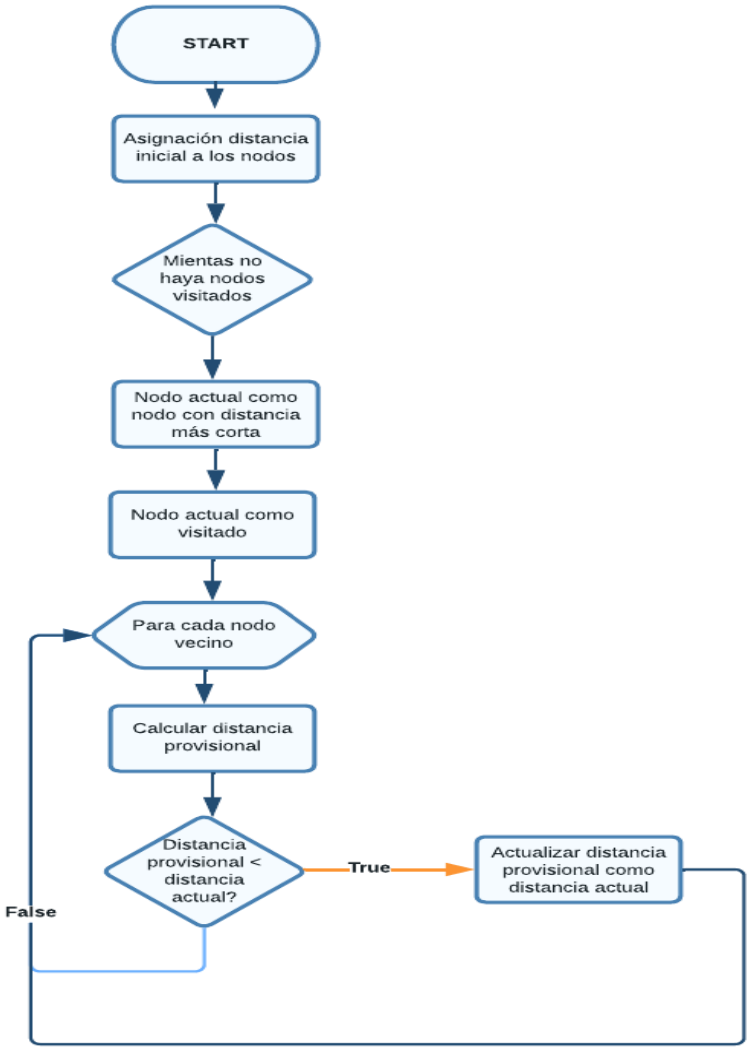


Figura 5.- Algoritmo de Dijkstra

La ejecución del algoritmo de Dijkstra comienza asignando una distancia inicial a todos los nodos del grafo. El nodo origen tiene una distancia de 0, mientras que los demás nodos se establecen a infinito, lo que indica que inicialmente no se ha encontrado ninguna ruta hacia ellos. A continuación, se selecciona el nodo con la distancia más corta como el nodo actual y se marca como visitado.

Después se actualizan las distancias, para cada vecino del nodo actual y se calcula la distancia provisional sumando la distancia del origen al nodo actual más la distancia del nodo actual al nodo vecino. Si esa distancia provisional es menor que la actual, se actualiza la distancia.

El proceso se repite con todos los vecinos del nodo actual hasta que todos hayan sido visitados y se hayan actualizado sus distancias o hasta que el nodo con la distancia más corta sea infinito, lo que indicaría que no se ha encontrado una solución y no hay rutas posibles hacia esos nodos.

1.2 Objetivos

Para este Trabajo de Fin de Grado se han definido los siguientes objetivos:

- Conseguir extraer información de un sensor complejo como una cámara de visión en el espectro visible a partir de su hoja de características e información del fabricante.
- Aumentar los conocimientos sobre los algoritmos de resolución de laberintos utilizados durante la navegación autónoma del robot.
- Desarrollar la habilidad para generar código de calidad en el lenguaje de programación seleccionado.
- Establecer un método para la comunicación entre el procesamiento de imágenes y el robot, de manera que se envíe la información necesaria para que el robot sea capaz de resolver el laberinto.
- Comparar los resultados obtenidos con ambos enfoques y sacar conclusiones acerca de cuál de ellos es más óptimo.

2 Material y tecnologías

2.1 Materiales para la construcción del laberinto

Para la construcción del laberinto se usan las siguientes piezas blancas que contienen una línea negra dentro. Hay varios tipos de piezas que sirven para construir el laberinto. (Figura 6)

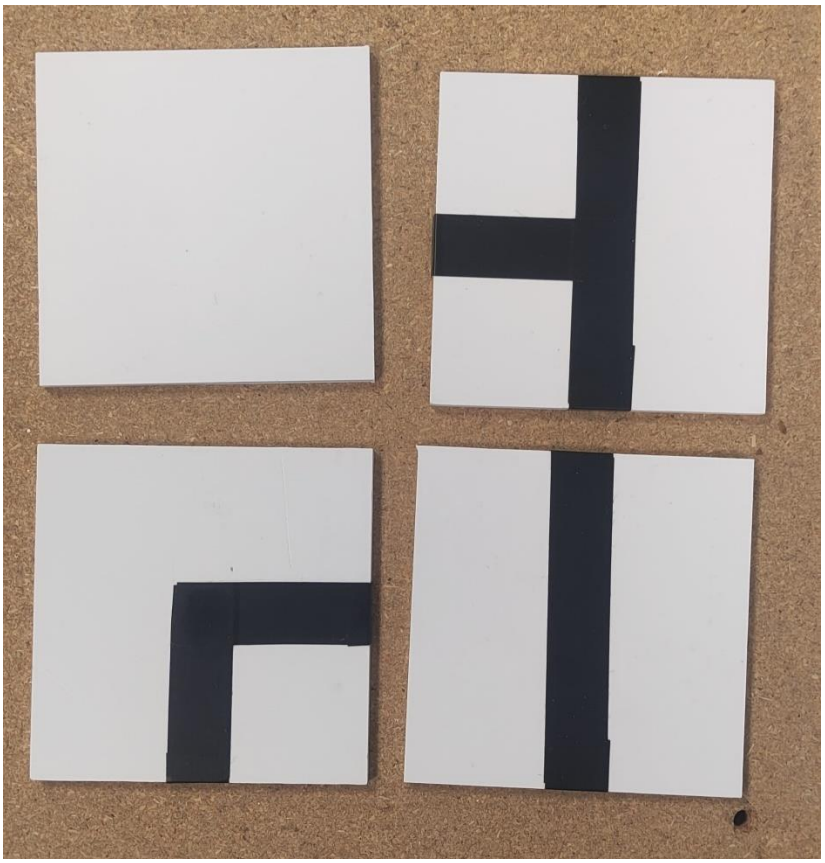


Figura 6.- Piezas para la construcción del laberinto

2.2 Placa programable Micro:Bit V2

Micro:Bit es una pequeña placa programable de tan solo 4x5 cm, diseñada para simplificar el proceso de programación. Su característica distintiva radica en el uso de un lenguaje de programación basado en bloques, lo que la convierte en una herramienta muy atractiva para introducir a personas en el mundo de la robótica y la programación. Con esta placa, se pueden explorar diferentes niveles de programación, ofreciendo tres opciones: control a través de una aplicación para dispositivos Android y iOS, programación con bloques de JavaScript y desarrollo en Python.

Figura 7.-Micro:Bit V2

2.3 Robot Maqueen Plus V2

El robot Maqueen es un robot especialmente diseñado para ser compatible con las placas programables de Micro:Bit.

Está creado para usarse para educación STEM (por sus siglas en inglés), que es el acrónimo de los términos en inglés *Science, Technology, Engineering and Mathematics.*

Entre sus características más destacadas del robot están: sus dos motores N20 que controlan las ruedas del robot, sensor ultrasónico SRO4, 5 sensores infrarrojos, entre muchos sensores incorporados. La programación del robot Maqueen es muy versátil gracias a su compatibilidad con las placas programables de Micro:Bit. Puede ser programado utilizando el lenguaje de programación por bloques con MakeCode de Microsoft, lo que facilita el proceso de programación para principiantes. También es posible programarlo con Python, lo que brinda una opción más avanzada y flexible para aquellos con conocimientos en este lenguaje.

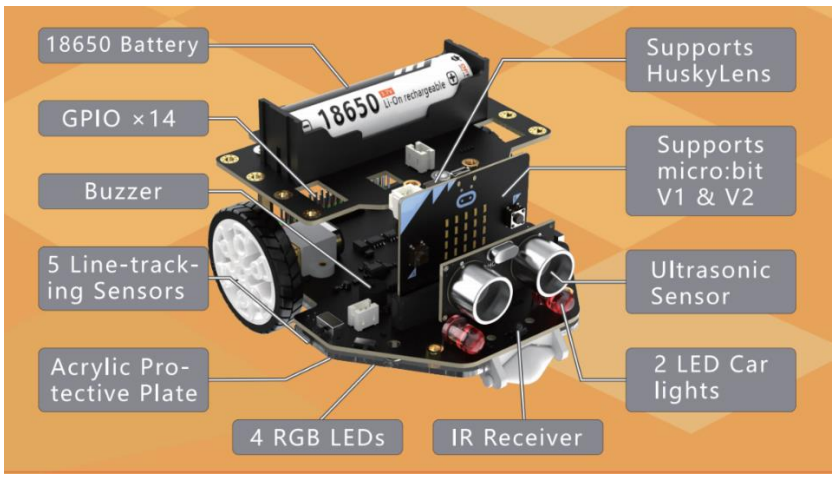


Figura 8.- Robot Maqueen Plus V2

Además de las características mencionadas anteriormente, en este proyecto se ha incorporado un módulo Bluetooth, específicamente el módulo HC-06, al robot Maqueen, conectado a los pines P0 y P1 del mismo. Este módulo Bluetooth agrega una funcionalidad adicional al robot, ya que permite enviar órdenes y comandos a la placa Micro:Bit que controla al robot.

Gracias a la implementación del módulo Bluetooth, el robot Maqueen adquiere la capacidad de recibir instrucciones de manera inalámbrica. Esto significa que se pueden enviar comandos desde un dispositivo externo, como es en este caso desde el ordenador, hacia la placa Micro:Bit, y esta a su vez transmitirá las órdenes a los motores y sensores del robot, permitiéndole moverse dentro del laberinto o realizar otras acciones programadas.

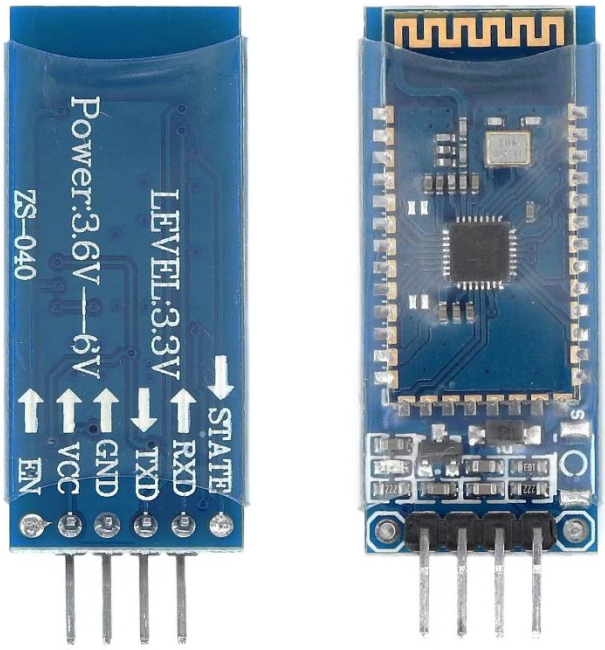


Figura 9.- Módulo Bluetooth HC-06

2.4 Cámara y objetivo IDS

La cámara que se usará para este proyecto es la UI-1220LE-C-HQ de la marca IDS. Se trata de una cámara industrial a color que cuenta con un sensor de tipo CMOS, lineal. La resolución de esta cámara es de 752 x 480 Pixel.

Figura 10.- Cámara UI-1220LE-C-HQ

Para el objetivo, se usará un objetivo marca AZURE. Este será el objetivo AZURE – 0420MM, con una distancia focal de 4 mm.

Figura 11.-Óptica AZURE – 0420MM

2.5 Software utilizado

Para la programación de Micro:Bit se usará el entorno de programación MakeCode de Microsoft. En esta herramienta se usará programación en bloques.

También se puede programar con Javascript o con Python, pero para este proyecto no es necesario profundizar en una programación más complicada, por lo que con la programación en bloques basta para programar la placa Micro:Bit.

Para el procesamiento de imágenes y toda la lógica principal de este proyecto se utilizará MATLAB.

MATLAB es una plataforma de programación y cálculo en forma de software orientada a tareas de ingeniería y matemáticas basado en matrices.

Entre sus prestaciones básicas se hallan la manipulación de matrices, la representación de datos y funciones, implementación de algoritmos y creación de interfaces de usuario, las cuáles se usarán a lo largo del proyecto. Además, se pueden ampliar las capacidades de MATLAB con las cajas de herramientas (toolboxes), que se usará para obtener la Toolbox de Image Processing y Computer Vision necesarias para el procesamiento de la imagen capturada e implementación de los algoritmos necesarios de resolución de laberintos.

3 Resolución mediante sensores Maqueen Plus V2

La construcción del laberinto se realizará con las piezas mencionadas anteriormente en el apartado de materiales. Un ejemplo de construcción del laberinto se puede ver en la Figura 12.

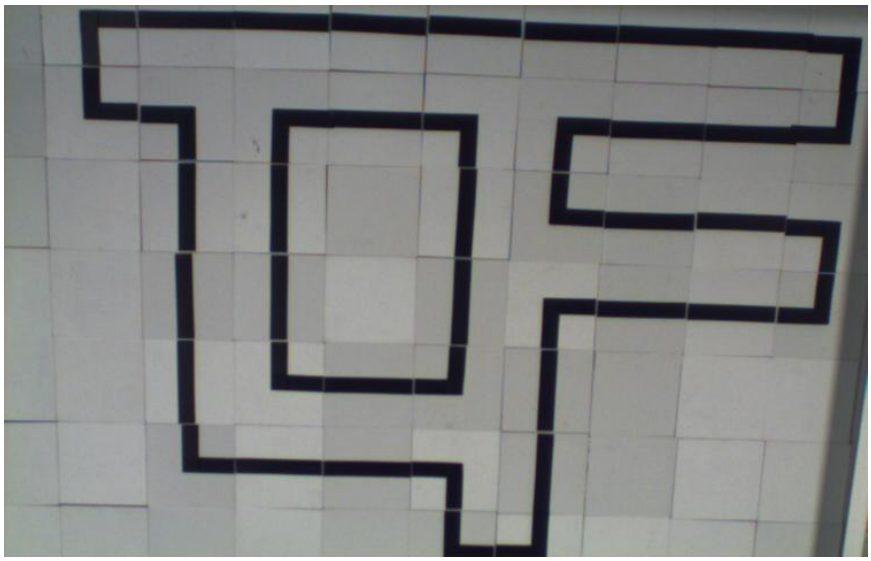


Figura 12.- Ejemplo de construcción del laberinto

En la solución de este problema, prescindiendo de la integración de sistemas de visión por computadora, se aprovechará la funcionalidad de los sensores incorporados en Maqueen Plus V2, en específico, los sensores infrarrojos (como se ilustra en la Figura 13) y el sensor de ultrasonidos. El enfoque algorítmico seleccionado para abordar esta situación será la estrategia de la regla de la mano derecha/izquierda o el seguidor de paredes, debido a su capacidad para adaptarse de manera óptima a las capacidades de los sensores disponibles.

Figura 13.-Sensores infrarrojos Maqueen Plus V2

Para este caso se aplicará la regla de la mano derecha, que hará que el robot siempre siga la pared del laberinto por la parte derecha. Partiendo de ahí, se creará el programa que tendrá la placa Micro:bit. Primero se creará una función que hará de seguidor de paredes por la parte derecha como se indica en la siguiente figura:

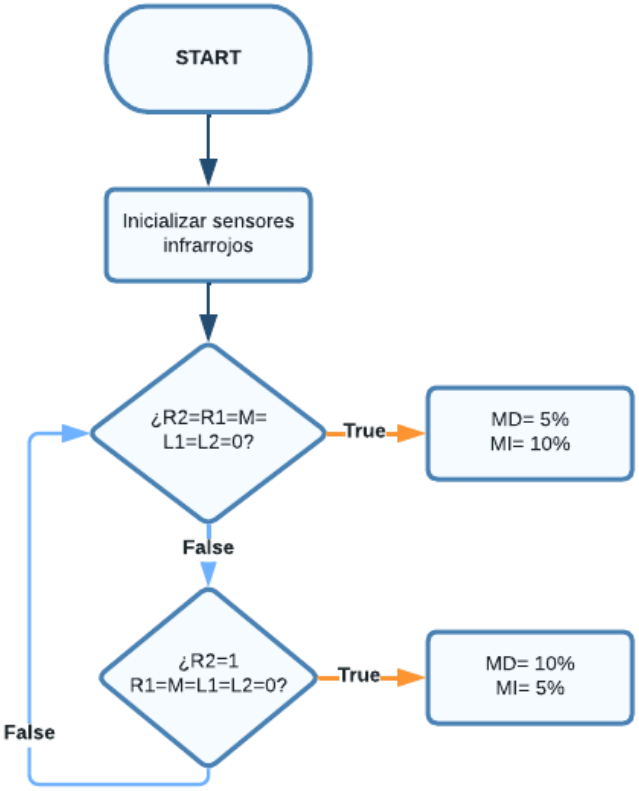


Figura 14.-Seguidor de pared derecha

Como se ve en la figura anterior, la función sólo contempla dos casos, en los que ningún sensor detecta línea negra (pared), en el que girará a la derecha en busca de la línea negra, pero con un poco de avance. En el otro caso, todos los sensores están a 0 menos el sensor R2 que es el que está más a la derecha del robot, cuando lo detecta, girará a la derecha, pero con un poco de avance. Esto hará que el robot siga una línea que actúa como pared por el lado derecho. El problema de esto es que en los giros de 90º que son más profundos se puede dar el caso de que el robot pase de largo, por lo que se hará un programa principal en el que se usarán también los sensores delanteros R1, M y L1, para hacer los giros de 90º. Además, se usará también

el sensor de ultrasonidos para detener el robot al final del laberinto, ya que al final del laberinto habrá un objeto que determinará el fin.

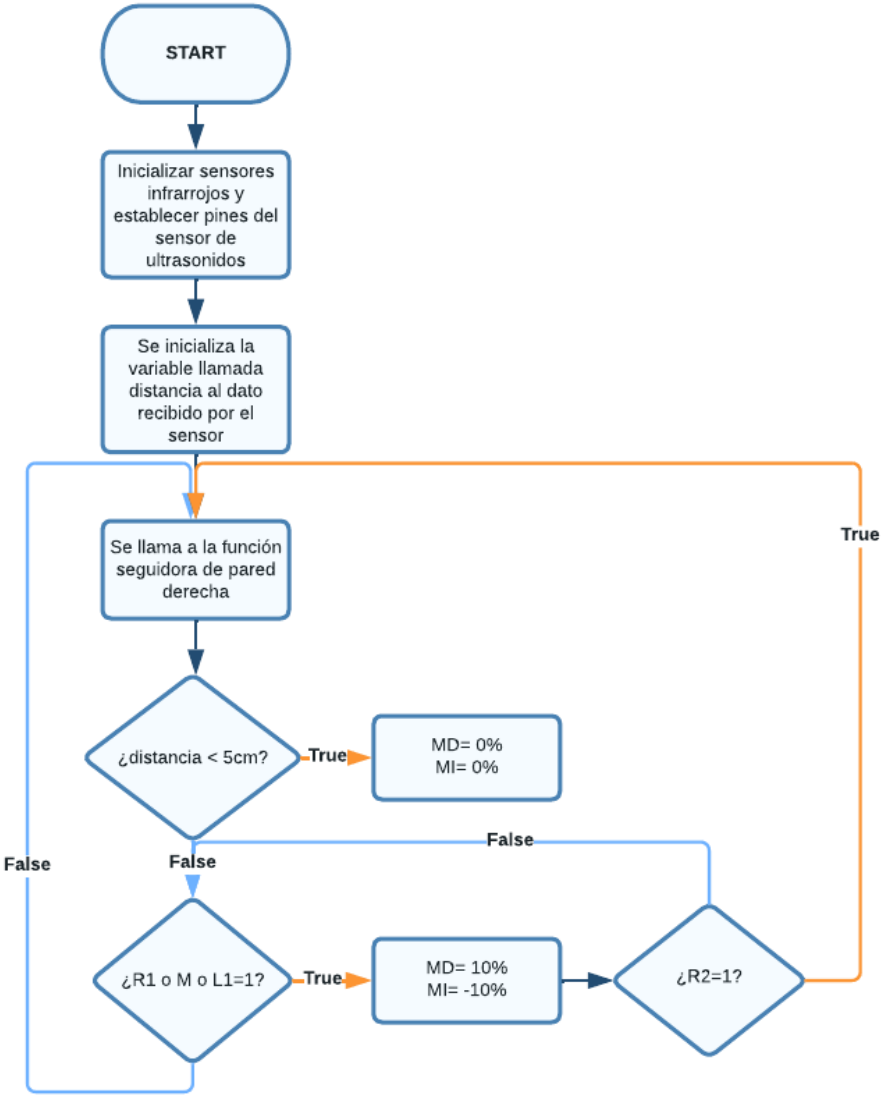


Figura 15.- Programa principal seguidor de pared

En el programa principal se empieza inicializando los sensores infrarrojos, estableciendo los pines correspondientes del sensor de ultrasonidos e inicializando la variable de distancia al dato que se recibe por el sensor de ultrasonidos. Se llama a la función seguidora de pared anterior, para que el robot siempre siga la pared derecha y se va comprobando si se ha llegado al destino, comprobando si la distancia que recoge el sensor es menor de 5 cm, si la distancia es menor el robot parará. Como he dicho antes, para los giros más bruscos de 90º se hace uso de los sensores R1, M y L1. Si alguno de ellos

detecta una línea, el robot girará sobre sí mismo hacia la izquierda hasta que el sensor R2 vuelva a detectar una línea. Si el sensor R2 vuelve a detectar línea, se vuelve a llamar a la función seguidora de paredes.

La velocidad de resolución de este algoritmo depende mucho de donde estén ubicadas la entrada y la salida del laberinto y de la pared elegida a seguir, ya que nada más empezar el laberinto hay que decidir si la pared que se va a seguir es la izquierda o la derecha. En el momento en que se elija cuál de las dos paredes hay que seguir, no se puede cambiar a la otra pared, por lo que la velocidad de resolución del laberinto depende mucho de que pared se elija al comienzo de resolver.

4 Resolución mediante la integración de visión por computador

Este apartado se dividirá en varios pasos que se han seguido al momento de la resolución: procesamiento de la imagen, obtención de la ruta, obtención de la posición del robot y envío de la ruta al robot.

Para la resolución de este apartado, se usará la herramienta de Microsoft, MakeCode y MATLAB, que servirá también para la creación de una interfaz para trabajar con el programa.

4.1 Procesamiento de la imagen

Figura 16.- Ejemplo del laberinto

En la Figura 16, se muestra un ejemplo de un laberinto que se usará para explicar el procesamiento de la imagen para la obtención de la ruta. Los caminos están limitados por líneas negras, y como se puede ver, se pueden tomar varios caminos para llegar a distintos puntos dentro del mismo. No hay una entrada y una salida definidas en la construcción del laberinto, se definirán en la interfaz que mostraré a continuación, para así poder hacer más combinaciones de entradas y salidas dentro de un mismo laberinto sin necesidad de añadir piezas o desmontarlo.

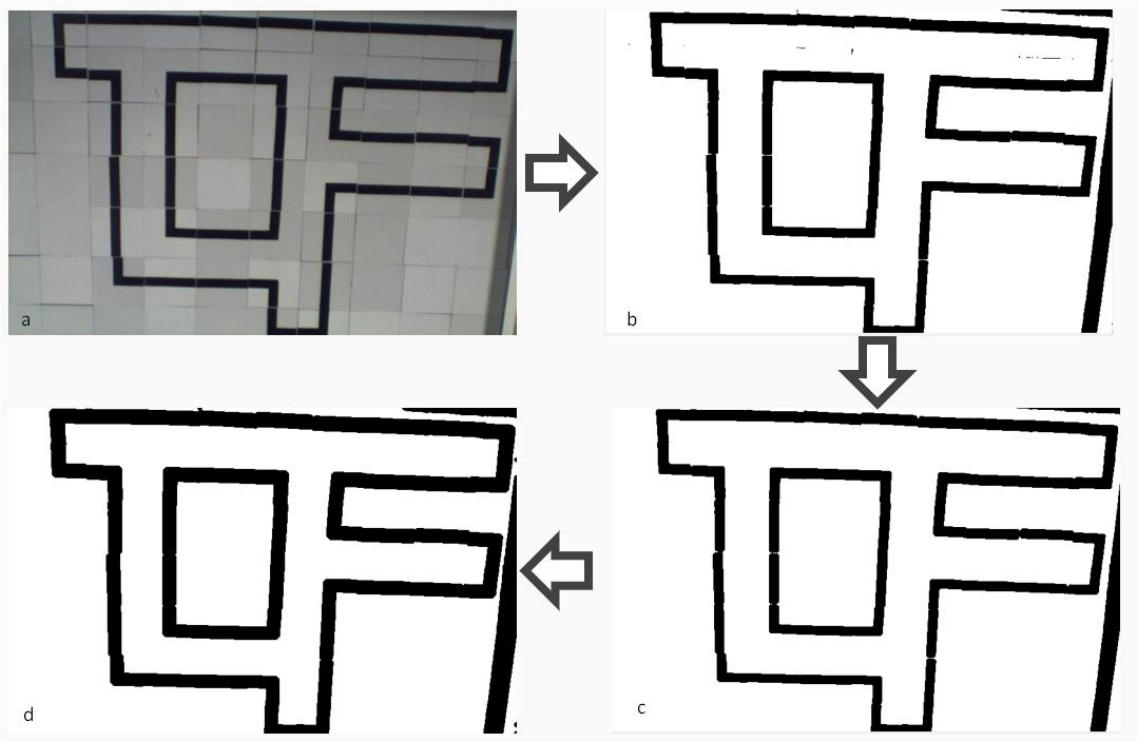


Figura 17.- Procesamiento de la imagen; a: imagen original; b:imbinarize; c:imclose; d: imerode

Para entender cómo se va a seleccionar la entrada y salida del laberinto y como se va a aplicar el algoritmo de resolución dentro del laberinto, primero se tiene que explicar con qué imagen se va a trabajar para todo ello. Para ello, primero hay que procesar la imagen, que pasa por las distintas fases que se ven en la Figura 17. Para poder trabajar con la imagen se necesitan cubrir los huecos que quedan entre pieza y pieza. Para ello, primero se binariza la imagen (Figura 17, b), una vez hecho esto se aplica una clausura con la función imclose con la que se conseguirá eliminar pequeños defectos en la imagen que no sirven, como líneas muy finas que no aportan nada en la imagen, y por último se aplica la función imerode para eliminar los pequeños huecos blancos que pueden quedar en la línea negra que forman los límites del laberinto.

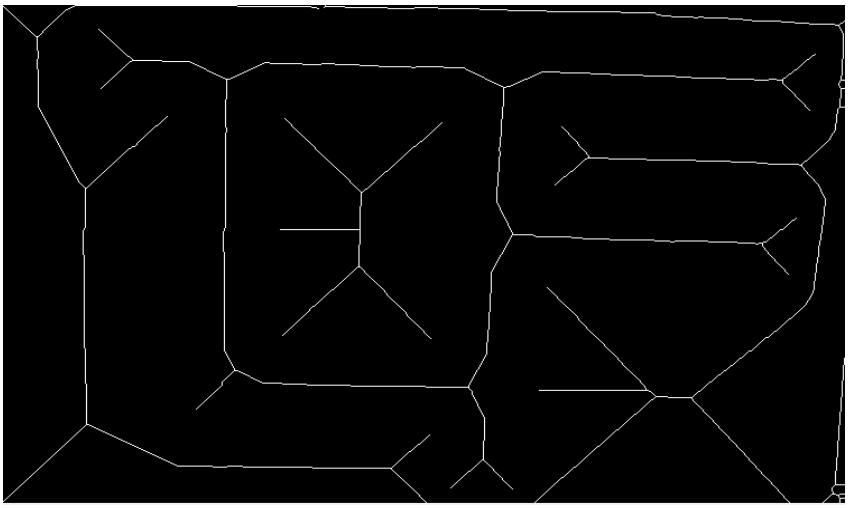


Figura 18.- Esqueleto de la imagen

Por último, se usa la función bwmorph con el parámetro ‘skel’, que realiza una operación de esqueletonización (o adelgazamiento) en la imagen binaria.

La esqueletonización es una operación que reduce los objetos en una imagen, en nuestro caso las líneas blancas dentro de la imagen, a una representación esquelética compuesta por líneas de un solo píxel. El resultado es una representación simplificada de la estructura de los objetos en la imagen.

Como se mencionó en el apartado de antecedentes y estado del arte, gracias a esto, podemos asemejar el laberinto de este proyecto a un grafo, en el que cada pixel es un vértice del grafo, por lo que se pueden aplicar algoritmos de resolución de laberintos basados en algoritmos de búsqueda dentro de un grafo.

4.2 Obtención de la ruta

Una vez procesada la imagen con la que se trabajará, se continua con la parte en la que se obtienen las coordenadas de inicio y de fin del laberinto. Las coordenadas de inicio y fin son seleccionadas por el usuario dentro de la interfaz de MATLAB con el mismo ratón haciendo clic dentro de la imagen, pero como se ha visto antes, la imagen sobre la que se trabaja es el esqueleto del laberinto, por lo que es muy difícil acertar a hacer clic y que coincida con un punto dentro del laberinto, por lo que una vez se hace clic dentro de la imagen

se procede a aproximar esa coordenada que se ha fijado a la coordenada más cercana al laberinto.

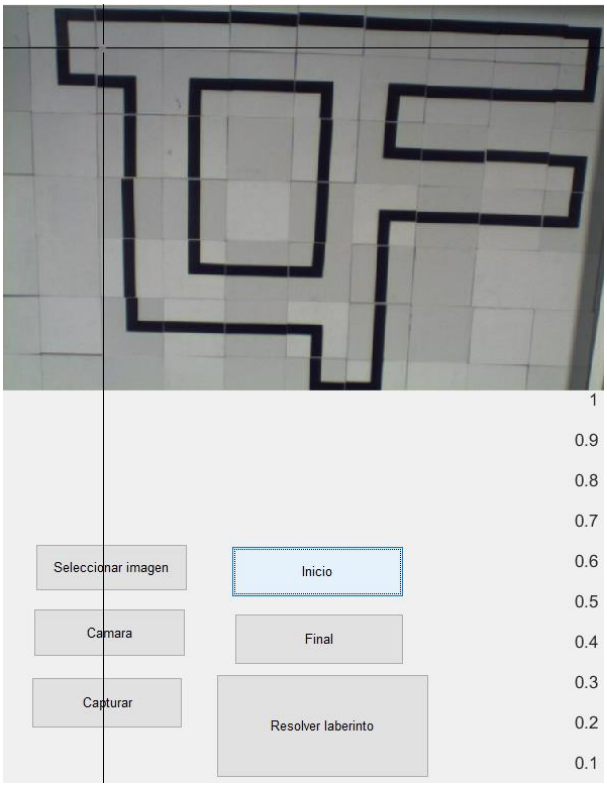


Figura 19.- Selección de la coordenada de inicio del laberinto

Como se puede ver en la Figura 19, con la función ginput se puede seleccionar una coordenada dentro de una imagen, en este caso la imagen tomada por la cámara que hay encima del laberinto. Para aproximar esta coordenada se pueden utilizar también algoritmos de búsqueda dentro de un grafo, ya que, en caso de hacer clic en una zona fuera del laberinto, que es un pixel negro, un 0 al ser una imagen binaria, se aproximaría al pixel más cercano a 1, que es parte del laberinto. En este caso se usará el algoritmo de Dijkstra, detallado a continuación:

I. Se definen las direcciones posibles para explorar los vecinos de un pixel en una matriz, que contienen los desplazamientos en el eje x y eje y de un pixel. Contiene 8 combinaciones de movimiento, en horizontal, vertical y diagonal.
II. Se inicializan dos matrices: ‘dist’, una matriz de distancias que almacena la distancia desde el punto de partida hasta cada píxel de la imagen y se inicia con todos los valores establecidos en

infinito, excepto el punto de partida que se establece a 0; y la matriz ‘visited’, que índice si un píxel ha sido visitado o no, que inicialmente está a 0. Ambas matrices con el tamaño de la imagen con la que se va a trabajar.

III. Comienza un bucle principal que se repetirá hasta que se visiten todos los píxeles o se encuentre el píxel objetivo.

IV. Se busca el píxel no visitado con la menor distancia. Se recorren todos los píxeles de la imagen y se encuentra el píxel con el valor de distancia más pequeño que aún no haya sido visitado y se almacena en las variables ‘min_x’ y ‘min_y’.

V. Se marca el píxel mínimo como visitado estableciendo el valor correspondiente en la matriz de visitados ‘visited’.

VI. Se verifica si este píxel es el objetivo, y si es así se establecen las coordenadas del píxel como resultado y se rompe el bucle inicial.

VII. En caso contrario se actualizan las distancias a los vecinos de ese píxel, se recorren todas las direcciones posibles y se calculan las coordenadas de los vecinos, verificando que están dentro de los límites y no han sido visitados.

VIII. Si se cumple la condición anterior, se calcula la distancia al vecino y si la distancia es menor que la distancia almacenada en la matriz ‘dist’ para ese vecino, se actualiza la matriz.

IX. Una vez actualizada la matriz de distancias, se vuelve al paso IV y se repite el proceso hasta encontrar el píxel objetivo o visitar todos los píxeles.

X. Finalmente, se devuelven las coordenadas del píxel objetivo encontrado, o en caso de que se hayan visitado todos los píxeles y no se haya encontrado el píxel objetivo, se devuelve como coordenadas el punto de origen.

De esta forma, se asegura que el punto de partida y el punto objetivo del laberinto esté dentro de los píxeles blancos, es decir, píxeles a 1 dentro de la imagen binaria sobre la que se va a trabajar. A partir de aquí teniendo el punto de entrada y de salida se puede continuar con la búsqueda del camino entre la entrada y salida del laberinto. Para ello se va a hacer uso de otro de los

algoritmos mencionados anteriormente, en este caso el algoritmo de búsqueda en anchura (*BFS – Breadth First Search*), que hace uso de las coordenadas de entrada, de salida y la matriz de 0 y 1 del laberinto ya obtenidos:

I. Se crea una matriz booleana llamada ‘visited’ del tamaño del laberinto para mantener un registro de los nodos que ya se han visitado inicialmente a false.
II. Se crea una cola ‘queue’ inicialmente con la posición inicial que almacenará los nodos que se deben explorar
III. Crea una matriz de celdas ‘paths’ del mismo tamaño del laberinto para almacenar rutas hacia cada nodo. Contiene vectores que representan el camino desde la posición inicial hasta cada celda del laberinto. Inicialmente solo contiene la ruta hacia la entrada.
IV. Comienza el bucle mientras la cola ‘queue’ no esté vacía
V. En cada iteración se obtiene el primer nodo de la cola y lo elimina, se obtienen sus coordenadas y se verifica si coincide con la posición objetivo. Si es así, significa que ha encontrado una ruta válida y la función devuelve el camino almacenado en ‘paths’ para esa posición. En el primer caso no será así ya que el primer nodo de la cola va a ser la coordenada de inicio.
VI. El nodo actual se marca como visitado en la matriz ‘visited’.
VII. Se obtienen los vecinos válidos del nodo actual, siendo posibles los píxeles que estén a 1 alrededor del actual y se itera sobre los vecinos.
VIII. Si un vecino no ha sido visitado, se actualiza su ruta en la matriz ‘paths’, concatenándola con la ruta al nodo actual.
IX. Marca el vecino como visitado en la matriz ‘visited’ y se agrega a la cola ‘queue’ para que se explore en futuras iteraciones.
X. El algoritmo continúa explorando nodos y actualizando rutas hasta que encuentra el objetivo o hasta que la cola esté vacía, lo que significa que no hay un camino válido al objetivo y se devuelve un camino vacío.

Una vez hecho esto, se obtiene una matriz de dos columnas correspondientes a las posiciones x e y de cada uno de los píxeles de la ruta a seguir y tantas filas como píxeles hay desde el inicio hasta el final del laberinto. Con esta ruta ya almacenada, se puede continuar con el envío de información de la ruta al robot.

4.3 Determinación de la posición y envío de la ruta al robot.

4.3.1 Orientación del robot.

Una vez obtenida la matriz de coordenadas desde la posición inicial hasta la posición final, hay que enviar esas coordenadas al robot. Para hacerlo, primero hay que conseguir que el robot pueda moverse de un punto a otro y una vez conseguido, enviar las coordenadas de la solución para que el robot vaya de punto a punto de la solución. Primero, se obtendrá con la cámara, la posición en todo momento del robot y además un punto auxiliar en la parte delantera del robot para conocer también la orientación del mismo. Con estos dos puntos se puede conseguir la orientación del robot con respecto al punto objetivo.

El primer paso es obtener las coordenadas de posición del robot y las coordenadas del punto auxiliar que nos ayudará a obtener la orientación. Para conseguir esto, se necesita que el robot tenga algo visible que se pueda detectar fácil con la cámara. En este caso serán tres leds colocados en la parte superior del robot como se puede ver en la Figura 20.

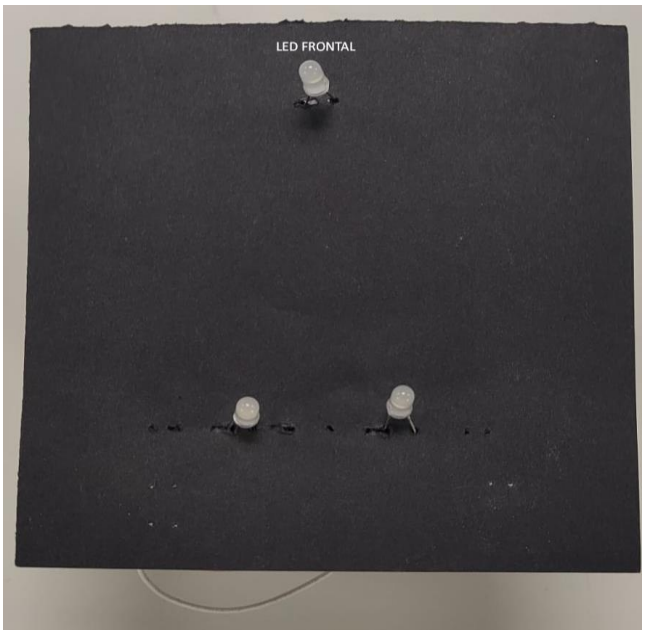


Figura 20.- Leds para determinar la posición del robot

Se colocan tres leds para conocer qué parte del robot es la delantera y cual la trasera, porque en el caso de que solo se colocaran dos leds, no sería posible diferenciar cuál es cuál debido a que los leds son iguales. De esta forma, la parte delantera correspondería al led que está sólo, y los otros dos que están más cerca el uno del otro corresponderían a la parte trasera del robot, siendo el punto medio entre ambos, el eje central de las ruedas. Para la obtención de las coordenadas, la exposición de la cámara se baja y se binariza la imagen. Así se consigue que la única información disponible en la imagen que se va a procesar sean los tres leds, por lo que es más fácil de obtener las coordenadas (Figura 21).

Figura 21.- Obtención coordenadas. a: Imagen original; b: Imagen con la exposición baja; c: Imagen binarizada

Ya con la imagen binarizada se procede a la obtención de las coordenadas siguiendo los siguientes pasos:

I. Se etiqueta y segmenta las regiones conectadas en la imagen binarizada con la función bwlabel y se calculan las propiedades de las regiones etiquetadas. En este caso se obtienen las coordenadas del centroide y el área de cada región.
II. Se define un valor de área mínima y se seleccionan las regiones cuya área sea mayor de este valor. Esto se hace para asegurarnos de que nos quedamos únicamente con las regiones que forman los leds blancos y no con regiones que pueden formar posibles defectos en la imagen.
III. Después se buscan las dos regiones más cercanas entre sí, calculando la distancia euclidiana entre los centroides de todas las posibles combinaciones de regiones, y se encuentras las coordenadas de las dos regiones con la menor distancia entre ellas.
IV. Se calcula el punto intermedio entre los dos centroides de las dos regiones cercanas y se almacenan para tener las coordenadas del robot.
V. Para obtener la coordenada restante, se usa la función setdiff que compara dos matrices y devuelve los elementos de la primera matriz que no están presentes en la segunda matriz. Por ello se comparan todas las regiones posibles con las regiones de las dos coordenadas más cercanas, para quedarse sólo con la coordenada restante y así poder almacenarla en una variable como coordenada auxiliar, siendo la parte frontal del robot.

Una vez hecho esto, se trazan dos vectores, uno que va desde las coordenadas del robot hasta la coordenada auxiliar para determinar la orientación del robot, y otro que va desde las coordenadas del robot hasta cada uno de los puntos de la solución que está en la matriz de solución del laberinto como se puede observar en la Figura 22.

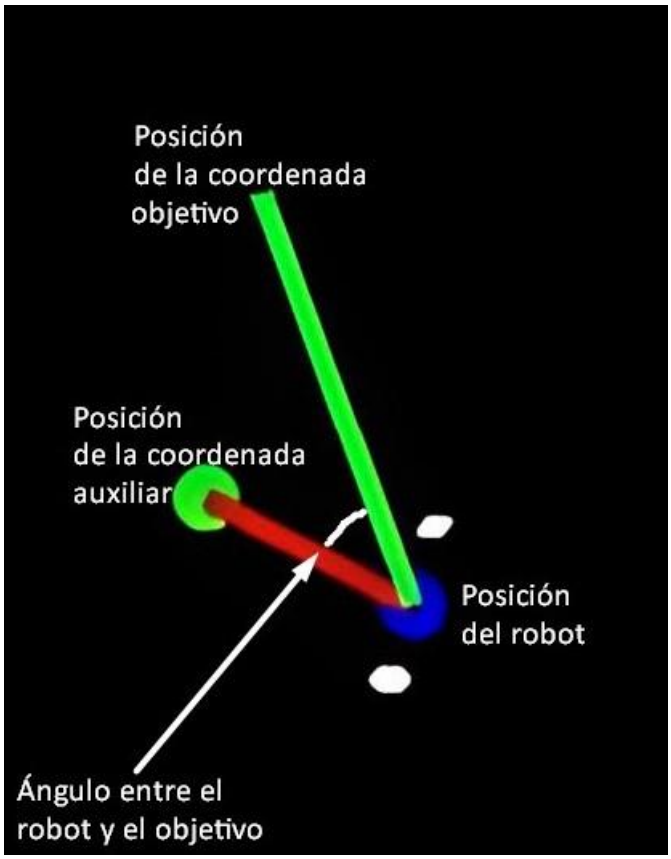


Figura 22.- Vectores de orientación del robot

Para calcular el ángulo que hay entre el robot y el objetivo se usará una función llamada vector2angle que nos dará el ángulo que hay entre un vector y el otro. Esta función nos dará el ángulo más pequeño que hay entre dos vectores, por lo que siempre dará un ángulo menor de 180 grados que no nos permitirá saber la orientación exacta. Para poder saber la orientación se calcula el producto vectorial de ambos vectores que nos dará un número negativo o positivo en función de la orientación con respecto al punto objetivo. Si el resultado es positivo, el robot está orientado a la izquierda con respecto al objetivo, por lo que tendría que girar a la derecha, y si el resultado es negativo, estará a la derecha con respecto al objetivo, por lo que tendrá que girar a la izquierda. Ahora que ya se sabe cómo está orientado el robot dentro del laberinto hay que pasar a mover los motores en función de la posición con respecto al punto objetivo. Se han destacado dos distintas soluciones para abordar este problema. La primera solución, donde se corrige la orientación del robot, el control se basa en detener el robot y girarlo hacia la dirección correcta cada vez que se supera el ángulo objetivo, que en este caso se espera que sea lo más cercano a cero posible. Y la segunda solución, en la que se propone un

control con corrección de ángulo en la que se ajusta la velocidad de los motores de manera proporcional al ángulo.

4.3.2 Control por Reglas con Umbral de Dirección (CRUD)

Cada uno de los dos tipos de control van a tener su propio programa de Micro:bit para controlar al robot y una serie de órdenes en MATLAB para enviar las ordenes correspondientes a la tarjeta. En este caso el programa de Micro:bit va a ser el siguiente:

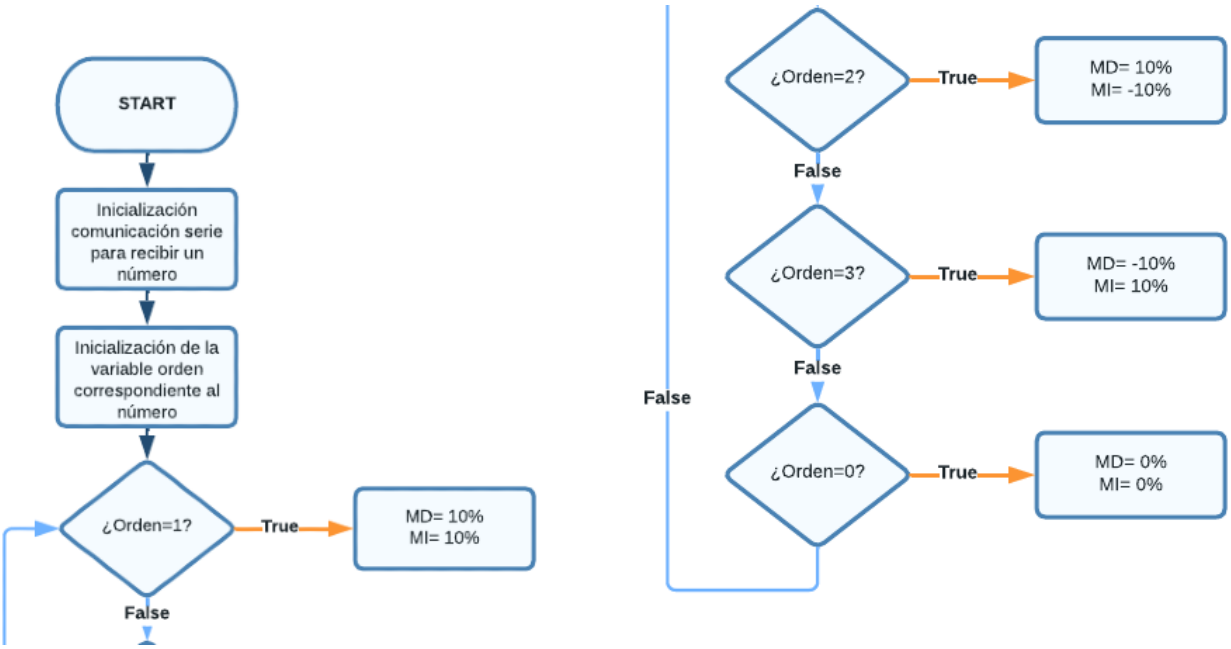


Figura 23.- Programa Micro:bit para Control por Reglas con Umbral de Dirección

Este será el programa encargado de recibir las órdenes desde MATLAB y actuar sobre los motores del robot. Para empezar, se inicializa la comunicación serie del robot abriendo el puerto con los pines a los que está conectado el módulo Bluetooth, los pines P0 y P1, y se establece un baud rate de 9600, que también se establecerá en el programa de MATLAB. Se inicializa una variable llamada orden que servirá para almacenar los datos recogidos por Bluetooth.

Una vez recibido el número, se actuará sobre los motores en función del número recibido:

- Si el número recibido es un ‘1’, se actuará sobre los motores estableciendo la misma velocidad para ambos, es decir, el robot irá hacia adelante.

- Si el número recibido es un ‘2’, se actuará sobre los motores estableciendo la misma velocidad, pero con distinta orientación, es decir, al motor derecho se le establece una velocidad positiva y al izquierdo una velocidad negativa, por lo que el robot girará sobre su propio eje en sentido antihorario, es decir, hacia la izquierda.
- Si el número recibido es un ‘3’, se actuará sobre los motores estableciendo la misma velocidad, pero con distinta orientación, esta vez, al motor izquierdo se le establece una velocidad positiva y al derecho una velocidad negativa, por lo que el robot girará sobre su propio eje en sentido horario, es decir, hacia la derecha.
- Si el número recibido es un ‘0’, se actuará sobre ambos motores estableciendo una velocidad cero, es decir, parando ambos motores.

Para el programa de MATLAB que se encargará de enviar las órdenes a la tarjeta de Micro:bit tenemos lo siguiente:

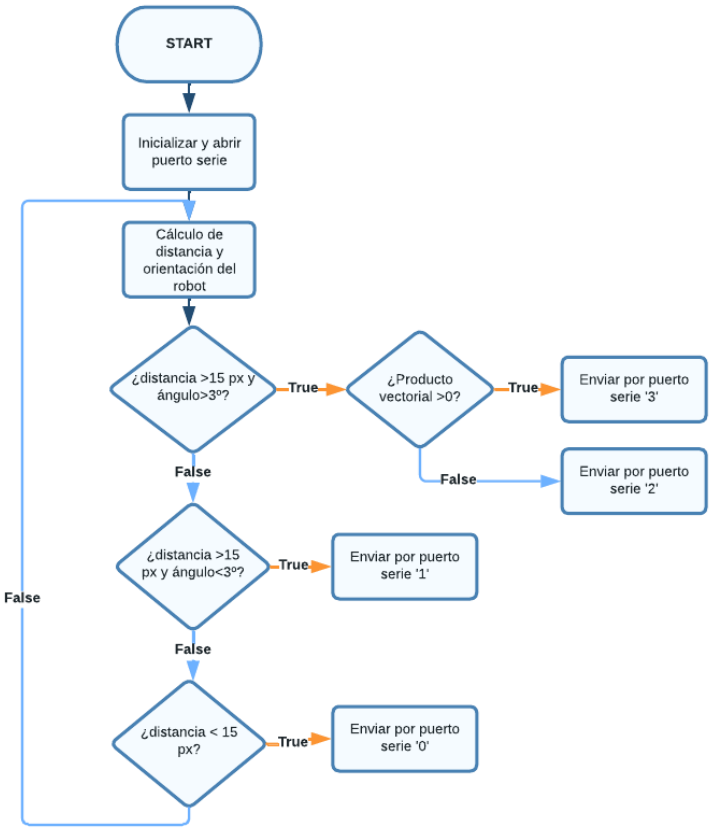


Figura 24.- Programa de MATLAB encargado de enviar las órdenes

Se inicializa y se abre el puerto serie con un baudrate de 9600 y el puerto COM correspondiente, que es necesario comprobar al conectar el módulo bluetooth al ordenador. Una vez inicializado se calcula la distancia y la orientación del robot tal y como se ha calculado anteriormente y se decide que se enviará por puerto serie en función de la distancia y la orientación:

- Si la distancia es mayor a un umbral, en este caso 15 píxeles y además el ángulo es mayor a otro umbral, en este caso 3º, se comprobará el signo del producto vectorial encargado de determinar que orientación tiene el robot con respecto al punto de destino. Si el producto vectorial es positivo, se enviará por puerto serie un '3', correspondiente al giro horario, hacia la derecha. Si el producto vectorial es negativo, se enviará por puerto serie un '2', correspondiente al giro antihorario, hacia la izquierda.
- Si la distancia es mayor a 15 píxeles y además el ángulo es menor a 3º, en este caso, el robot está orientado en frente del punto objetivo, por lo que se enviará por puerto serie un '1' correspondiente a que ambos motores tienen la misma velocidad y sentido, por lo que irá hacia adelante.
- Si la distancia es menor a 15 píxeles, es decir, ha llegado a uno de los puntos objetivo, o ha llegado al último punto de la matriz, por lo que se enviará un '0' correspondiente a establecer la velocidad 0 en ambos motores.

Este método nos asegura que el robot va a corregir el ángulo en cualquier momento, por lo que va a ir en la dirección correcta siempre con un error pequeño. El inconveniente de este método es la velocidad de resolución del laberinto, ya que al no haber movimiento lineal cada vez que corrige la orientación, el tiempo de resolución del laberinto puede ser muy alto.

4.3.3 Control por Reglas con Potencia Proporcional (CRPP)

En el caso anterior, para corregir la orientación del robot, el robot gira sobre sí mismo para corregir el ángulo y después avanza cuando el ángulo es menor a un umbral. Es decir, para corregir la trayectoria, el robot no tiene velocidad lineal, sólo velocidad angular, estableciendo a ambas ruedas la

misma velocidad, pero con distinto sentido, por lo que cada vez que se corrige el ángulo, el robot se queda quieto girando sobre sí mismo.

En este caso, siempre habrá velocidad lineal, pero se controla la velocidad de una de las ruedas en función de la orientación del robot, usando el producto vectorial como en el caso anterior. Dependiendo de la orientación, una de las dos ruedas está a una velocidad máxima fijada y el valor de la otra rueda será el valor de la velocidad máxima fijada restado a un factor que varía en función del ángulo de orientación del robot con respecto al punto objetivo.

Para empezar, el programa que tiene la tarjeta Micro:bit es distinto como se puede ver en la Figura 25, ya que esta vez no sólo se espera un único valor por el puerto serie. En este caso, se envían por puerto serie dos datos que corresponden a las velocidades de las dos ruedas.

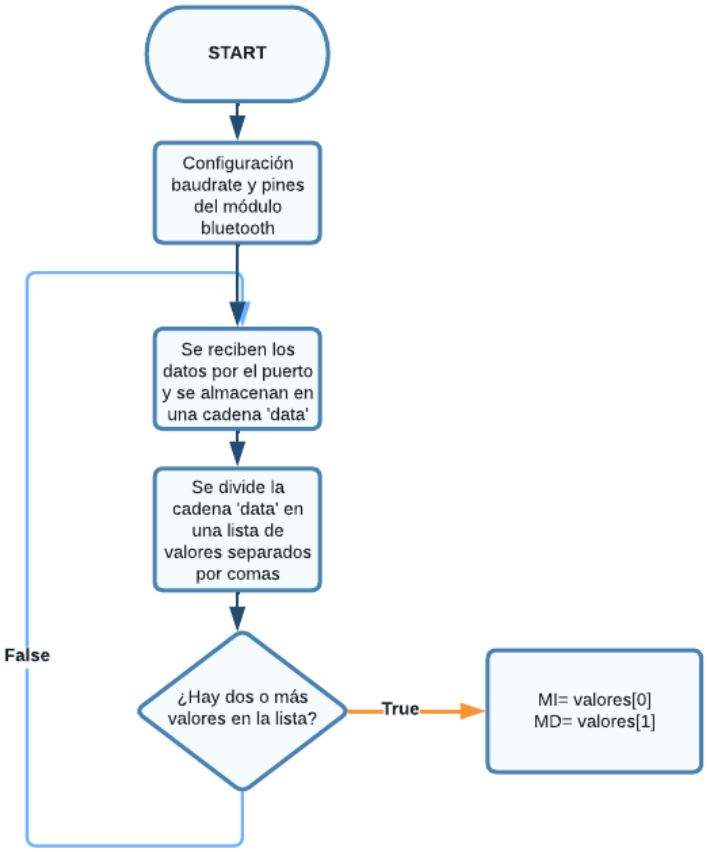


Figura 25.- Programa Micro:bit para control de motores

Para este caso, el programa que hay dentro de la tarjeta Micro:bit es más sencillo. Se inicializa la comunicación serie del robot abriendo el puerto con los pines a los que está conectado el módulo Bluetooth, los pines P0 y P1, y se establece un baud rate de 9600. Se reciben los datos por el puerto serie y se almacenan en una cadena llamada 'data'. Esa cadena se divide en una lista de valores separados por una coma, que es como se envían los datos en MATLAB, una cadena de dos datos separados por coma. Por último, se comprueba si esa lista de valores tiene al menos dos o más elementos en la lista. Si se cumple la condición se establece al motor izquierdo el valor de la lista en la posición 0 y al motor derecho el valor de la lista en la posición 1. En este caso, a diferencia del anterior, no se establece ninguna potencia a los motores dentro del programa de Micro:bit, ese valor se enviará desde MATLAB, por lo que no es necesario hacer el control dentro de la tarjeta, en MATLAB se procesará la información para enviar la potencia.

Ahora hay que configurar el programa de MATLAB para enviar las potencias de los motores en función del ángulo:

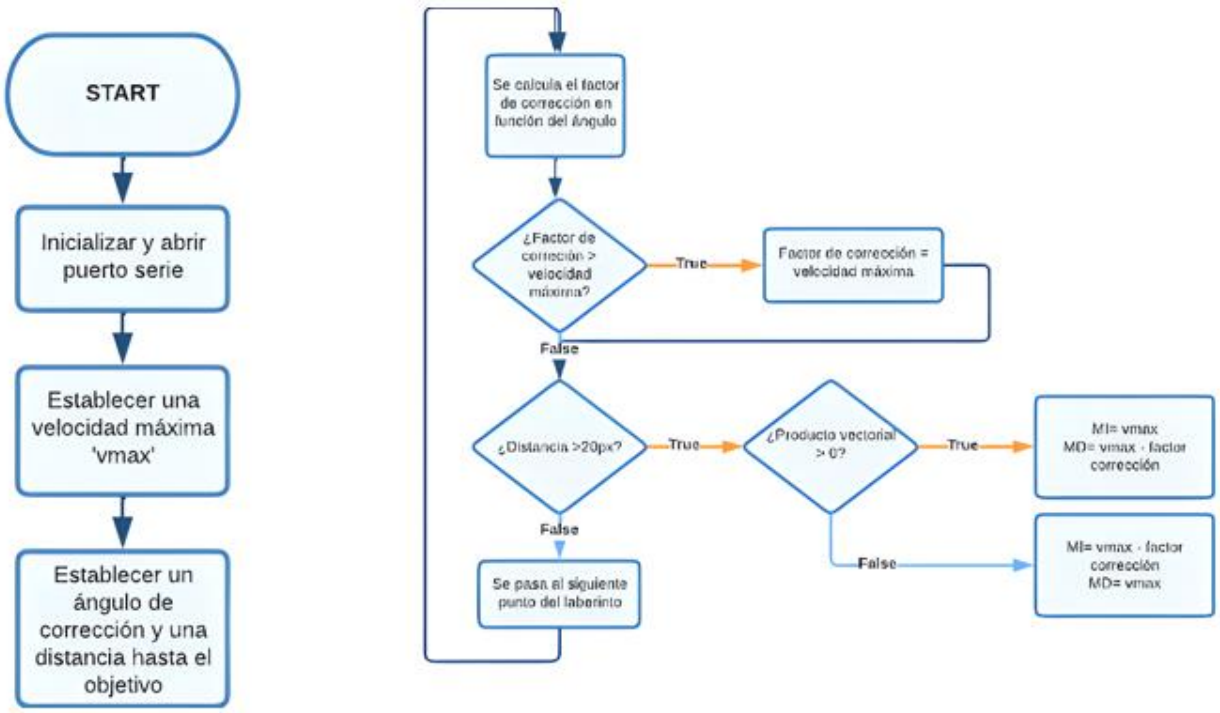


Figura 26.- Programa de MATLAB encargado de enviar la velocidad

Para empezar, se inicializa el puerto serie tal y como se configuró antes con un baudrate de 9600 y el puerto COM correspondiente. Se establece una velocidad máxima que puede ir de 0 a 255, que es el valor correspondiente a la potencia que se puede establecer a cada uno de los motores. Después se establece un ángulo de corrección, que en este caso va a ser de 45º. Este ángulo de corrección hace que el robot empiece a corregir la posición lo antes posible, por lo que, si el ángulo es menor, el robot empezará a corregir antes. Este ángulo puede ir de 0 a 180 grados, que es el valor máximo al que puede estar el robot orientado con respecto al punto objetivo.

Como se ha explicado anteriormente, cada una de las ruedas tendrá una potencia en función de su orientación con respecto al punto objetivo, que dependerá de a qué lado tiene que girar. Una de ellas tendrá una potencia máxima fijada que no varía, y la otra tendrá esa misma potencia restada de un valor 'factor de corrección', que varía dependiendo de los grados a los que esté el robot del punto objetivo. Este valor no puede ser superior a la velocidad máxima, que será su mayor valor. Para conseguir esto se hace una relación entre la velocidad y la orientación del robot. El máximo valor de la velocidad es 'vmax' y el máximo valor en grados al que puede estar orientado el robot es a 180º, que se representa en la Figura 27.

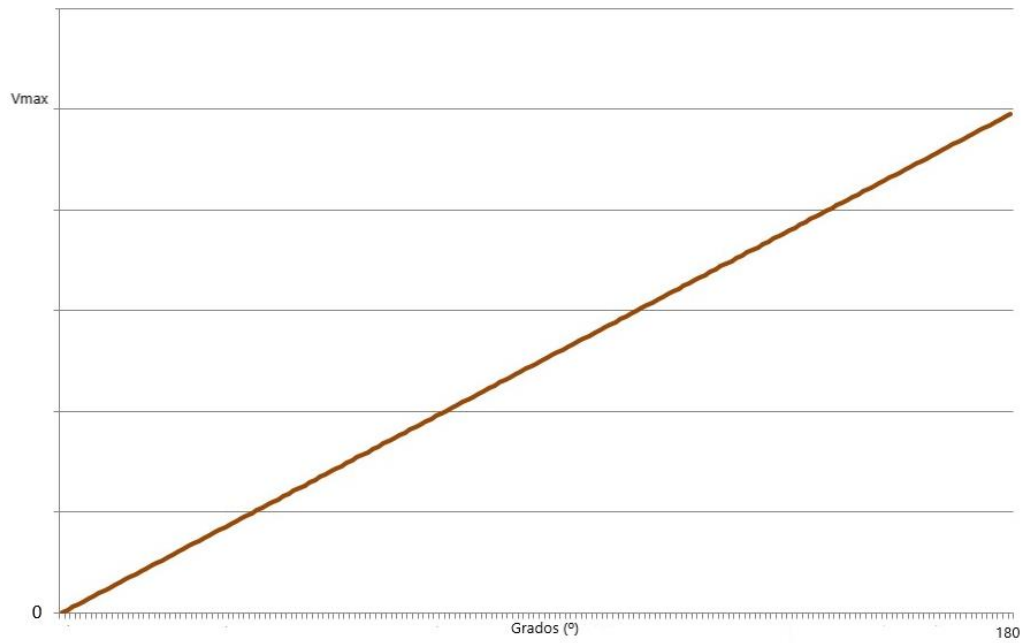


Figura 27.- Representación del factor de corrección como la velocidad en función del ángulo

Como se puede apreciar, cuanto más cerca está de 180º, más se acerca al valor de la velocidad máxima, lo que haría que al restarse a sí misma, al estar a 180º, la rueda estaría quieta, por lo que el robot giraría sobre la otra rueda. El factor de corrección quedaría así:

$$factor\ corrección = \frac{Vmax}{ángulo\ máximo\ de\ orientación} * ángulo\ actual$$

Siendo en este caso el ángulo máximo de orientación 180º, y el ángulo actual dependerá de cómo esté el robot orientado en ese momento. El problema de esto es que, si la orientación del robot con respecto al siguiente punto es muy pronunciada, el robot puede tender a hacer una curva hacia el punto objetivo, lo que sería problemático a la hora de moverse dentro del laberinto, ya que puede pisar las líneas que forman los muros del laberinto.

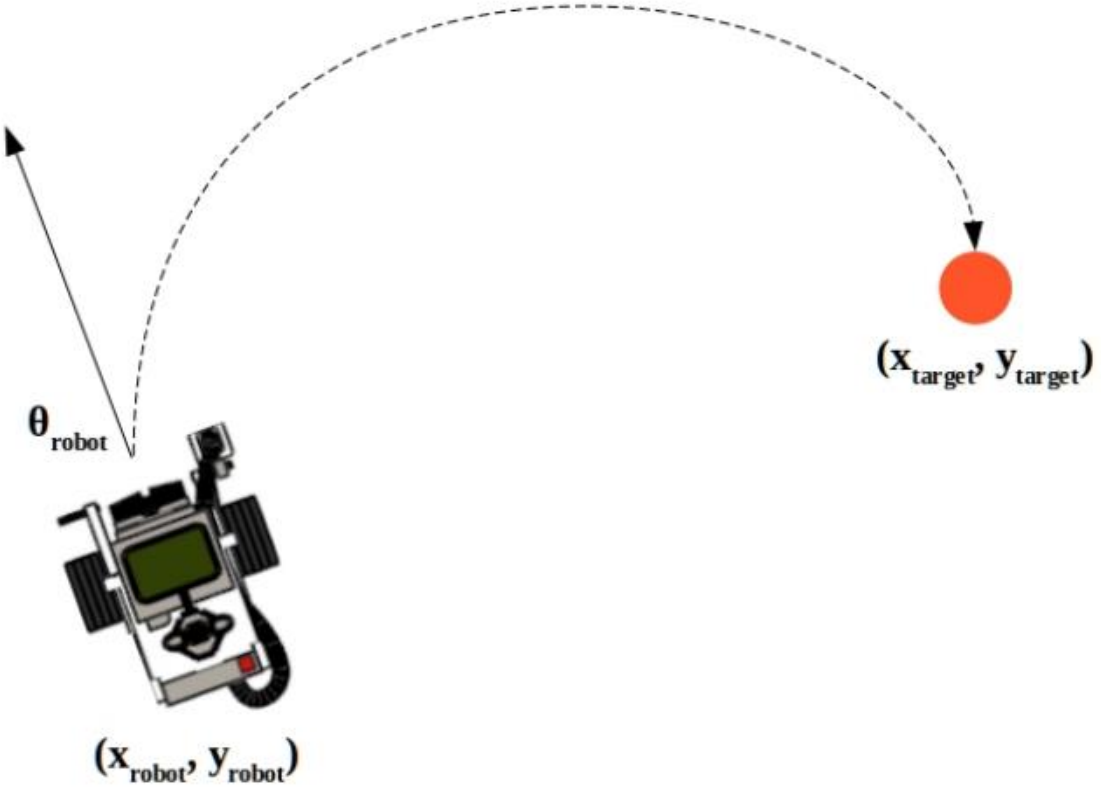


Figura 28.- Representación del movimiento de un punto a otro

Para solucionar esto, se puede disminuir el ángulo al que el robot empieza a corregir la posición, es decir, que el factor de corrección llegue a ser el valor de la velocidad máxima a un valor menor de 180º. Para este caso, se

elegirá 45º, como se puede ver en la Figura 29, para que la corrección sea más estricta a partir de 45º.

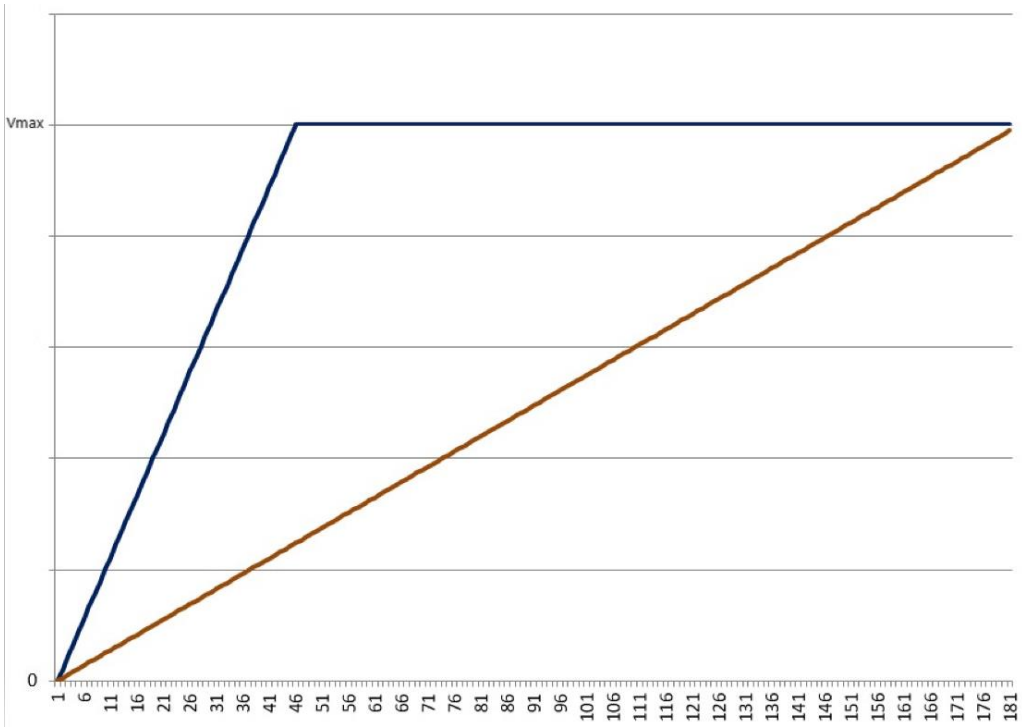


Figura 29.- Factor de corrección con ángulo de 45º

Con esto se consigue que la pendiente de la recta sea mayor, por lo que el factor de corrección incrementará más rápido y en el momento en que llegue a 45º y se alcance la velocidad máxima, mantenerlo en ese valor hasta 180º.

Una vez calculado el factor de corrección, se comprueba la distancia a la que está del punto objetivo, si aún no ha alcanzado el punto, se comprueba la orientación del robot igual que en el apartado anterior, con el producto vectorial del vector de orientación y el vector del robot al punto.

Si el producto vectorial es positivo, el robot tiene que girar a la derecha, es decir, está orientado a la izquierda con respecto al punto, por lo que la potencia del motor izquierdo se fija a la velocidad máxima y el motor derecho será esa velocidad menos el factor de corrección:

$$MI = Vmax$$

$$MD = Vmax - factor\ de\ corrección$$

Si el producto vectorial es negativo, al contrario que el anterior, el robot tiene que girar a la izquierda, es decir, está orientado a la derecha con respecto al punto, por lo que la potencia del motor izquierdo será la velocidad máxima fijada anteriormente menos el factor de corrección y el motor derecho estará fijado a la velocidad máxima:

$$MI = Vmax - factor\ de\ corrección$$

$$MD = Vmax$$

Si la distancia entre el robot y el punto objetivo es menor de un valor, en este caso 20 píxeles, se pasa al siguiente punto. Una vez alcanzado el último punto, se establece a 0 ambos motores y el robot habrá llegado al final del laberinto.

5 Resultados

Para poder comparar los resultados, se han puesto a prueba los métodos de resolución explicados anteriormente con distintos laberintos como se puede ver en la Figura 30, a los que se denominarán Laberinto 1, Laberinto 2 y Laberinto 3.

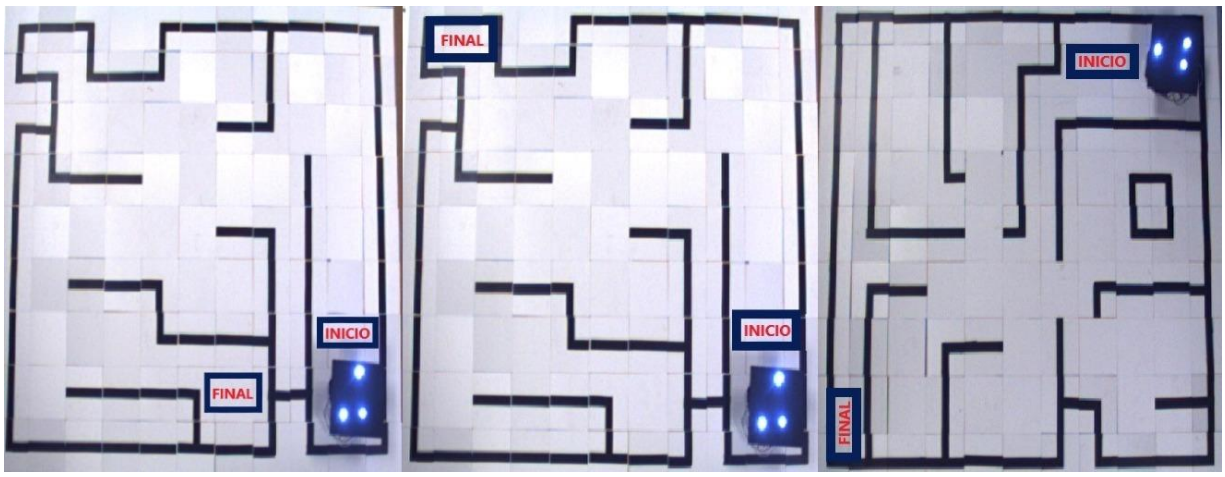


Figura 30.- Laberintos puestos a prueba; a: Laberinto 1; b: Laberinto 2; c: Laberinto 3

Como se mencionó antes, para la resolución con los sensores integrados en el robot, el algoritmo usado para la resolución del laberinto es el algoritmo de la mano derecha o seguidos de paredes. Quedando el camino para cada uno de los laberintos presentados en la figura anterior como se muestra en la Figura 31.

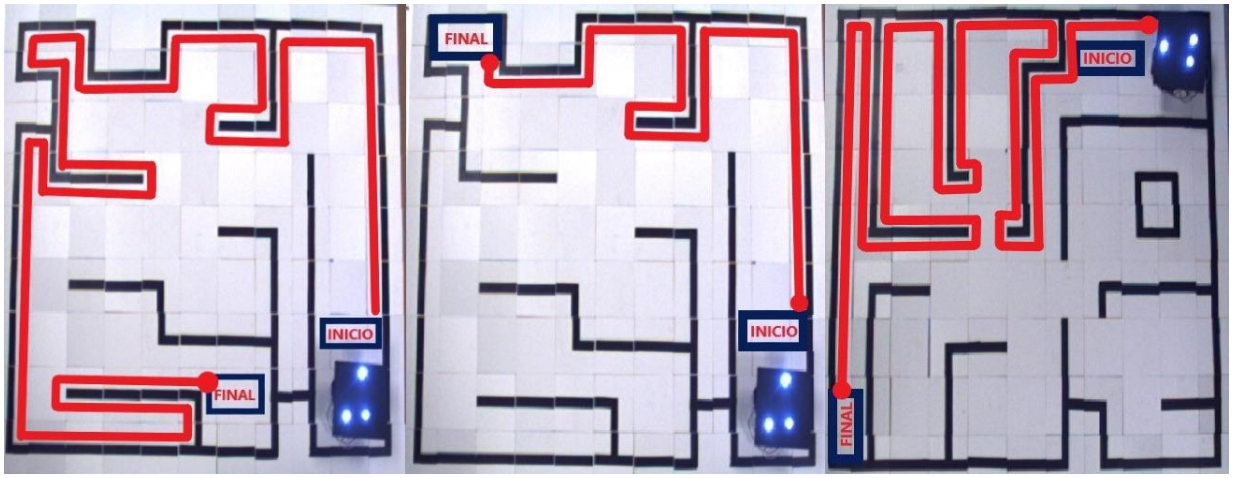


Figura 31.- Caminos resueltos con algoritmo de la mano derecha

Así es como quedaría el camino que tiene que recorrer el robot siguiendo el algoritmo de la mano derecha usando sólo los sensores incorporados en el robot.

Para la resolución del laberinto integrando sensores de visión por computador, independientemente del tipo de control por reglas que se aplique para el movimiento del robot, el algoritmo utilizado es el algoritmo de búsqueda en anchura (*BFS – Breadth First Search*), quedando el camino para cada uno de los laberintos como se muestra en la Figura 32.

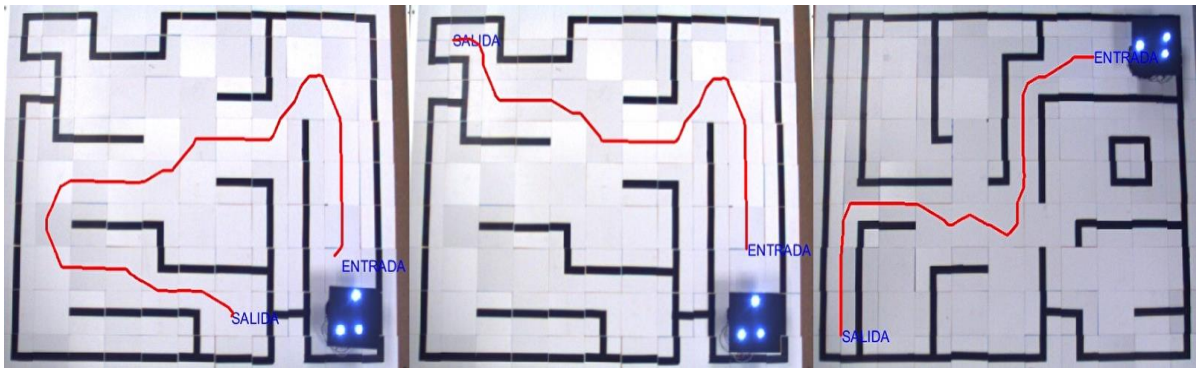


Figura 32.- Caminos resueltos con el algoritmo de búsqueda en anchura

Como se puede comparar comprobando ambas imágenes, con el algoritmo de búsqueda en anchura, se encuentra el camino más corto entre ambos puntos, y con el algoritmo de la mano derecha dependiendo mucho de donde esté colocada la entrada y salida, puede salir un camino más corto o más largo.

Para poder comparar mejor los resultados, se pone el robot en marcha y se cronometran los tres laberintos para cada uno de los tres métodos explicados en este proyecto: el algoritmo de la mano derecha haciendo uso de los sensores integrados en el robot; el algoritmo de búsqueda en anchura usando el Control por Reglas con Umbral de Dirección (CRUD), en el que el robot detiene su velocidad lineal y solo tiene velocidad angular para corregir su orientación; y el algoritmo de búsqueda en anchura usando el Control por Reglas con Potencia Proporcional (CRPP), en el que se corrige la orientación del robot sin necesidad de detener su velocidad lineal.

El primer laberinto que se pondrá a prueba es el Laberinto 1 mencionado anteriormente en la Figura 30, que se muestra en la siguiente figura:

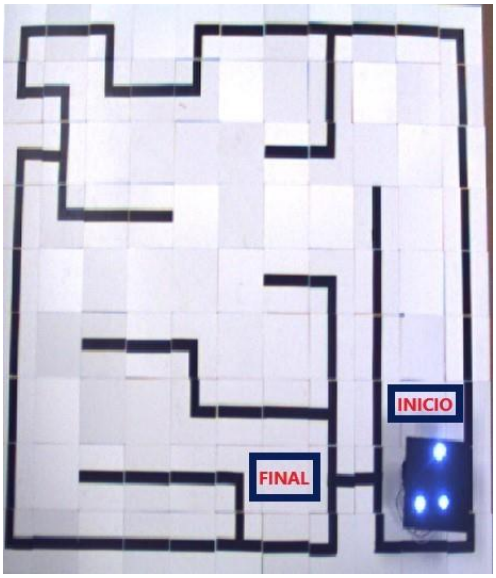


Figura 33.- Laberinto 1

Se ponen a prueba los tres métodos mencionados anteriormente en este laberinto y se cronometra cuanto tarda cada uno en llegar a la meta.

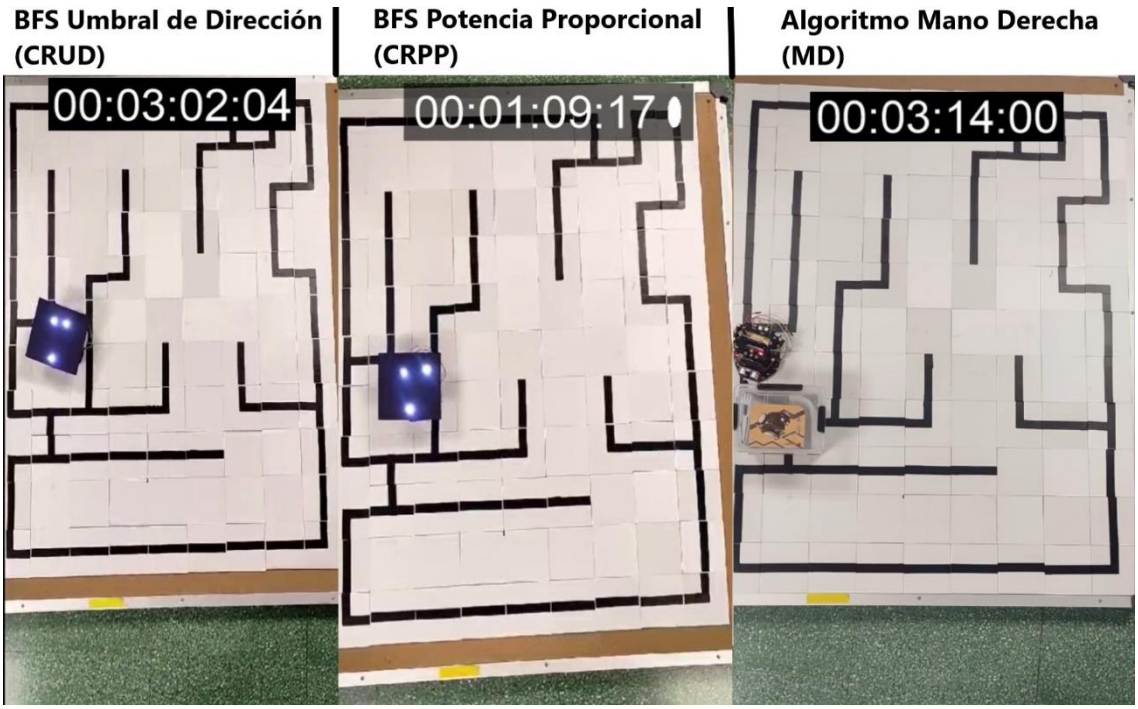


Figura 34.- Robot cronometrado para el Laberinto 1

En este laberinto, por orden, el primero en terminar es el algoritmo de búsqueda en anchura con el Control por Reglas con Potencia Proporcional, que

termina en 1 minuto y 9 segundos, seguido del algoritmo de búsqueda en anchura con el Control por Reglas con Umbral de Dirección, con un tiempo de 3 minutos y 2 segundos. Por último, el Algoritmo de la Mano Derecha con un tiempo de 3 minutos y 14 segundos. En este primer laberinto se puede comprobar que la mejor opción es el Control por Reglas con Potencia Proporcional por mucha diferencia con respecto a los otros dos, y que en los dos últimos no hay mucha diferencia entre ambos, aunque el último sea el algoritmo de la mano derecha.

La siguiente prueba se realizará con el Laberinto 2 mostrado en la siguiente figura:

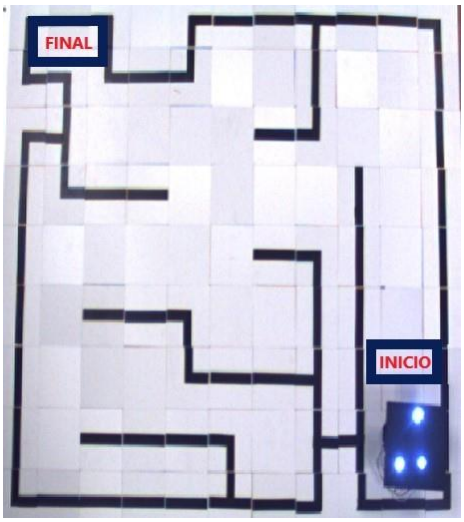


Figura 35.- Laberinto 2

Los resultados obtenidos para este laberinto son los mostrados en la siguiente figura:

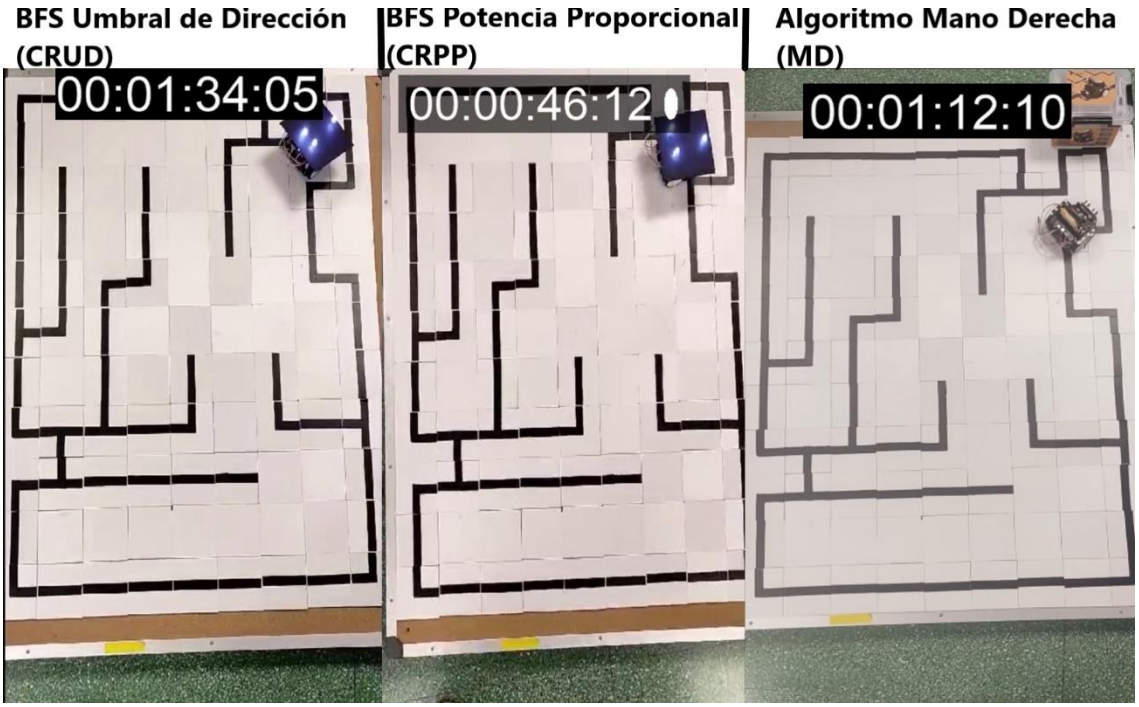


Figura 36.- Robot cronometrado para el Laberinto 2

Para este segundo laberinto, el primero en terminar, con un tiempo de 46 segundos ha sido de nuevo el algoritmo de búsqueda en anchura con el Control por Reglas con Potencia Proporcional. El cambio esta vez ha sido que el Algoritmo de la Mano Derecha terminó segundo con un tiempo de 1 minuto y 12 segundos, seguido del algoritmo de búsqueda en anchura con el Control por Reglas con Umbral de Dirección con un tiempo de 1 minuto y 34 segundos.

Este es un claro ejemplo de la importancia de la elección de la pared que se va a seguir en el algoritmo de la mano derecha, puesto que, si hubiéramos aplicado el algoritmo, pero con la pared izquierda, el robot habría tardado mucho más en encontrar la salida, que siguiendo la pared derecha. También se puede apreciar que el Control por Reglas con Umbral de Dirección a veces puede ser muy lento ya que no cuenta con velocidad lineal al corregir el ángulo, lo que hace que aumente su tiempo de resolución.

Para finalizar, una última prueba que se realizará con el Laberinto 3, mostrado en la siguiente figura:

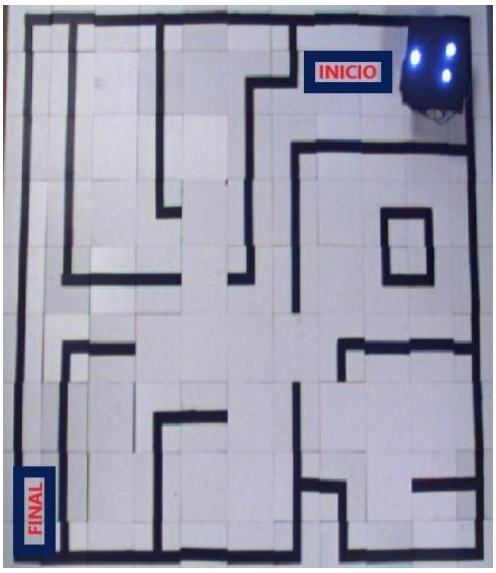


Figura 37.- Laberinto 3

Para este último laberinto, los resultados se muestran en la Figura 38:

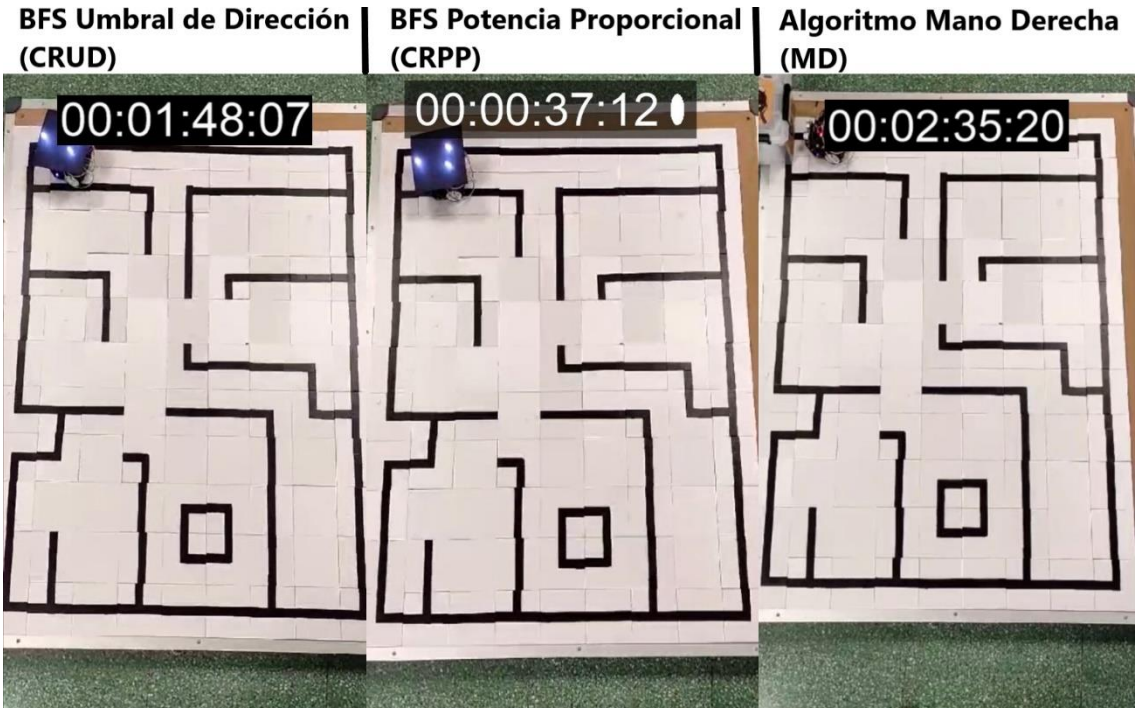


Figura 38.- Robot cronometrado para el Laberinto 3

Para este último caso, el primero en terminar vuelve a ser como en los otros casos y por bastante diferencia, el algoritmo de búsqueda en anchura con el Control por Reglas con Potencia Proporcional, con un tiempo de 37 segundos, seguido esta vez del algoritmo de búsqueda en anchura con el Control por Reglas con Umbral de Dirección con un tiempo de 1 minuto y 48

segundos. Y en este caso el Algoritmo de la Mano Derecha queda el último con un tiempo de 2 minutos y 35 segundos.

En este caso también se puede comprobar como el Algoritmo de la Mano Derecha o seguidor de pared depende mucho de que pared se elija al comienzo del laberinto, ya que la diferencia de tiempo que hay entre lo que tarda al algoritmo de la mano derecha con los demás es mucha, porque tiene que recorrer el laberinto entero para llegar a la salida. En caso de haber ido por la pared izquierda habría tardado mucho menos.

Para tener un vistazo más rápido y poder sacar una conclusión de los resultados se comparan los tiempos que ha tardado cada uno de los métodos usados en los 3 laberintos en las siguientes tablas con los tiempos y un podio de velocidad de resolución:

Tabla 1: Tiempos de resolución de cada método

	LABERINTO 1	LABERINTO 2	LABERINTO 3
Algoritmo de la Mano Derecha	3 minutos y 14 segundos	1 minuto y 12 segundos	2 minutos y 35 segundos
BFS con CRPP	3 minutos y 2 segundos	1 minuto y 34 segundos	1 minuto y 48 segundos
BFS con CRUD	1 minuto y 9 segundos	46 segundos	37 segundos

Tabla 2: Orden en que termina cada método

	LABERINTO 1	LABERINTO 2	LABERINTO 3
Algoritmo de la mano derecha	3º	2º	3º
BFS con CRPP	2º	3º	2º
BFS con CRUD	1º	1º	1º

6 Conclusiones

En conclusión, si lo que se compara es la ruta más corta obtenida, el uso de sensores de visión por computador siempre encontrará la ruta más corta desde la entrada hasta la salida, y, por lo tanto, el uso del algoritmo de búsqueda en anchura, ya que los sensores de visión por computador ofrecen datos muy necesarios para encontrar la ruta más corta, tales como la posición de entrada y de salida del laberinto. Por el contrario, sólo con el uso de los sensores incorporados en el robot, esa información no se puede obtener, por lo que se tiene que hacer uso del método de la mano derecha, el cual depende mucho de la posición de entrada y de salida del laberinto, ya que se pueden dar situaciones en las que encuentre una ruta corta muy parecida a la obtenida por el algoritmo de búsqueda en anchura y otras veces que encuentra una ruta más larga. Por lo que la integración de sensores de visión por computador es la mejor opción siempre que se quiera recorrer la ruta más corta de un punto a otro, independientemente del método de Control por Reglas usado, ya que facilita la incorporación de algoritmos de resolución de laberintos en los que se conoce la posición de entrada y salida.

Sin embargo, si lo que se desea comparar es la velocidad de resolución del robot dentro del laberinto, esto cambia ligeramente. Como se ha visto en las pruebas realizadas anteriormente, este aspecto depende mucho del método de Control por Reglas que se utilice para el envío de la ruta al robot una vez obtenida mediante sensores de visión por computador.

El método de Control por Reglas con Potencia Proporcional (CRPP) es el que da los mejores resultados en cuanto a velocidad de resolución de los 3 métodos utilizados, por lo que se puede volver a confirmar, que el uso de sensores de visión por computador es una mejor opción con respecto al uso sólo de los sensores incorporados en el robot.

Pero si el método de Control por Reglas usado es el de Umbral de Dirección (CRUD), en el que el robot corrige su orientación solo con velocidad angular, esto no es así. Si el laberinto tiene una solución simple, en la que el Algoritmo de la Mano Derecha encontraría un camino similar al que se

obtendría con el algoritmo de búsqueda en profundidad, puede darse la situación en que el Algoritmo de la Mano Derecha, sea el método con mayor velocidad contra el método de Control por Reglas con Umbral de Dirección, ya que este método demora mucho tiempo al corregir su orientación, debido a que el robot sólo tiene velocidad angular al momento de corregir su trayectoria, por lo que el robot se mueve más lento dentro del laberinto, aunque la trayectoria sea la más corta.

Por lo tanto, en este proyecto, el método más eficaz es el uso del algoritmo de búsqueda en anchura usando sensores de visión por computador y el envío de la ruta usando el método de Control por Reglas con Potencia Proporcional. Es el método que obtiene la ruta más corta desde la entrada a la salida y además en el que el robot tarda menos en resolver el laberinto.

7 Referencias

Bienias, Ł., Szczepanski, K., & Duch, P. (2016). Maze exploration algorithm for small mobile platform. *Image Processing & Communications, 21*(3), 15-26. doi:10.1515/ipc-2016-0013

DFR Robot. (s.f.). Obtenido de Maqueen Plus V2: https://wiki.dfrobot.com/SKU_MBT0021-EN_Maqueen_Plus_STEAM_Programming_Educational_Robot

Institute for the Future of Education . (s.f.). Obtenido de https://observatorio.tec.mx/edu-news/educacion-stem-que-es-y-como-sacarle-provecho/

Matemáticas Digitales. (s.f.). Obtenido de http://www.matematicasdigitales.com/como-escapo-de-este-laberinto/

microlog. (s.f.). Obtenido de https://www.micro-log.com/microbit/3560-microbit-v2.html#:~:text=Micro%3Abit%20V2%20es%20una,con%20otras%20placas%20micro%3Abit.

Planells, A. (2021). Los senderos lúdicos que se bifurcan: el juego como mundo y el. *Arte, Individuo y Sociedad*, 1131-5598. doi:https://dx.doi.org/10.5209/aris.68408

Real Academia Española. (13 de abril de 2023). *Diccionario de la lengua española, 23.ª ed., [versión 23.6 en línea].* Obtenido de https://dle.rae.es/laberinto

Tomás Mariano, V. T., Núñez Cárdenas, F. d., & Andrade Hernández, E. (2018). Análisis de algoritmos de búsqueda en espacio de estados. *Ciencias Huasteca Boletín Científico de la Escuela Superior de Huejutla*, 2-4. doi:10.29057/esh.v3i5.1089

Anexo I. Interfaz de usuario

Para el uso del algoritmo de la mano derecha, no es necesario explicación muy complicada, simplemente cargar el programa necesario en la placa Micro:Bit, conectarla al robot Maqueen Plus V2, poner un objeto en la salida para que el sensor ultrasónico pueda detectarlo y poner el robot en la entrada del laberinto pegado a la pared derecha. Este apartado está destinado más bien al uso de la interfaz, el código para ambos métodos de control por reglas y todo lo relacionado con el uso de sensores de visión por computador relacionados con este proyecto.

Antes de empezar hay que saber cuál de los dos controles se desea ejecutar, ya que hay un apartado del programa distinto para cada uno de los dos métodos, siendo cada uno los mostrados en las siguientes figuras:

```
401                 %------------Control por reglas paso a paso------------%
402 -                if ((abs(solx-xrobot) > 15) || (abs(soly-yrobot)>15)) & deg >=3
403 -                    if producto_cruz > 0
404 -                        result.Value = '3';
405 -                        fprintf(s, '%s\n', result.Value);
406 -                        robot_moved = false;
407 -                    end
408 -                    if producto_cruz < 0
409 -                        result.Value = '2';
410 -                        fprintf(s, '%s\n', result.Value);
411 -                        robot_moved = false;
412 -                    end
413 -                end
414 -                if ((abs(solx-xrobot) > 15) || (abs(soly-yrobot)>15)) & deg < 3
415 -                        result.Value = '1';
416 -                        fprintf(s, '%s\n', result.Value);
417 -                        robot_moved = false;
418 -                end
419 -                if (abs(solx-xrobot) < 15) & (abs(soly-yrobot)<15)
420 -                        result.Value = '0';
421 -                        fprintf(s, '%s\n', result.Value);
422 -                        robot_moved = true;
423 -                end
424                 %--------------------------%
```

Figura 39.- Control por Reglas con Umbral de Dirección

```
%-----------------Control por reglas de velocidad proporcional al ángulo--------------------%
vmax=30;
angulo_correccion=45;
correccion = (vmax/angulo_correccion)*abs(deg);
distancia=20;
if correccion > vmax
    correccion = vmax;
end
 if ((abs(solx-xrobot) > distancia) || (abs(soly-yrobot)>distancia))
     if producto_cruz > 0
         velocidadIzquierdo = vmax;
         velocidadDerecho = round(vmax-(correccion));
         data = [num2str(velocidadIzquierdo), ',', num2str(velocidadDerecho)];
         fprintf(s, '%s\n', data);
         robot_moved = false;
     end
     if producto_cruz < 0
         velocidadDerecho = vmax;
         velocidadIzquierdo = round(vmax-(correccion));
         data = [num2str(velocidadIzquierdo), ',', num2str(velocidadDerecho)];
         fprintf(s, '%s\n', data);
         robot_moved = false;
     end
 end
 if ((i<length(solution)) & (abs(solx-xrobot) < distancia) & (abs(soly-yrobot)<distancia))
         robot_moved = true;
 end
%----------Condición de parada control por reglas de velocidad proporcional al ángulo-----------%
    velocidadDerecho = 0;
    velocidadIzquierdo = 0;
    data = [num2str(velocidadIzquierdo), ',', num2str(velocidadDerecho)];
    fprintf(s, '%s\n', data);
```

Figura 40.- Control por Reglas con Potencia Proporcional

Una vez que se ha elegido cuál de los dos métodos de control se va a usar, el programa ya está listo para ejecutarse. Pero antes de ejecutar, primero hay que encender el robot y una vez hecho ejecutar el programa para poder establecer la conexión por puerto serie con el robot, una vez establecida se abrirá la interfaz, y se coloca al robot en la entrada del laberinto que es seleccionada por el usuario.

Figura 41.- Robot situado en la entrada del laberinto elegida por el usuario

Ya que el robot está colocado correctamente en la entrada del laberinto se puede pasar a interactuar con la interfaz que se ha abierto al establecer conexión con el robot.

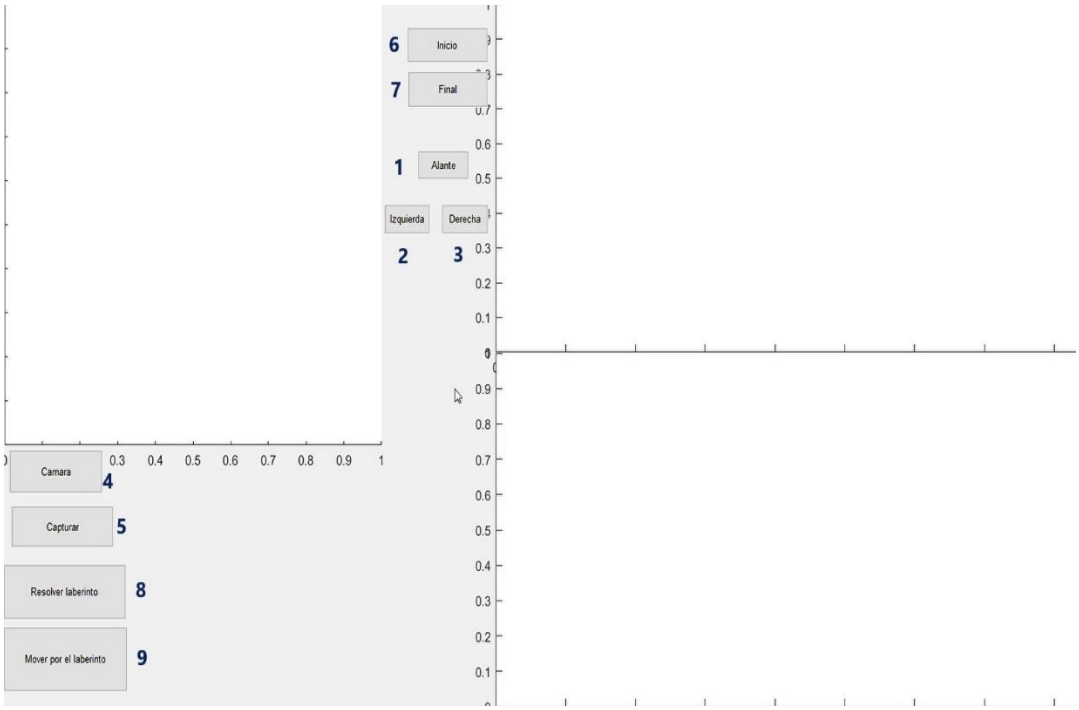


Figura 42.-Interfaz de usuario

Esta es la interfaz de usuario que se abre en cuanto se establece conexión con el robot y el programa es ejecutado. Como se puede ver dispone de varios botones que se explican a continuación y tres figuras que sirven para mostrar la información necesaria para la ejecución del programa. Los tres primeros botones sirven para mover el robot hacia delante, izquierda y derecha. Aunque el robot ya está colocado en la entrada del laberinto, son unos botones que pueden ser útiles si se quisiera corregir la posición sin necesidad de mover al robot físicamente. A partir de aquí se explican los botones necesarios para que el robot se mueva a través del laberinto.

En primer paso, el hay que seleccionar el botón 4 llamado Cámara, que mostrará en la figura de la izquierda lo que está enfocando la cámara que hay encima del laberinto.

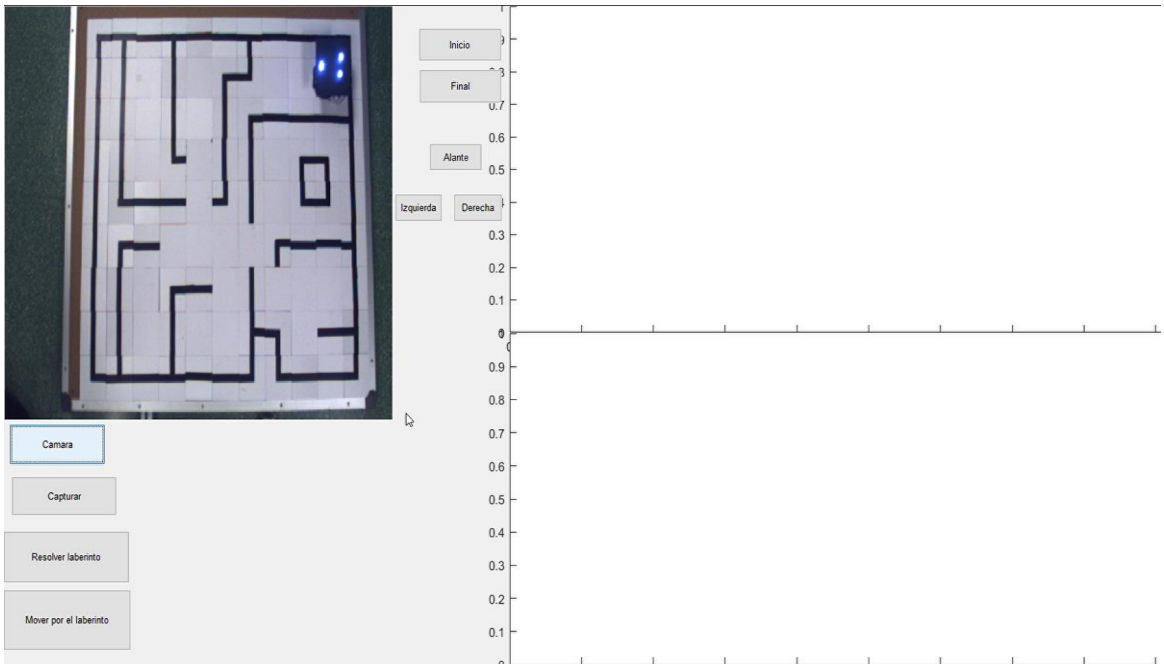


Figura 43.-Botón cámara

Una vez que se ha pulsado el botón Cámara, se pulsa en el botón 5, llamado Capturar, que servirá para capturar la imagen que hay en la figura de la izquierda para poder a continuación seleccionar la entrada y salida del laberinto. Para eso, una vez capturada la imagen, con los botones 6 y 7 se selecciona el inicio y final del laberinto.

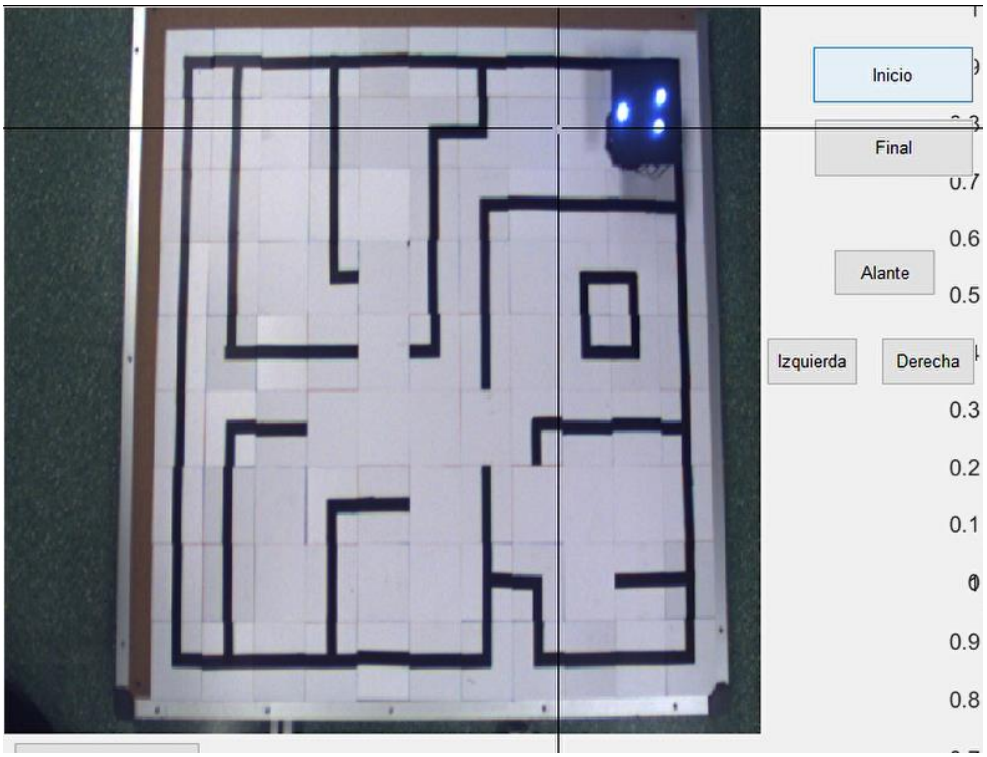


Figura 44.-Selección entrada del laberinto

Al pulsar el botón se abre un cursor con el que se puede seleccionar dentro de la figura. Para seleccionar la entrada, hay que pulsar justo delante de donde se ha colocado el robot, para que sea el primer punto al que vaya el robot. Una vez seleccionada la entrada, al pulsar el botón de Final, también se abre un cursor para seleccionar la salida dentro del laberinto. Una vez seleccionadas, se pulsa el botón 8 llamado Resolver laberinto, que al pulsar muestra en la figura de arriba a la derecha el camino que deberá tomar el robot para encontrar la salida.

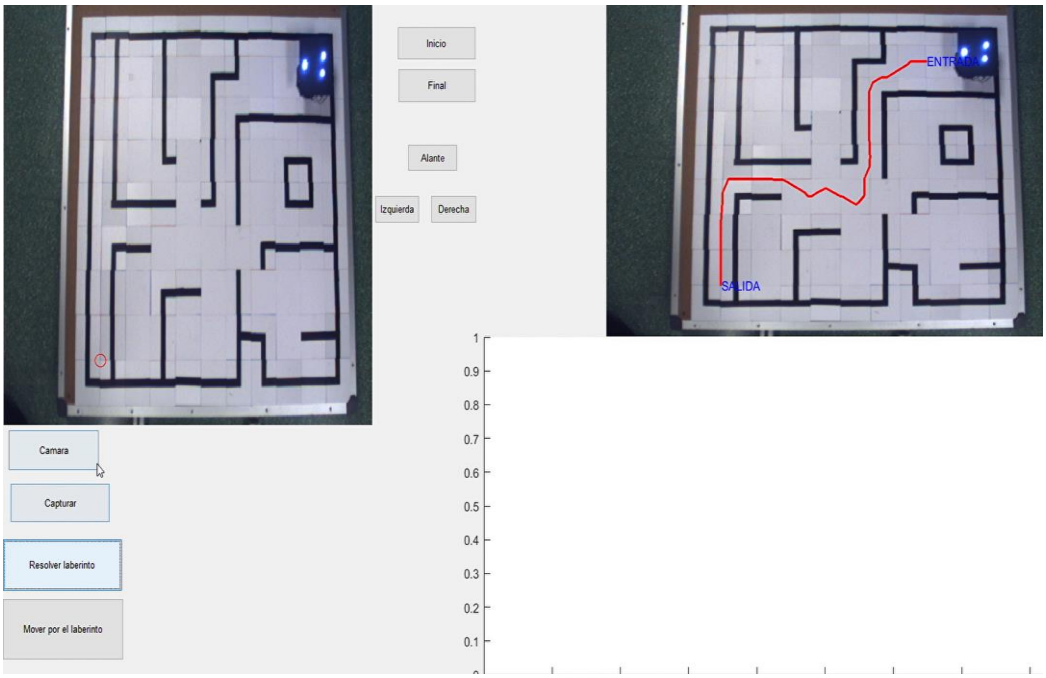


Figura 45.- Ruta obtenida al pulsar el botón Resolver laberinto

Una vez pulsado, en la figura de arriba a la derecha se muestra la ruta obtenida aplicando el algoritmo de búsqueda en anchura, y los puntos que debe seguir el robot para ir de la entrada a la salida. Una vez hecho, se vuelve a pulsar en el botón de Cámara y para finalizar en el botón Mover por el laberinto.

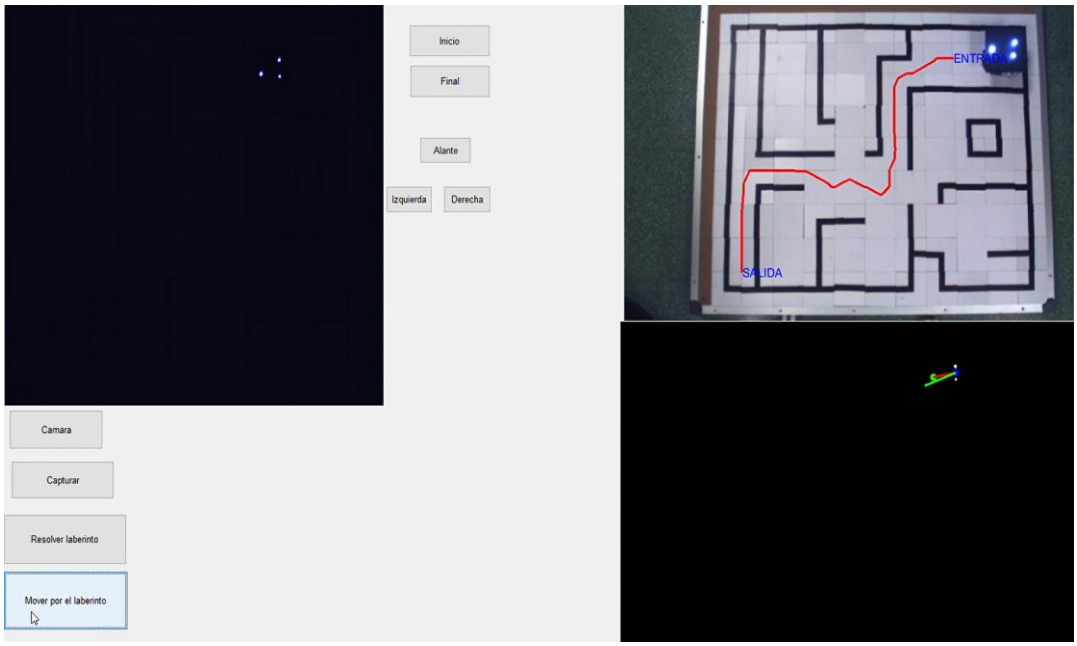


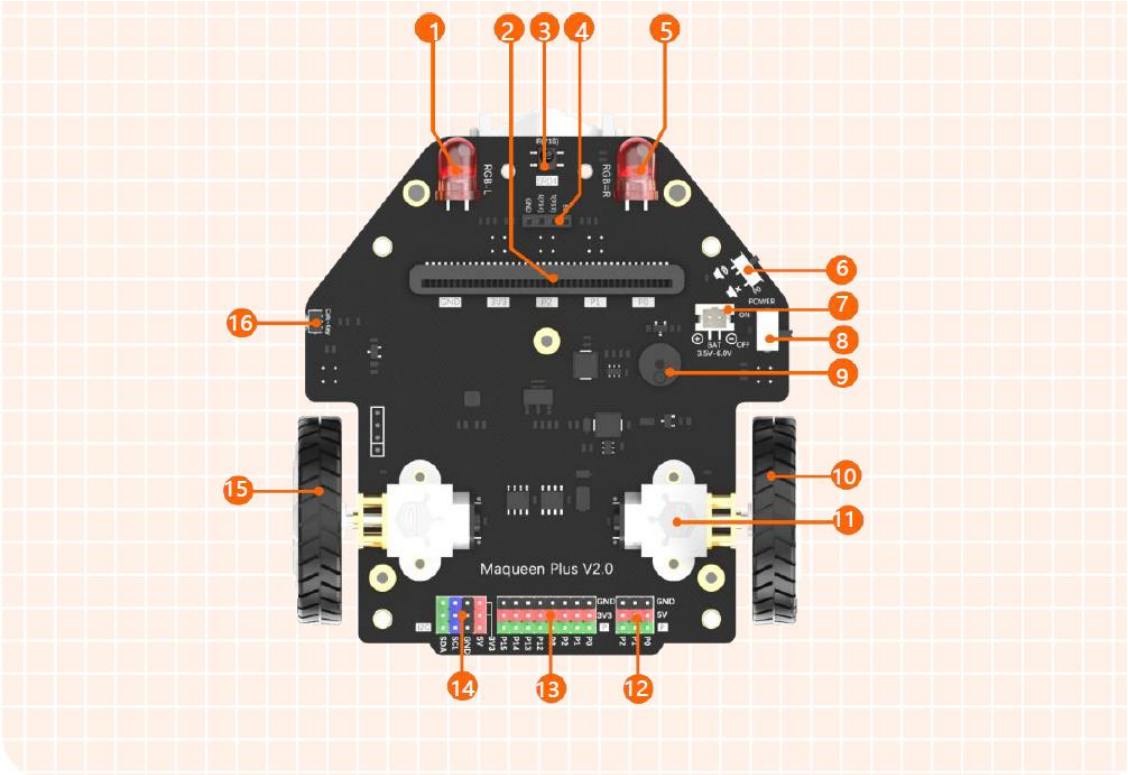
Figura 46.- Robot moviéndose a través del laberinto

Ya pulsado el botón, en la figura de la izquierda se ven los 3 leds necesarios para controlar la posición y la orientación del robot, en la figura de arriba a la derecha la ruta que tiene que seguir el robot y abajo a la derecha los vectores que forman los 3 puntos utilizados para la orientación del robot: el punto medio del robot, el punto auxiliar y el punto al que tiene que llegar.

Anexo II. Hojas de caraterísticas

Maqueen Plus V2

Maqueen Plus V2 is the upgraded version of the STEAM Maqueen Plus educational robot, offering a variety of enhanced features to strengthen its capabilities, more flexible and easier to use. It supports graphical coding platforms like MakeCode and Mind+, on which we can program it to make various interesting projects by simply dragging and snapping blocks.



1	RGB-LED-L	2	micro:bit socket	3	Infrared receiver	4	Ultrasonic sensor port	5	RGB-LED-R	6	Buzzer switch
7	Power supply port	8	Power supply switch	9	Buzzer	10	Right wheel	11	Metal motor	12	Servo port
13	I/O port	14	I2C port	15	Left wheel	16	Line-tracking reset key				

Figura 47.- Vista frontal Maqueen Plus V2

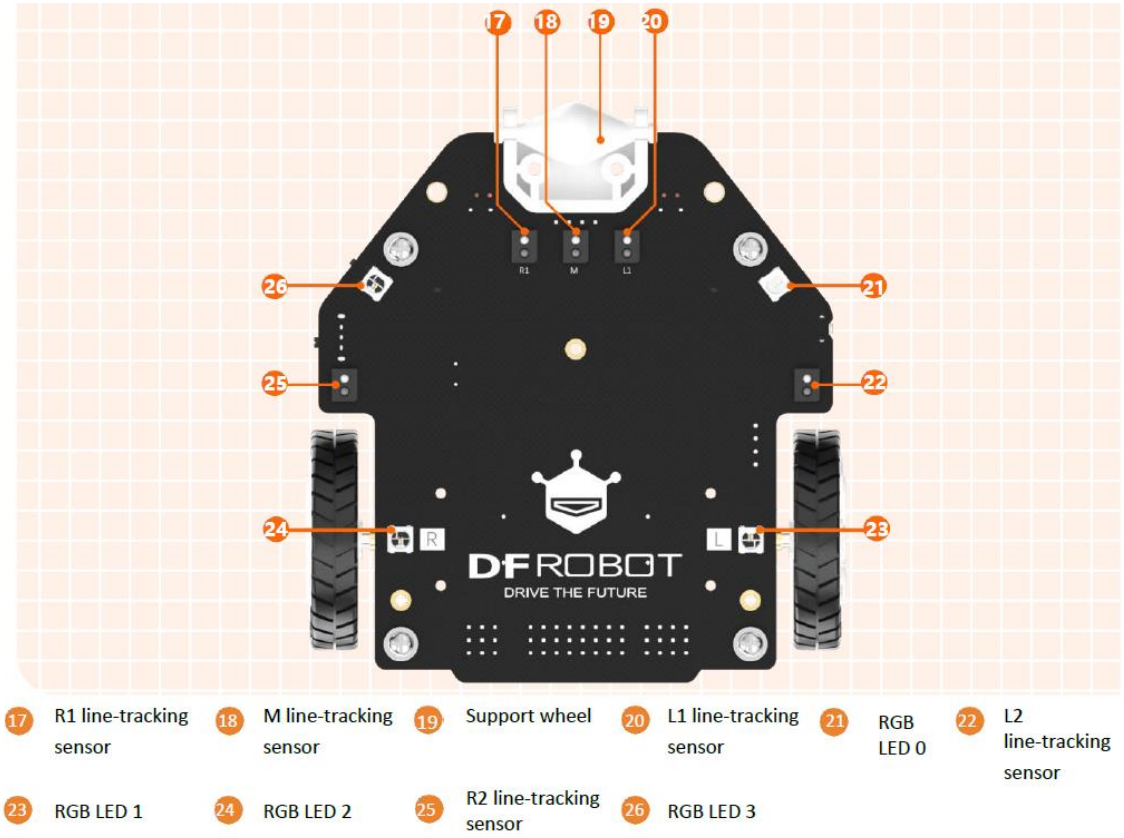


Figura 48.- Vista trasera Maqueen Plus V2

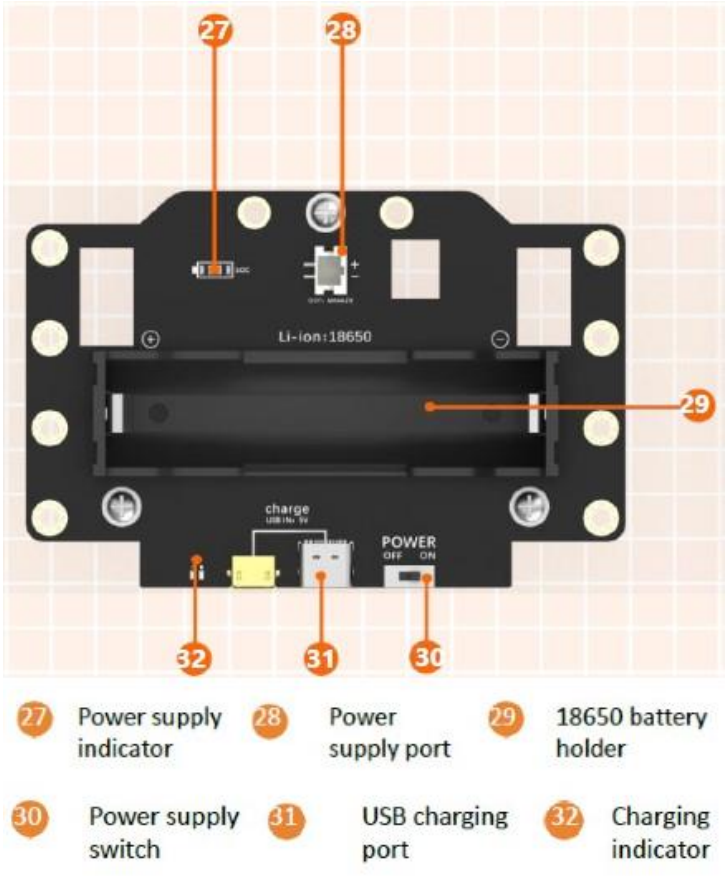


Figura 49.- Información general para estuche de pilas 18650

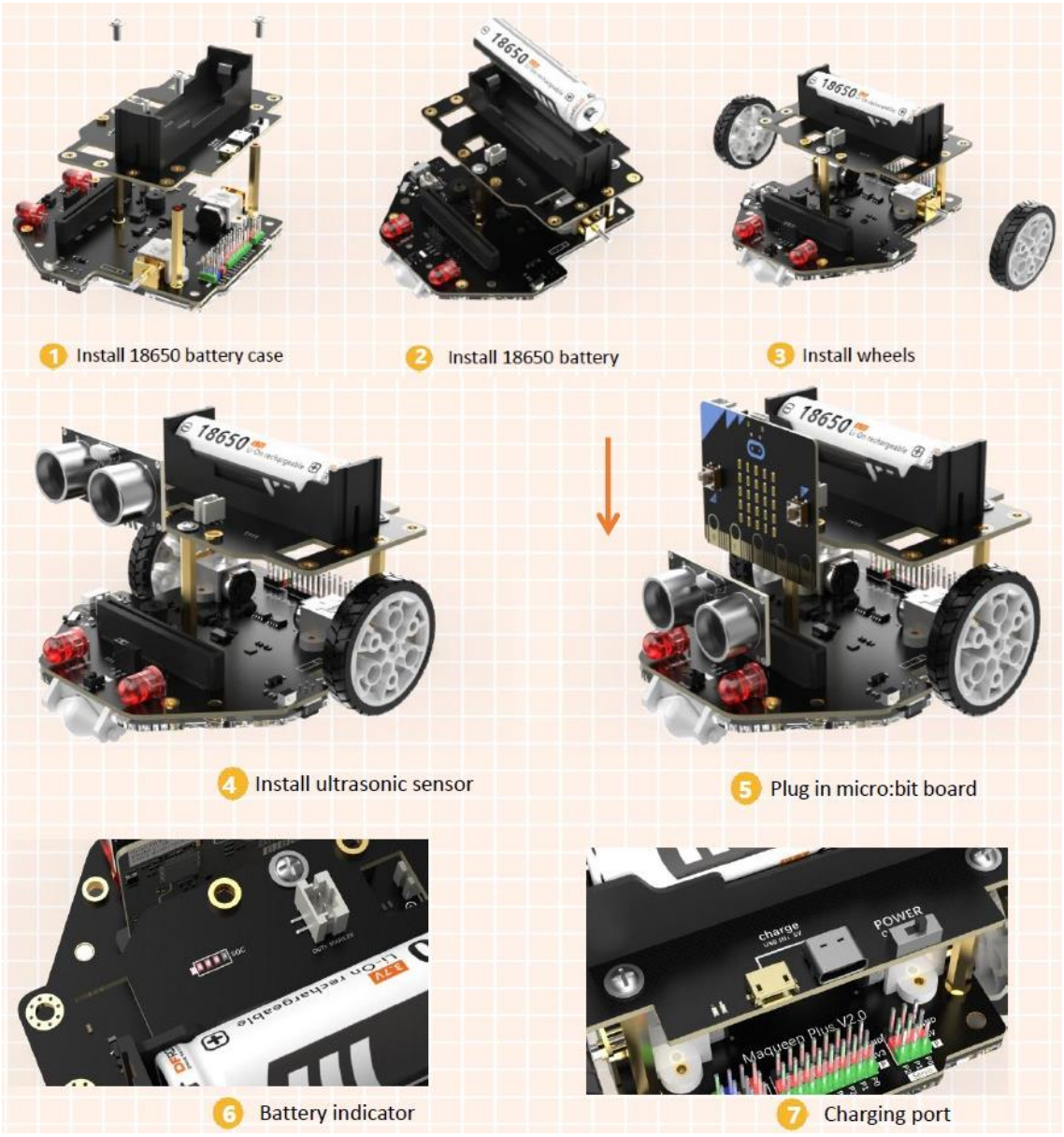


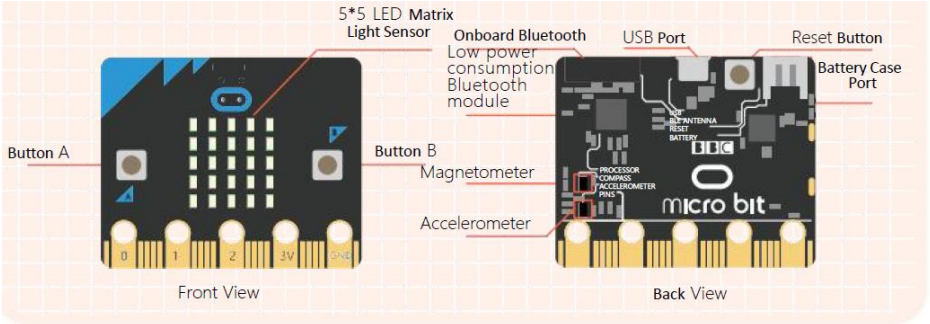
Figura 50.- Diagrama montaje Maqueen Plus V2 con batería 18650

Micro:Bit

The micro:bit can be programmed to do various interesting things. For example, it can interconnect with automobiles via onboard Bluetooth, thus the automobile can send commands to control the MCU. Multiple micro:bit boards can communicate with each other via radio to complete the tasks requiring remote control. Also, it can be a navigation device as it features an on-board compass to track the direction of the wearer. And there are 5 × 5 LED matrix on the board that can display funny patterns and animation, and two programmable buttons that can be used to control game-play or pause/play a track.

In addition, micro:bit is equipped with commonly-used sensors like light sensor and temperature sensor. So, it can be widely used in light-related projects and applications requiring temperature detection, and so on.

On the credit card size board (5cm*4cm/1.97*1.57") there are 5 × 5 programmable LED matrix, two programmable buttons, light sensor, accelerometer, compass, temperature sensor, Bluetooth module and other electric modules.



5×5 Programmable LED Matrix	There are 25 programmable LED lights on micro:bit which can display patterns, words, and numbers.
2 Programmable Buttons	Used separately or together to make things happen. For example, press down A to display a heart pattern.
Light Sensor	The 25 LEDs can act as light sensors to measure how much light is falling on the micro:bit.
Accelerometer and Compass	Measure the gestures or forces in 3 dimensions, such as shaking, tilting, free fall.
Temperature Sensor	Detect the temperature in the current environment.
Bluetooth & Radio	Your micro:bit can communicate with other micro:bits by radio, and with other devices using Bluetooth.

Figura 51.- Características Micro:Bit

UI-1220LE-C-HQ

UI-1220LE-C-HQ Rev.2 (AB02420)

Discontinued
The model has been discontinued.



uEye industrial cameras now also work with IDS peak! We recommend the Software Development Kit for the implementation of new projects. Learn about the process here and switch now.
Please note: The technical data given here was measured using the IDS Software Suite.

Specification

Sensor

Sensor type	CMOS Color
Shutter	Global Shutter
Sensor characteristic	Linear with knee points
Readout mode	Progressive scan
Pixel Class	0.4 MP
Resolution	0.36 Mpix
Resolution (h x v)	752 x 480 Pixel
Aspect ratio	14:9
ADC	10 bit
Color depth (camera)	8 bit
Optical sensor class	1/3""
Optical Size	4.512 mm x 2.880 mm
Optical sensor diagonal	5.35 mm (1/2.99")
Pixel size	6 µm
Manufacturer	Onsemi
Sensor Model	MT9V032C12STC
Gain (master/RGB)	4x/5x
AOI horizontal	increased frame rate
AOI vertical	increased frame rate
AOI image width / step width	16 / 4
AOI image height / step width	4 / 2
AOI position grid (horizontal/vertical)	4 / 2
Binning horizontal	increased frame rate
Binning vertical	increased frame rate
Binning method	-
Binning factor	-
Subsampling horizontal	-
Subsampling vertical	-
Subsampling method	-
Subsampling factor	-

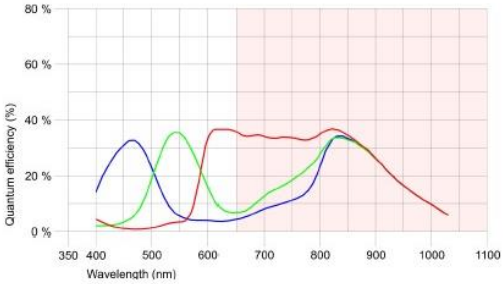


Figura 52.- Características UI-1220LE-C-HQ

AZURE – 0420MM

Item No		AZURE-0420MM
Resolution		100lp/mm(2MP)
Format		1/2"
Mount		C
Focal Length		4mm
F/No		F2.0~F16
Back Focus		8.77mm
Iris Type		M-Iris
Angle of view (DXVXH)	1/2"Format	90°×61.8°×77.4°
	1/3"Format	73.74°×48.46°×61.93°
	1/4"Format	53.13°×33.4°×43.6°
Distortion		< -2.8%
Working Distance		0.1m~∞
Design Wavelength		400~700nm
Filter Thread		-
Dimension		¢44.5×35.97mm
Weight		130g

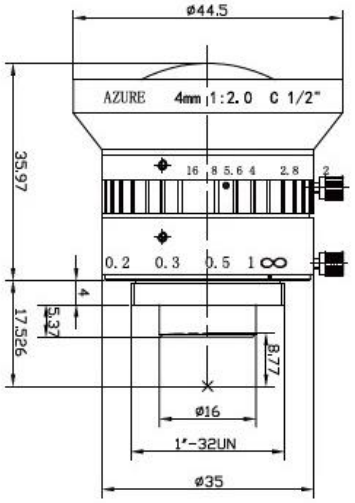


Figura 53.- Características AZURE – 0420MM