DESARROLLO DE UNA APLICACIÓN EN MATLAB PARA EL CONTROL DE TRAYECTORIAS DE ROBOTS LEGO MINDSTORMS IDENTIFICANDO EL RECORRIDO MÁS CORTO ENTRE DOS PUNTOS EN AMBIENTES CON OBSTÁCULOS.

SANTIAGO FEDERICO MORALES SALINAS

CÓD: 2166030

UNIVERSIDAD SANTO TOMÁS
FACULTAD DE INGENIERÍA MECÁNICA
DIVISIÓN DE INGENIERÍAS
BOGOTÁ D.C.
2020

DESARROLLO DE UNA APLICACIÓN EN MATLAB PARA EL CONTROL DE TRAYECTORIAS DE ROBOTS LEGO MINDSTORMS IDENTIFICANDO EL RECORRIDO MÁS CORTO ENTRE DOS PUNTOS EN AMBIENTES CON OBSTÁCULOS.

SANTIAGO FEDERICO MORALES SALINAS

CÓD: 2166030

PROYECTO DE TRABAJO DE GRADO EN LA MODALIDAD DE SOLUCIÓN A UN PROBLEMA DE INGENIERÍA PARA OPTAR AL TÍTULO DE INGENIERO MECÁNICO

DIRECTOR
JESÚS DAVID VILLARREAL LÓPEZ

CODIRECTOR
JOSÉ LIBARDO ROJAS FERNANDEZ

UNIVERSIDAD SANTO TOMÁS
FACULTAD DE INGENIERÍA MECÁNICA
DIVISIÓN DE INGENIERÍAS
BOGOTÁ D.C.
2020

TABLA DE CONTENIDO

LISTADO DE FIGURAS

LISTADO DE TABLAS

1. INTRODUCCIÓN

Las nuevas tendencias tecnológicas están realizando cambios en el mundo, entre más pasa el tiempo cada vez se están viendo más robots y aplicaciones en el ámbito cotidiano de la vida, y esto indica que las tareas de a poco están siendo realizadas ya por los robots sin necesidad que intervenga el ser humano.

Es aquí donde entra la robótica la cual estudia el diseño y construcción de los robots capaces de realizar los trabajos que proponen las personas, la robótica puede verse como una combinación de ingenierías en las que están la mecánica, electrónica y sistemas, cada una de ellas se ve reflejada en una máquina, ya sea por sus mecanismos y diseño, sus algoritmos y programas que lo controlan, sus circuitos y sensores que ayudan a todos los movimientos que este desarrolle.

La robótica tiene tantas clasificaciones como la móvil, hibrida, androide entre otras, y nunca va a parar de avanzar con el tiempo, esto implica que las personas deberán adquirir competencias relacionadas con el diseño, la fabricación y el mantenimiento de sistemas automáticos programables y aplicaciones en la misma.

Este trabajo se va a enfocar en la robótica móvil, la cual se ve involucrada en muchos aspectos de la vida cotidiana y en muchos entornos como en la milicia, hospitales, industriales, seguridad, en el hogar, entre otros, y esto gracias a que pueden ser usados en los entornos difíciles o inaccesibles para los seres humanos.

Como se viene diciendo a medida que avanza el tiempo la robótica móvil y los dispositivos han avanzado de tal manera que son mejores que antes, y lo que se quisiera llegar a obtener es un robot móvil lo suficientemente autónomo, esto quiere decir que tenga la menor supervisión del ser humano, para que los trabajos o tareas se hagan de manera más efectiva.

Los problemas que se pueden observar en este tipo de robots, son sus sistemas de desplazamiento, lo que son las ruedas, orugas, entre otro tipo de sistemas, ya que se tiene que ver qué tipo superficie en donde se va a desarrollar el desplazamiento, y esto también influye en la planificación de trayectorias, que dependiendo ya el tipo de

entorno si es estructurado o no y si se trata de un lugar a la intemperie o no, cambia la programación y condiciones de planificación.

El presente trabajo tiene por finalidad implementar y desarrollar una aplicación para el control de trayectorias de robots Lego Mindstorms identificando el recorrido más corto entre dos puntos mediante el uso del software MATLAB.

2. OBJETIVO GENERAL

Desarrollo de una aplicación en Matlab para el control de trayectorias de robots Lego Mindstorms identificando el recorrido más corto entre dos puntos en ambientes con obstáculos.

2.1. OBJETIVOS ESPECÍFICOS

- Definir la arquitectura del robot móvil teniendo en cuenta las características del entorno de desplazamiento establecido.
- Desarrollar un sistema de obtención y tratamiento de imágenes para la generación de la trayectoria más corta entre dos puntos evitando obstáculos.
- Desarrollar un sistema de control para los actuadores del robot.
- Realizar pruebas para verificar el funcionamiento de la aplicación.

3. MARCO REFERENCIAL

3.1. MARCO CONCEPTUAL

A continuación, se explicarán algunos conceptos importantes.

3.1.1. PLATAFORMA

Una plataforma es un sistema que sirve para hacer funcionar determinados módulos en un software o hardware, pero siempre y cuando estos sean compatibles con la plataforma. Para poder definir una plataforma se deben integrar los sistemas operativos, lenguajes de programación, entre otros [1].

3.1.2. LENGUAJE DE PROGRAMACIÓN

Los lenguajes de programación son los que sirven para crear programas que permitan la comunicación entre hombre/máquina y dar soluciones a problemas utilizando como medio el procesador y almacenamiento de computadoras [2].

3.1.3. SOFTWARE

El software, es el conjunto de programas ejecutables por el computador [3], esto hace referencia a todo lo que no es material o lo que no se puede tocar pero que hace funcionar el computador, ejemplos de estos son: Navegadores de internet, Antivirus, Videojuegos, entre otros [4].

3.1.4. ROBÓTICA

Es una de las ramas científicas que se centra en desarrollar una clase de sistemas mecánicos, llamados “robots”, los cuales se usan en una amplia variedad de aplicaciones ya sean científicas, industriales, domesticas, entre otras [5].

3.2. MARCO TEÓRICO

3.2.1. PLANIFICACIÓN DE TRAYECTORIAS

La planificación de trayectorias es la que se encarga de buscar y de encontrar una ruta determinada que va desde la posición inicial del Robot Móvil hasta la posición final o meta, esta ruta tiene que estar libre de cualquier colisión entre el robot y alguno de los obstáculos, la planificación de trayectorias se clasifica en: determinísticos y aleatorios,

en este trabajo se usara trayectorias de tipo determinístico y está clasificado en: campos potenciales artificiales y en basados en grafos.

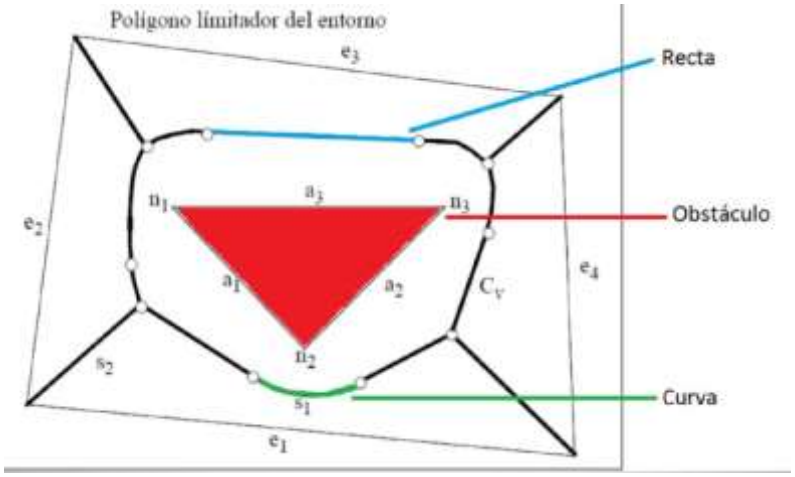
- **PLANIFICACIÓN BASADA EN GRAFOS**

Esta planificación consta de dos fases, las cuales consisten en primero construir distintas rutas o grafos, que conectan el inicio de la ruta con el final, y la segunda fase consiste ejecutar el algoritmo que se usará y este seleccionaría la trayectoria más eficiente, es decir, la trayectoria más corta en la cual no existe ninguna colisión con los obstáculos.

- **DIAGRAMAS DE VORONOI**

Este método de planificación genera rutas que están a la distancia más óptima posible de los obstáculos que se encuentran en el espacio de trabajo. El método genera líneas entre los nodos ya sean rectas o curvas, según la forma que tiene el obstáculo, así como se ve en la Figura 1, en donde el obstáculo tiene forma de triángulo y se observa que en las caras del polígono el diagrama genera líneas rectas, mientras que en las puntas el diagrama genera curvas, siempre intentando estar lo más lejos posible del obstáculo para no tener interferencias [6]**.**

Figura 1. Ejemplo de Diagrama de Voronoi.



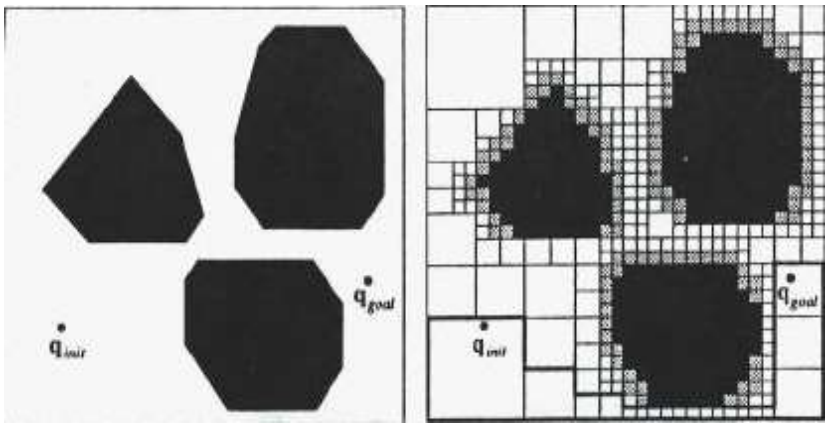
Fuente: [6].

- **DESCOMPOSICIÓN EN CELDAS**

Este método de planificación genera una cuadricula o una serie de celdas sobre el espacio trabajado, la cuadricula puede ser exacta o aproximada, esta última se genera desarrollando un enmallado del espacio, refinando la malla en las zonas donde se encuentran los obstáculos, de manera que se obtiene una malla con una discretización más detallada en las zonas donde se requiere [1].

En la Figura 2 se expone un ejemplo de un espacio con sus obstáculos, especificando los puntos de inicio y fin de la trayectoria. En la imagen derecha se muestra un ejemplo del refinamiento del enmallado del espacio de manera aproximada, detallando las zonas donde están los obstáculos, para evitar que no se genere una colisión y dando un rango más amplio en las zonas más alejadas de estos.

Figura 2. Ejemplo de Descomposición de celdas.



Fuente: [2].

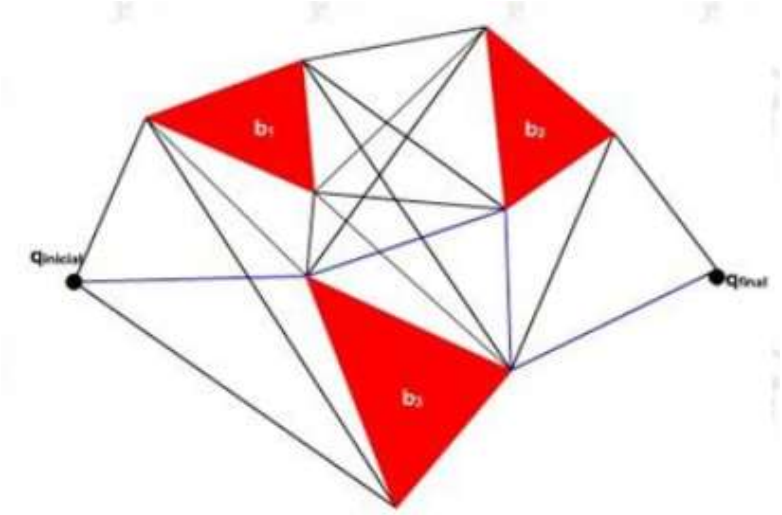
- **GRAFOS DE VISIBILIDAD**

El método de grafos de visibilidad toma las formas de los obstáculos y las representa como polígonos regulares, teniendo los obstáculos ubicados y caracterizados a continuación procede a generar diferentes trayectorias, todas estas siempre pasando por los vértices de los polígonos, el algoritmo toma la ubicación inicial y final de la ruta como nodos, luego el método selecciona la mejor trayectoria por donde se realizará el movimiento.

En la Figura 3 se observa el funcionamiento del algoritmo de grafos de visibilidad, en el cual se puede ver como los obstáculos están siendo representados por triángulos rojos,

y se muestra la cantidad de segmentos posibles unidos en todos los vértices de los polígonos, el paso siguiente del algoritmo es generar todas las trayectorias posibles, y por ultimo seleccionar la mejor, el caso de ejemplo selecciona la trayectoria resaltada de azul como la ruta resultante que será usada por el dispositivo [1].

Figura 3. Ejemplo de Grafo de Visibilidad.



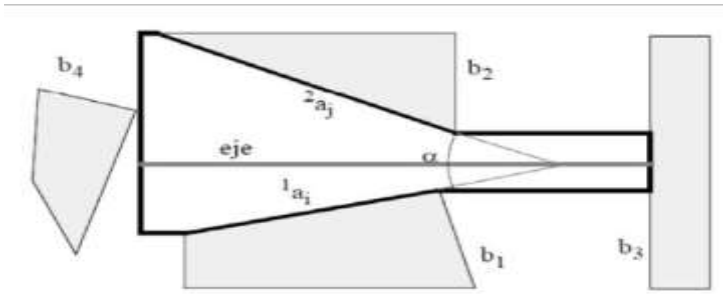
Fuente: [7].

- **MODELADO DEL ESPACIO LIBRE**

El método de modelado del espacio libre se aplica a entornos de desplazamiento en los que existan obstáculos con figuras de polígonos. La realización de la malla se da mediante formas de cilindros rectilíneos generalizados o CRG. De manera que cuando se utilizan los cilindros como método de planificación, se busca que el dispositivo se traslade de la manera más segura, esto quiere decir que siempre estará lo más alejado posible que se pueda de los obstáculos.

En la Figura 4 se puede ver la manera en cómo se usan los CRG, desde la configuración inicial hasta la final, estos CRG están interconectados, se generan por medio de las caras de los obstáculos, y se puede ver la ruta trazada que pasa en mitad de todos los CRG cumpliendo su función, manteniendo al dispositivo lo más alejado posible de los obstáculos [1].

Figura 4. Ejemplo de modelado de Espacio Libre.

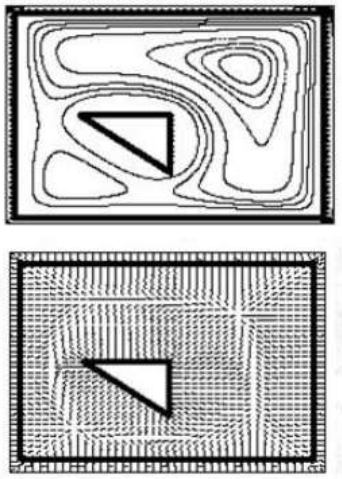


Fuente: [7].

- **PLANIFICACIÓN EMPLEANDO CAMPOS POTENCIALES**

Este método de planificación se debe plantear de manera que cada objeto en el entorno de trabajo tendrá una “carga eléctrica” igual o distinta dependiendo cual sea. Por ejemplo el robot y los obstáculos deben tener la misma carga, esto para que se genere repulsión entre ellos y no se puedan chocar o cruzar entre ellos, mientras que el objetivo o meta tendrá carga opuesta para atraer el robot, en la Figura 5 se puede observar un par de imágenes, en estas se pueden identificar un campo magnético y en la otra un campo de curvas de nivel, en el campo eléctrico se percibe cómo se repelen las cargas ubicadas en el entorno. Para generar los campos potenciales solo se requiere el cálculo de gradientes, y los campos generados pueden ser convexos o no convexos, se debe tener cuidado cual tipo de campo se genera ya que podría ser no convexo y esto haría que el robot cayera en un hueco o trampa y quedar atrapado [2].

Figura 5. Ejemplo de Planificación usando Campos Potenciales.



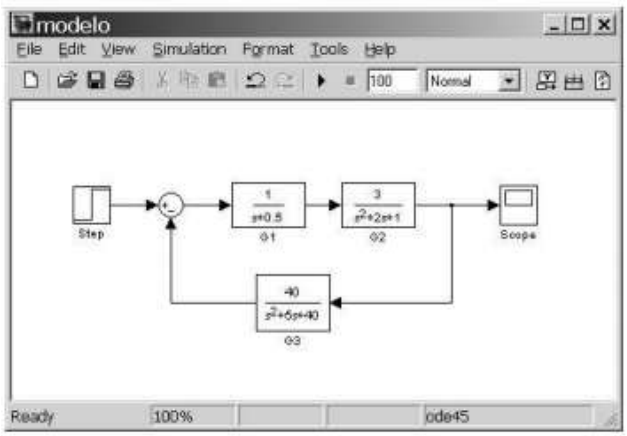
Fuente: [7].

Para el desarrollo de plataformas computacionales se han usado algunos de los siguientes complementos o software.

3.2.2. SIMULINK

Es un complemento o Toolbox de Matlab, en el que se simulan y construyen modelos por medio de diagramas de bloques, en la Figura 6 se aprecia un modelo, cada modelo contiene diagramas de bloques, cada uno con un conjunto de características como los nombres y funciones específicas que pueden realizar. Es una herramienta amplia que se complementa con numerosos elementos opcionales que se pueden descargar como complementos, la capacidad de almacenamiento de datos y simulaciones es bastante cómoda con lo que necesite resolver [3].

Figura 6. Modelo en Simulink.



Fuente: [3].

3.2.3. RWTH

El RWTH es un complemento o Toolbox de Matlab, el cual está desarrollado para controlar ladrillos inteligentes NXT, estos son los cerebros o centro de mando de los robots Lego Mindstorms NXT, las conexiones se pueden realizar de manera inalámbrica o alámbrica, las funciones que vienen incluidas en el Toolbox principalmente son para controlar el NXT Brick o bloque cerebro del robot, tras recibir los datos el NXT Brick da las órdenes a los actuadores. En la Figura 7 se aprecia un algoritmo que comanda los sensores del robot, este Toolbox brinda un importante conjunto de posibilidades para darle inteligencia artificial y características a los Robots de manera rápida y sencilla [4].

Figura 7. Algoritmo que lee el valor actual del sensor de luz usando comandos directos de NXT.

```
% initialize the sensor
NXT_SetInputMode(SENSOR_1, 'LIGHT_ACTIVE', 'RAWMODE', 'dontreply');

data = NXT_GetInputValues(SENSOR_1); % get light sensor value

if ~data.Valid              % check valid flag, re-request data if necessary
    startTime = clock();    % initialize timeout counter
    timeout   = 0.3;        % set time out to 300 ms

    while (~data.Valid) && (etime(clock, startTime) < timeout)
        data = NXT_GetInputValues(SENSOR_1); % re-request until valid/timeout
    end
end

light = double(data.NormalizedADVal); % use normalized light value (10 bit)
```

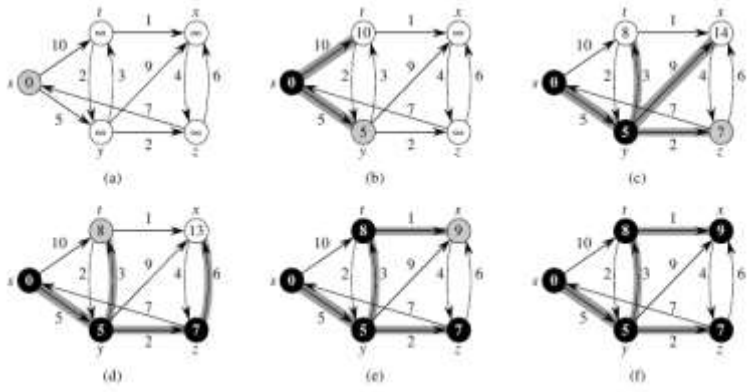
Fuente: [4].

3.2.4. ALGORITMOS DE BÚSQUEDA

- **DIJKSTRA**

El algoritmo Dijkstra es uno de los algoritmos más usados en la búsqueda de caminos mínimos, el funcionamiento del algoritmo es básicamente un número de iteraciones, que a medida que la red o cantidad de nodos va creciendo el cálculo de cada uno incrementa su dificultad, ya que son más iteraciones y en cuestiones prácticas esto es una gran desventaja en comparación con otros algoritmos de búsqueda.

Figura 8. Demostración Algoritmo Dijkstra.



Fuente: [8].

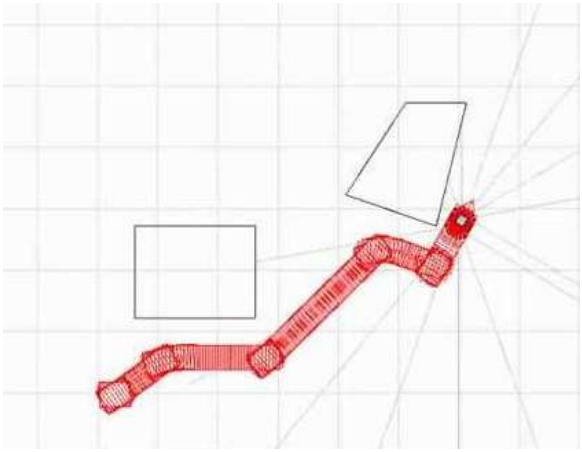
El algoritmo Dijkstra resuelve rutas ya sean cíclicas o a cíclicas, esto significa que el paso por un nodo de la red puede repetirse si así se determina la trayectoria, por ejemplo en la Figura 8 se puede observar un ejemplo de funcionamiento del algoritmo

con una red que es tipo cíclica, que en este caso lo que se hace es identificar o etiquetar cada uno de los nodos, estas etiquetas son permanentes o temporales, las etiquetas funcionan de una manera la cual va pasando por cada nodo y al identificar que el nodo es necesario para la trayectoria lo convierte en etiqueta permanente, y así continua hasta el nodo final, la trayectoria son todos los nodos que tengan etiqueta permanente, las etiquetas tienen la información relacionada con un valor acumulado de la trayectoria y con el nodo más próximo que sigue en la ruta [9].

- **DISTBUG**

El algoritmo Distbug se emplea para la generación de trayectorias para la navegación de robots móviles, en la Figura 9 se puede observar un ejemplo de funcionamiento de este algoritmo, siempre se parte de una posición inicial con unas coordenadas específicas, y se usa un láser que hace un barrido de 180° para identificar si existe algún tipo de obstáculo, mientras que no exista obstáculo el algoritmo continua avanzando, cuando se encuentra con un obstáculo vuelve a realizar el barrido para buscar la nueva ubicación con el fin de poder continuar con la trayectoria, y esto se realiza hasta llegar a la posición final o meta esta con unas coordenadas finales [10].

Figura 9. Demostración Algoritmo Distbug.



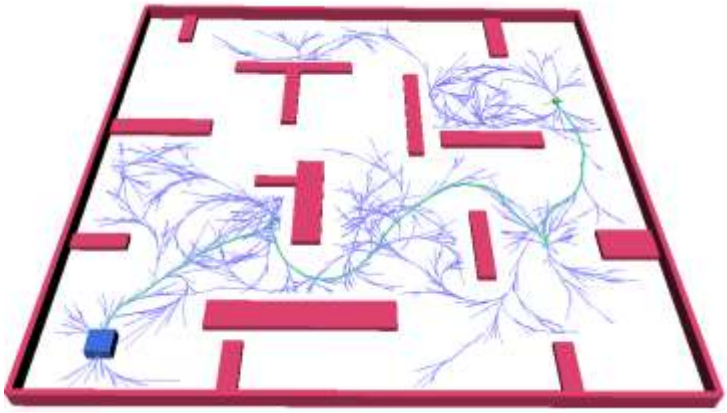
Fuente: [11].

- **RAPIDLY RAMDOM TREES**

El algoritmo RRT es usado principalmente para buscar trayectorias, de manera más rápida y eficientemente para espacios de trabajo con tamaños grandes, en el que se crea un árbol o un tipo de ramificación por todo el espacio a trabajar, cada vez que se

encuentra con un obstáculo termina la ramificación y crea una nueva por un lugar distinto donde este libre su camino, este tipo de algoritmos son usados en la búsqueda de trayectorias de robots móviles [10].

Figura 10. Demostración Algoritmo RRT.



Fuente: [12].

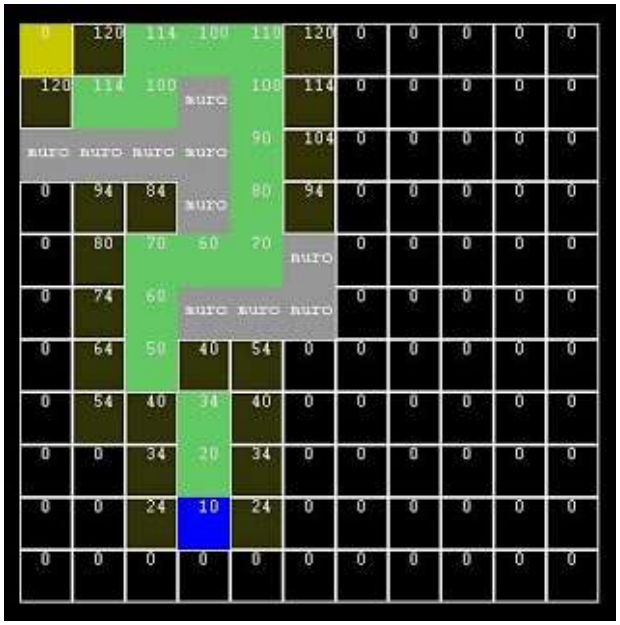
En la Figura 10 se observa la demostración del algoritmo, en donde la figura azul representa el robot y el punto verde representa la meta o ubicación final, como se observa la trayectoria se genera luego de que las ramificaciones se conectan, en este algoritmo cada uno de los elementos del entorno inician sus caminos o ramificaciones individuales por todo el espacio, cuando se conectan en alguna zona del espacio ya se detiene el algoritmo, y así genera la trayectoria, como se puede apreciar en el ejemplo es la que esta subrayada con color verde en esta figura.

- **ALGORITMO A***

El algoritmo A* es otro de los algoritmos de búsqueda usados para la planeación de trayectorias para robots, es un algoritmo que se emplea para calcular las rutas más rápidas en un espacio, se trata de una función heurística la cual etiqueta cada nodo de dos distintas maneras, una que el nodo es abierto cuando fue revisado pero aún faltan nodos cercanos por visitar y verificar cuál de todos es menos costoso, luego de revisar todos los nodos cercanos y saber cuál es el nodo que menos cuesta, este nodo pasa a ser cerrado y así continua hasta llegar al nodo final, y la trayectoria del robot va a ser el conjunto de nodos cerrados, como se ve en la Figura 11 en donde la trayectoria menos costosa es la de color verde y se puede ver el costo de los nodos cercanos que es más

alto que los escogidos, este es de los algoritmos más completos ya que tiene en cuenta el costo real del recorrido completo y el valor heurístico entre nodos intermedios.

Figura 11. Demostración de Algoritmo A*.



Fuente: [13].

3.3. ESTADO DEL ARTE

A continuación, se mostrarán algunos de los proyectos desarrollados sobre generación de trayectorias y comunicación con robots móviles.

3.3.1. PLANIFICACIÓN DE TRAYECTORIAS PARA UN ROBOT MÓVIL

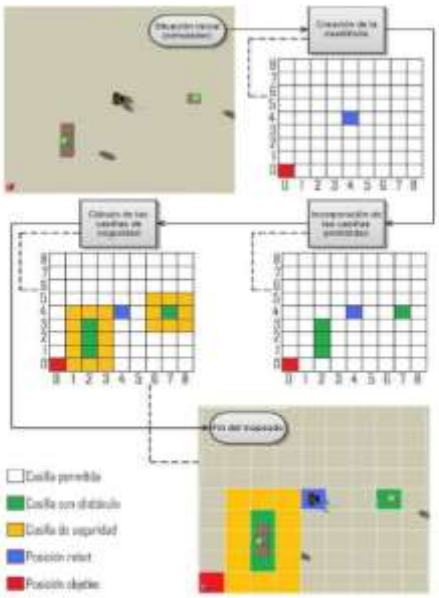
El objetivo del proyecto fue diseñar la estrategia de planificación para un robot móvil, el cual debe atravesar el campo sin chocar contra ningún obstáculo, el proyecto tenía un robot móvil como el que se observa en la Figura 12, este dispositivo estaba capacitado con un módulo de visión estereoscópica, la conexión entre el robot y el ordenador se realizaba a través de una red Wi-Fi [1].

Figura 12. Robot Surveyor SRV-1.

Fuente: [1].

El módulo de trayectorias se encarga de buscar una ruta, por la que el robot móvil pueda desplazarse entre una posición de inicio y una final, para que esto ocurra el algoritmo utiliza la función A* la cual realiza un mapeo del entorno y genera la ruta, en la Figura 13 se puede observar el funcionamiento del mapeo, igualmente la ubicación de cada uno de los obstáculos, robot, y meta del programa [1].

Figura 13. Mapeo e ubicación del robot en el entorno.



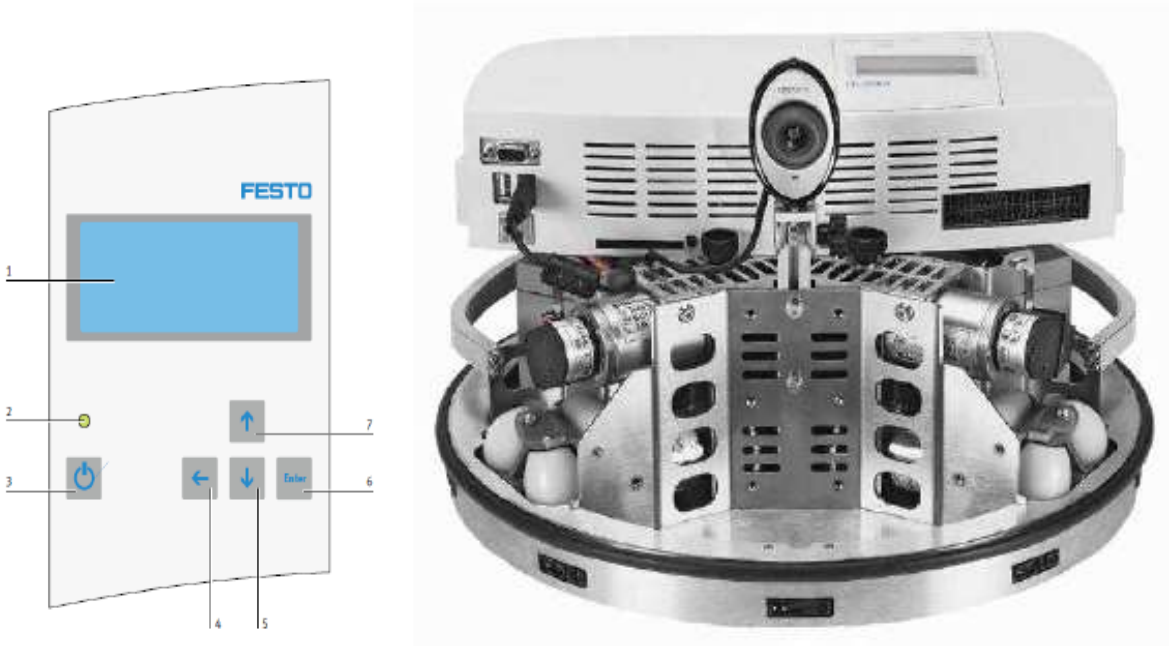
Fuente: [1].

Para concluir el proyecto genero el modulo o programa capaz de generar la trayectoria deseada, además de que el proyecto también tiene un módulo de posicionamiento tridimensional lo que hace referencia a que la ubicación del robot siempre se sabe, esto para poder ir realizando el barrido desde su cámara o visión estereoscópica y así trazar su trayectoria.

3.3.2. PLANEACIÓN Y SEGUIMIENTO DE TRAYECTORIAS PARA UN ROBOT MÓVIL

En el proyecto realizado se estudian los distintos métodos para poder realizar una planificación de trayectoria, para luego escoger uno de ellos, de esta manera ser desarrollado e implementado en una plataforma educativa llamada Robotino®, en la Figura 14 se puede observar el robot móvil, la conexión realizada entre el ordenador y el robot fue mediante una red Wi-Fi, este robot viene dotado con un ladrillo inteligente similar al de los Lego NXT, este es el que recibe y envía los comandos necesarios para el movimiento del robot.

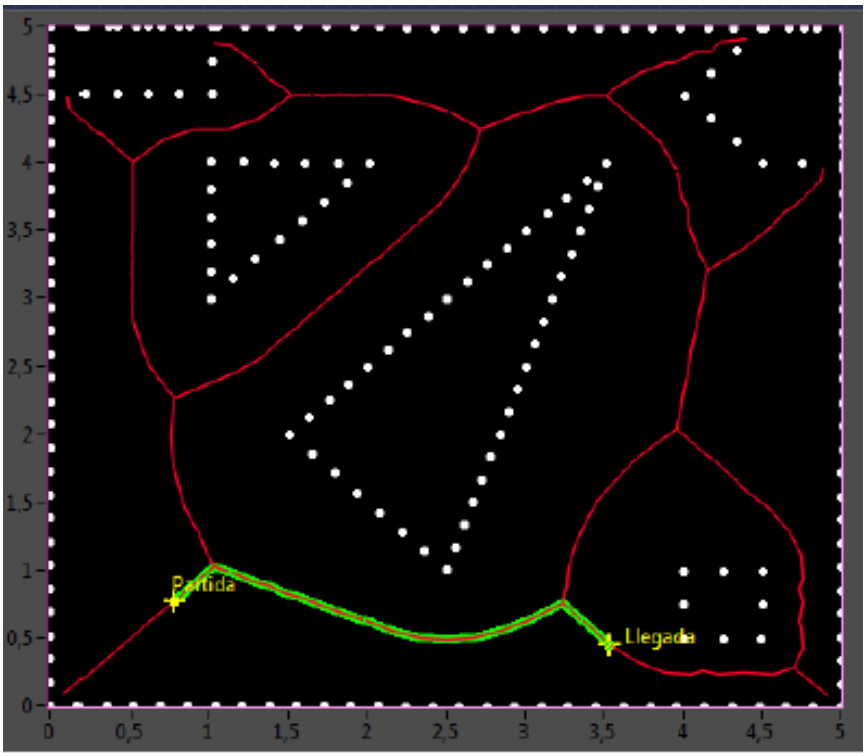
Figura 14. Plataforma educativa Robotino®.



Fuente: [6].

En el proyecto se explican los diferentes métodos de planificación de trayectorias que existen, escogen el método de diagramas de Voronoi para realizar el cálculo y planificación de la ruta, debido a la seguridad que entrega al robot para no encontrarse con ningún obstáculo, ya que es el que más alejado mantienen al robot del obstáculo, en la Figura 15 se puede observar un caso planteado donde las figuras de color blanco son los obstáculos, las líneas rojas son los diagramas generados por el método, que podrían usarse como rutas para ir del inicio al final, por último la línea verde que es la ruta generada gracias al algoritmo A* el cual escoge la ruta de menor valor entre todas, y se puede observar la manera en como el diagrama genera un espacio entre los obstáculos y la trayectoria.

Figura 15. Trayectoria trazada con método de Diagrama de Voronoi.



Fuente: [6].

Existen varios tipos de planeación y generación de trayectorias, el usuario es el que deberá decidir cuál de todos escoger esto quiere decir que tendrá que seleccionar el que más se adapte a las condiciones que él tenga.

4. DEFINICIÓN DE LA ARQUITECTURA Y CARACTERÍSTICAS TOPOGRÁFICAS DEL ENTORNO PARA EL DESPLAZAMIENTO DEL ROBOT MÓVIL.

4.1. NAVEGACIÓN DE LOS ROBOTS MÓVILES

La navegación son el conjunto de técnicas para dirigir la trayectoria del robot móvil mientras atraviesa el entorno en el que desarrolla su movimiento, esto lo hace sin olvidar su meta y sin chocar contra obstáculos que están en el camino [14].

Lo más importante para poder realizar una trayectoria es tener un mapa ya que es el que representa el entorno en donde se moverá el robot y lo hace más fácil ya que el robot o programa ejecutara la ruta deseada sabiendo los puntos de inicio y meta.

4.1.1. TIPOS DE MAPAS

Los tipos de mapas pueden dividirse entre los que son basados en información sensorial, y de acuerdo con lo que almacenan, algunos de estos tipos de mapas en donde pueden movilizarse los robots móviles son:

- **MAPAS DE MARCAS EN EL TERRENO – INFORMACIÓN SENSORIAL**

Las marcas en el terreno son zonas que el robot y sus sistemas sensoriales podrían identificar de una manera más cómoda, en los mapas estas marcas se pueden identificar como nodos, estos nodos se unen para representar la accesibilidad en ese punto. Esas líneas con las que se asocian los nodos logran ser características como distancias, direcciones las cuales suelen ser usadas para la generación de trayectorias [14].

- **MAPAS DE OCUPACIÓN – INFORMACIÓN SENSORIAL**

Los mapas de ocupación representan el entorno como una red, cada casilla tiene un valor el cual es útil tanto para el robot como para el algoritmo, de esta manera el robot tiene la posibilidad de saber si la casilla está o no ocupada, los valores que suelen ser usados en estos mapas son -1 el cual indica que la zona se encuentra libre, en el caso contrario usar +1 esto simboliza que la casilla se encuentra ocupada. Los mapas se construyen gracias a distintos tipos de asistencias visuales tales como imágenes tomadas con cámaras [14].

- **MAPAS DE OBJETOS – RESPECTO A LO QUE ALMACENAN**

Este tipo de mapa almacena la cantidad de obstáculos que se encuentran en el entorno, con los cuales el robot podría tropezar a medida que ejecuta la trayectoria deseada, el almacenamiento se genera identificando los obstáculos, y estos relacionarlos con un polígono para así poder guardar la información del centroide de la figura, con la extensión máxima en una serie de direcciones desde el punto central [14].

- **MAPAS COMPUESTOS - RESPECTO A LO QUE ALMACENAN**

Los mapas compuestos tienen la facilidad de almacenar tanto información de objetos como de espacios libres que se encuentren en el entorno, la manera como el algoritmo realiza la recolección de datos es generando fraccionamientos en el terreno, y a cada parte le da unas características específicas. El resultado es identificar si la región se encuentra ocupada, vacía o parcialmente ocupada [14].

- **MAPAS DE ESPACIO LIBRE - RESPECTO A LO QUE ALMACENAN**

Los mapas de espacio libre son semejantes a los mapas de marcas, los dos usan el mismo método de almacenamiento, la diferencia se encuentra en que en este tipo de mapa cada nodo representa un punto de parada, es allí en donde el robot explora el medio con sus diferentes sensores. Los nodos se unen por medio de líneas rectas, quienes se convertirán en la trayectoria del robot [14].

Para este proyecto se utilizó el mapa de ocupación, ya que el proyecto cuenta con dispositivos quienes son los encargados de la obtención de imágenes del terreno, de manera que no será necesario que el robot cuente con sistemas de sensores o infrarrojos, simplemente el robot necesitará de sus motores para el correcto desplazamiento a través del entorno.

4.2. TERRENOS EN DONDE SE MUEVEN LOS ROBOTS MÓVILES

Los terrenos en los cuales pueden operar los robots móviles son diversos, estos se clasifican en dos tipos, el primero que se caracteriza por los objetos que están presentes en el terreno en donde el robot hace su operación, y el segundo tipo de terreno se identifica según el área en donde realizan su operación.

En la primera clasificación se habla de los objetos que están presentes en el terreno de operación del robot, aquí se subdivide de manera que el entorno puede ser estructurado o no estructurado. Se habla de un tipo de terreno estructurado cuando los objetos que se encuentran en el entorno no cambian de forma ni posición, en ocasiones los objetos poseen características que podrían asemejarlos a figuras geométricas como cilindros, prismas [15].

Por el contrario, cuando el terreno es de tipo no estructurado los objetos que están en el medio son dinámicos, esto quiere decir que cambian sus características físicas, y los cambios que ocurren de maneras que no se pueden predecir. Un ejemplo de un entorno estructurado es una bodega de un almacén, en donde los estantes de la mercancía u objetos siempre se encuentran en el mismo lugar así como se puede ver en la Figura 17, además que se podrían asociar con figuras geométricas para realizar sus respectivos mapeos y trayectorias [15].

Un ejemplo para un entorno no estructurado es una fábrica en su línea de producción como se puede ver en la Figura 16, en la cual si existe movimiento por parte de otros robots, máquinas, personas entre otros, esto quiere decir que el entorno va cambiando con el tiempo, esto genera restricciones para el robot y para la generación de trayectorias ya que se deben tener en cuenta más factores adicionales.

Figura 16. Entorno no estructurado, línea de producción.

Fuente: [16].

Figura 17. Entorno estructurado, bodega de almacén.

Fuente: [17].

En la segunda clasificación de terrenos se habla según el área de trabajo en donde opere el robot, esta se clasifica en área interior o área exterior, se habla de área de trabajo interior cuando el terreno o entorno donde opera el robot se encuentra encerrado, esto quiere decir en los entornos en donde hay paredes y cubiertas, además de luz artificial o eléctrica, ejemplos de estas áreas son laboratorios, casas, bodegas de producción, así como se puede ver en la Figura 18, en la cual se observa un robot limpiador de una casa [15].

Cuando se habla de área de trabajo exterior se entiende que el entorno no se encuentra delimitado, la iluminación de estos lugares es natural, ejemplos de áreas de este tipo son parques, bosques, así como se puede apreciar en la Figura 19, en donde se puede observar un robot móvil el cual es utilizado en operaciones militares y que generalmente opera en espacios abiertos y de difíciles condiciones [15].

Figura 18. Área de trabajo interior, robot limpiador de casa.

Fuente: [18].

Figura 19. Área de trabajo exterior, Robot usado en operaciones militares.

Fuente: [19].

El terreno en el que el robot realizará su operación en este proyecto tendrá 2 características especiales, una de ellas es que el terreno será de tipo estructurado, de modo que el terreno contara con obstáculos que no cambiaran de posición mientras el robot realice su desplazamiento, y la siguiente característica se trata del área de trabajo la cual será en un espacio interior.

4.3. TIPOS DE SISTEMAS DE LOCOMOCIÓN

El sistema de locomoción es una de las partes más importantes que existen en un robot, ya que estos son los que permiten el movimiento del robot, el sistema que se utilice en el robot depende del entorno en donde realice su función, estos entornos están divididos en: terrestre, acuático, aéreo, subacuático.

El proyecto manejara el tipo de entorno terrestre, por lo que se va a profundizar en los tipos de sistemas se pueden encontrar en este tipo de entornos, con esto garantizar que el robot pueda desplazarse de manera correcta y segura a través del terreno.

4.3.1. TIPOS DE SISTEMAS EN ENTORNOS TERRESTRES

De acuerdo al terreno en donde se va a desarrollar el movimiento del robot, los sistemas pueden ser de ruedas, patas, cadenas, orugas. Cada cual tiene una característica que

la hace funcionar mejor en cada tipo de terreno como pueden ser, escalones, lodos, pisos, entre otros.

En el proyecto se va a manejar un terreno el cual es totalmente plano, no tiene ningún tipo de escalón o desnivel muy grande, por lo cual se ha seleccionado que el sistema que se va a implementar es el de las ruedas y se profundizara sobre este tipo de sistema para continuar la selección de la estructura correcta del robot.

- **SISTEMA DE RUEDAS**

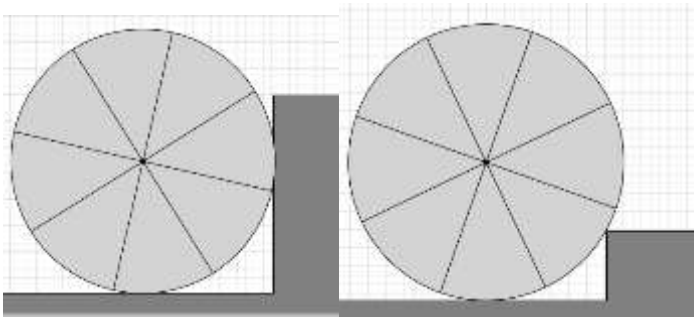
Los sistemas de ruedas son los que más se implementan en los robots, esto debido a que la fabricación de este tipo de sistemas es mucho más sencillo que los demás, también el valor de este sistema en comparación con los otros, es más bajo de lo que pueden llegar a constar los sistemas de cadenas, orugas, patas [20].

Según las características del terreno en el que los robots se van a desplazar sus sistemas de tracción y dirección cambian, los sistemas se dispensan en los ejes de las ruedas, para seleccionar la configuración adecuada se debe apreciar el número de ruedas que tendrá el robot, igualmente la manera de cómo van a estar distribuidas, y de esta manera poder superar la trayectoria de manera correcta.

El sistema de ruedas tiene ciertas desventajas, una de ellas es que los obstáculos deben ser de menor tamaño que el radio de la rueda, en la Figura 20 se pueden apreciar los dos casos que pueden ocurrir, en uno de los casos la rueda tiene un radio especifico, y el escalón tiene un mayor tamaño esto causaría un choque y el robot no podría superar el obstáculo, mientras que en el segundo caso donde la rueda es la misma, se cambió el tamaño del escalón por uno más pequeño de esta manera el robot lo supera sin ninguna dificultad.

Existen varios tipos de configuración de ruedas y cada uno tiene sus propias características, para implementarlos se debe apreciar el grado de maniobrabilidad, el terreno, y el uso del robot, a continuación, se van a explicar unos tipos de configuraciones como lo son: triciclo, ackerman y por último la diferencial.

Figura 20. Escalón de mayor tamaño, escalón de menor tamaño.

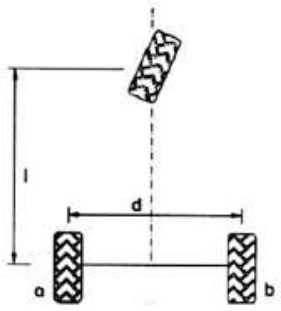


Fuente: [21].

- **CONFIGURACIÓN TIPO TRICICLO**

La configuración de tipo triciclo consta de 3 ruedas así como se puede ver en la Figura 21, una rueda delantera caracterizada por ser la principal, esto se debe a que es allí en donde encuentra el motor y la dirección ensamblada, de manera que es la parte que genera el movimiento del robot, en la parte trasera se ubican las 2 ruedas fijas, estas ensambladas sobre un mismo eje y son las que le aportan estabilidad al robot.

Figura 21. Configuración de robot móvil tipo triciclo.



Fuente: [20].

- **CONFIGURACIÓN TIPO DIFERENCIAL**

La configuración de tipo diferencial es semejante a la “tipo triciclo”, esto se debe a que de la misma manera en su configuración posee 3 ruedas, en la Figura 22 se logra observar un ejemplo de robot diferencial, la diferencia entre los dos tipos de configuración se da en la distribución de los motores y dirección, esta configuración tiene en su parte delantera 2 ruedas perpendiculares conjuntamente cada una viene dotada de un motor, la manera como el robot cambia de dirección o de marcha se da

por el cambio de velocidades angulares y lineales de cada motor, por último la rueda trasera sirve solo de apoyo para estabilidad del robot, esta rueda suele ser libre para mejorar el movimiento en los giros del robot [20].

Figura 22. Configuración de robot móvil tipo diferencial.

Fuente: [22]

- **CONFIGURACIÓN TIPO ACKERMAN**

La configuración de tipo Ackerman es la más común en el mundo, esto se debe a que su claro ejemplo es el automóvil, esta configuración consta de 4 ruedas así como se puede ver en la Figura 23, dos en la parte delantera y dos en la parte trasera, y cada par de ruedas cumple una función las delanteras funcionan como dirección mientras que las traseras tienen la función de darle la tracción al automóvil, esta configuración es muy estable y puede lograr atravesar casi cualquier tipo de superficie u obstáculo que se encuentre en su trayectoria o camino [20].

Figura 23. Configuración robot móvil tipo Ackerman.

Fuente: [23].

La configuración de ruedas usada en este proyecto fue de tipo diferencial, de manera que el robot estará dotado de 2 motores, cada uno independiente, estarán ubicados en cada rueda delantera, la rueda trasera será de tipo rueda loca que ayudará a la estabilidad del robot, es la configuración que más se adecua al terreno en el que se trabajara.

4.3. COMPONENTES DEL ROBOT

LEGO Mindstorms tiene una variedad de componentes para poder usar tales como motores, sensores de distintos tipos como por ejemplo: de proximidad, color, tacto, también de fichas para armar distintos tipos de estructuras en las cuales van soportados los componentes, además de la parte más importante del robot, el ladrillo NXT o cerebro, con todos los componentes nombrados se pueden construir una gran cantidad de diseños de robots o maquinas, los cuales son usados como herramientas para aprendizajes en diferentes áreas del estudio como en mecánica, electrónica y programación, o también para investigaciones científicas.

A continuación, se explicarán los dispositivos que componen el robot usado para la investigación:

- **LADRILLO NXT**

El ladrillo NXT es el cerebro de todos los componentes LEGO usados para la construcción del robot, en la Figura 24 se puede observar el ladrillo, este es el encargado de transmitir la información recibida de los dispositivos que componen el robot hacia el ordenador, de la misma manera del ordenador hacia los dispositivos para realizar la acción requerida por el usuario, esta información se puede transmitir a través de cable de datos USB o por Bluetooth.

Figura 24. Ladrillo Lego Mindstorms NXT

Fuente: [24].

- **MOTORES ROBOT**

El motor LEGO está compuesto por un juego de engranajes reductores, estos mejoran la fuerza del motor para no dejar que realice un esfuerzo mayor al necesario, así evitar disminuir la vida útil del mismo, también tiene un encoder o tacómetro interno que permite saber el número de vueltas que el motor ha dado, esto ayuda a una mejor precisión en los movimientos necesarios por el robot, de manera que por tener esa característica, en la programación se pueden escoger la cantidad de vueltas que tenga que dar el motor, en la Figura 25 se puede observar un motor LEGO [24].

Figura 25. Motor Lego Mindstorms

Fuente: [24].

4.4. MONTAJE DEL ROBOT

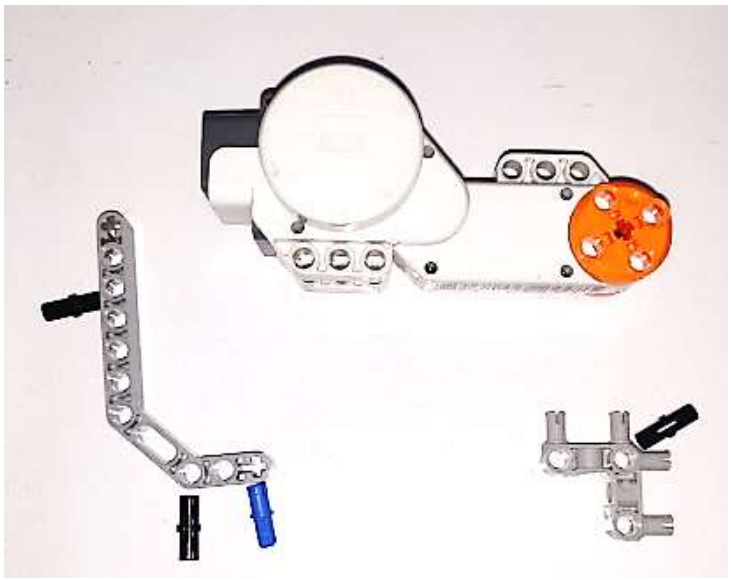
Para realizar el montaje del robot se tuvieron en cuenta los siguientes factores; el terreno y el área de trabajo los que se desplaza el robot. El tipo de terreno que se usó en esta investigación es un entorno estructurado, por lo que los obstáculos no se moverán, ni el espacio tendrá algún tipo de cambio. El área de trabajo será de tipo interior, esto quiere decir que será en un lugar cerrado, como por ejemplo una bodega de almacenamiento, en donde se tendrá iluminación artificial por lo cual, no hay que tener en cuenta espacios como escaleras o terrenos con irregularidades.

El sistema de locomoción estará integrado por ruedas, ya que esto permite un mejor desplazamiento en espacios en donde el terreno es liso sin ningún tipo de alteración. La configuración del robot será de tipo diferencial, esto debido a la facilidad de desplazamiento y también por la programación de los giros que deba hacer el robot, debido a que tiene un motor en cada rueda, esto significa que se podrá programar cada motor de manera independiente esto para mejorar la efectividad de los giros.

Una vez escogidas las características topográficas del terreno además de los sistemas de locomoción, el robot que se puede construir a partir del uso de fichas LEGO Mindstorms y que cumple con las características seleccionadas tendrá una configuración tipo triciclo, para tener esta configuración fue necesario la inclusión de 2 motores ubicados en la parte delantera uno a cada lado, en la Figura 26 se observan los elementos necesarios para poder montar los motores en la estructura, el ladrillo NXT es el cerebro encargado de recibir y enviar toda la información, este estará unido a la estructura por las fichas que se pueden ver en la Figura 31, por ultimo las ruedas del robot estarán montadas sobre el eje central del motor así como se puede observar en la Figura 33, a continuación se mostrara el registro fotográfico del ensamblaje completo del robot, este fue el resultado:

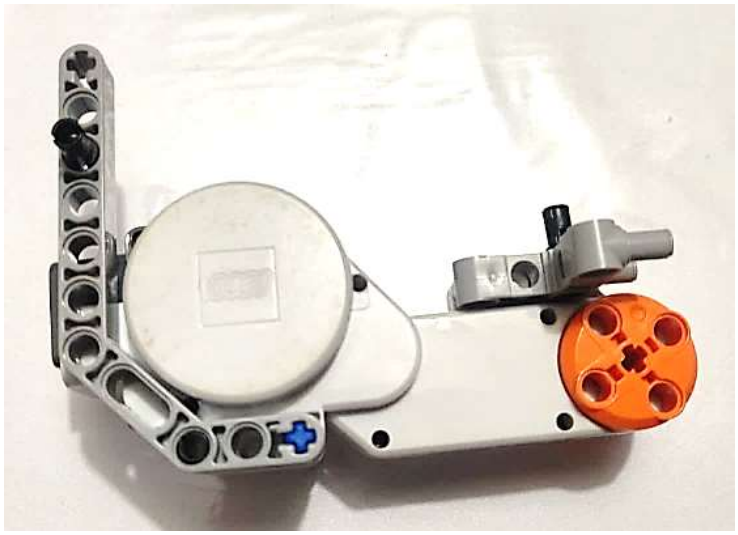
1. Montaje de los motores del robot

Figura 26. Piezas para montaje de motor.



Fuente: Autor.

Figura 27. Montaje de Motor Lego.



Fuente: Autor.

2. Montaje de motores a ladrillo NXT

Figura 28. Montaje de Motor con Ladrillo NXT.

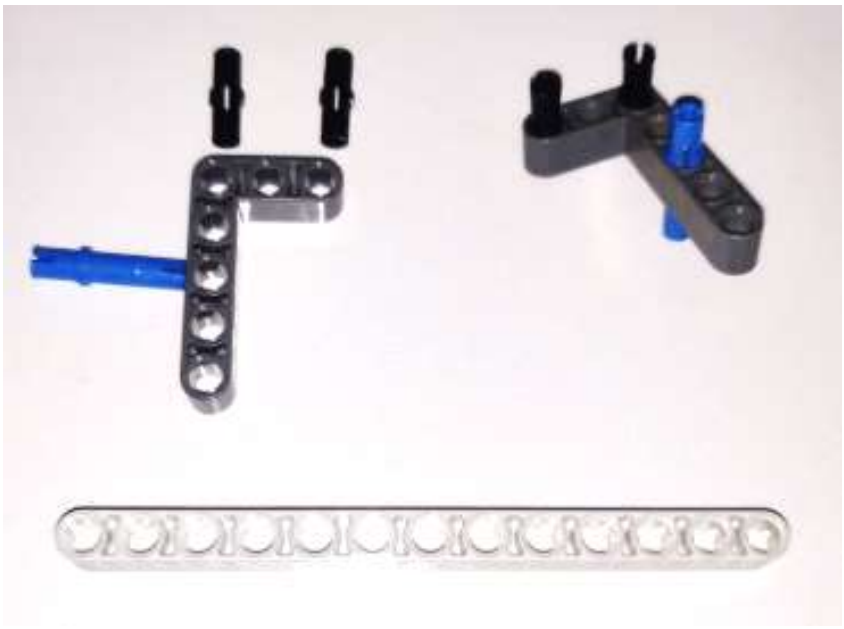
Fuente: Autor.

Figura 29. Ladrillo NXT con Motores.

Fuente: Autor.

3. Montaje de Barra para refuerzo horizontal

Figura 30. Piezas para barra transversal estructura del Robot.



Fuente: Autor.

Figura 31. Montaje de barra transversal en Robot.

Fuente: Autor.

4. Montaje de ruedas

Figura 32. Ruedas del Robot Lego.

Fuente: Autor.

Figura 33. Montaje de Ruedas en Robot Lego.

Fuente: Autor.

5. Montaje Final del Robot

Figura 34. Montaje final del Robot Lego.

Fuente: Autor.

Finalmente, el ensamble del Robot se puede apreciar en la Figura 34, en donde los motores van conectados al ladrillo NXT, por medio de cables de RJ12 especiales para el tipo conexión que hay en cada elemento del robot.

5. DESARROLLO DE UN SISTEMA DE OBTENCIÓN DE SEÑALES VISUALES PARA GENERACIÓN DE RUTAS DEL ROBOT MÓVIL

El control visual de robots o servoing hace referencia a los datos obtenidos por medio de algún dispositivo como celulares, cámaras, computadoras, estos datos usados para el control de un dispositivo como lo es un robot. El control de robots es un área amplia la cual a medida que pasa el tiempo debe tener un desarrollo constante, en esta investigación se abordará uno de los usos que se le puede dar a la visión por computadora y usarla en la navegación del robot móvil que se usará en el programa.

5.1. MÉTODOS DE CONTROL VISUAL

El control visual se clasifica por criterios tales como: el número de cámaras usadas, la configuración de sistemas de visión o según restricciones en el movimiento del robot. Las capturas visuales se pueden obtener de distintas maneras una de ellas es usando la cámara en el lugar de trabajo, la otra manera de obtener la información es colocando la cámara sobre el mismo robot, a continuación, se van a explicar las configuraciones de una manera más detallada.

- **CONFIGURACIÓN CÁMARA MONTADA SOBRE EL ROBOT**

La configuración de la cámara montada sobre el robot se puede apreciar en la Figura 35, se debe dar una relación entre la posición de la cámara, y la posición del robot ya que todos los movimientos que realice el robot también los efectúa la cámara por encontrarse sobre el mismo, la desventaja que existe en esta configuración se da cuando exista algún tipo de interferencia en la cámara, es decir cuando se deja sin visión al robot, ejemplo que se obstruya la visión con algún obstáculo de mayor tamaño o que tapen la lente de la cámara, en este caso se perdería la obtención de datos e imagen, lo que podría concluir en la perdida de comunicación, y dejando al robot sin ninguna instrucción a realizar [25].

Figura 35. Robot con cámara montada sobre él.

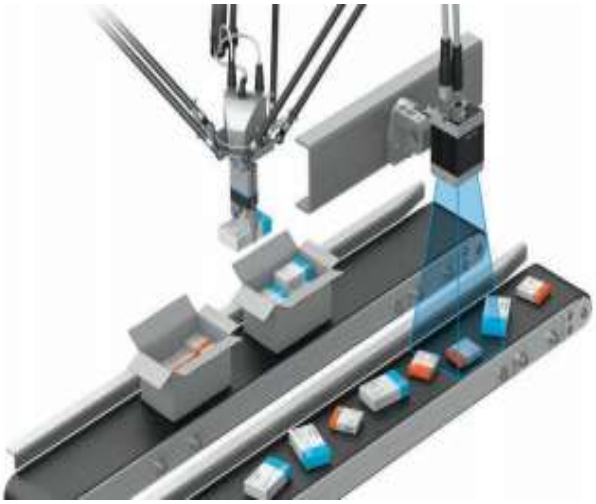
Fuente: [26].

- **CONFIGURACIÓN CÁMARA UBICADA SOBRE EL ESPACIO TRABAJADO**

La segunda configuración se trata de una situación en donde la cámara está puesta sobre algún lugar del área en donde se realizará el trabajo, en la Figura 36 se muestra un ejemplo en el cual se logra ver la cámara ubicada en una área específica del espacio de trabajo, en esta configuración la imagen capturada es independiente al trabajo realizado por el robot, esto quiere decir que el robot puede moverse libremente y no interfiere en el registro de video o fotográfico, es una configuración usada en los métodos de control visual, no hay relación mecánica entre el dispositivo de visión y el robot, pero el sistema de referencia del robot si es conocido por la cámara [25].

La información obtenida por las cámaras se extrae y es usada para sus respectivos algoritmos, esta información se puede utilizar para generar el control en los robots y a continuación se van a explicar 3 tipos de control basado en la información recolectada.

Figura 36. Cámara montada sobre un espacio trabajado.

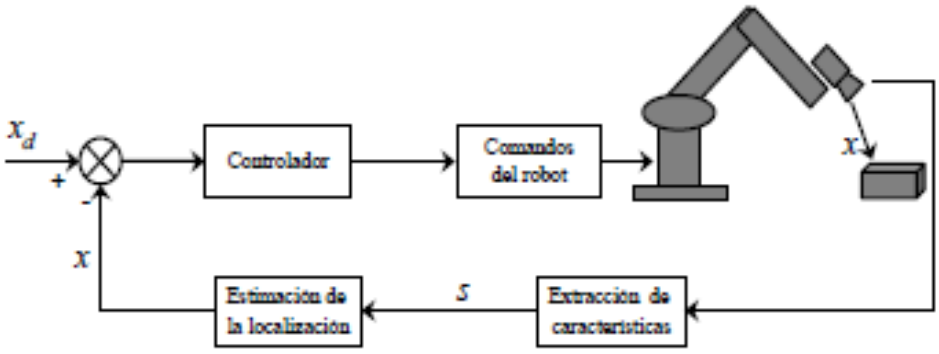


Fuente: [27].

- **CONTROL VISUAL BASADO EN POSICIÓN**

El control visual basado en posición inicia con la captura de la imagen, a continuación se extraen las características de la imagen para luego ser procesadas, esas características se vinculan con la información que se tiene del terreno y del robot, también se usa la información de la ubicación de la cámara, para así estimar la posición remota del robot respecto al sistema de visión, así finalmente un ordenador obtiene toda la información, entonces un algoritmo la ordena y la convierte en comandos, los envía a través de conexiones inalámbricas o alámbricas al robot móvil, estos comandos serán los que debe ejecutar el robot móvil, en la Figura 37 se observa el diagrama de funcionamiento de este tipo de control visual [25].

Figura 37. Control visual basado en posición.



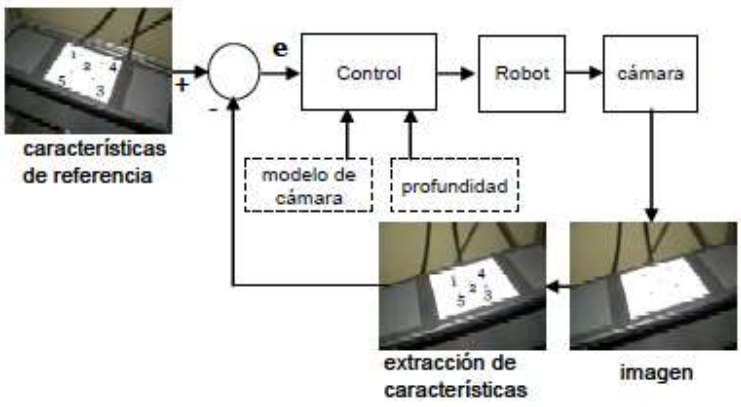
Fuente: [25].

- **CONTROL VISUAL BASADO EN IMAGEN**

El método el control visual se realiza extrayendo las características representativas de la imagen obtenida a través del sistema de visión, que representan la proyección del entorno en el plano 2D de la imagen. De esta manera, la proyección del entorno 3D no es necesaria, de esta forma el sistema está sujeto a cometer errores en sus modelos, que vienen directamente propios de las características de la imagen. En este método de control, la clave para su correcto funcionamiento es la matriz jacobiana de la imagen.

La matriz jacobiana establece una relación entre los cambios de posición y orientación del robot, estos cambios respecto al sistema de visión lo que produce cambio en las características de la imagen. La ventaja de este método de control es la precisión obtenida independiente del proceso de calibración y del modelo del robot. Entre los principales inconvenientes están: trayectoria de la cámara no controlable, no se puede asegurar la estabilidad global y la aparición de singularidades en el espacio de características, como oclusiones parciales, en la Figura 38 se puede ver el diagrama de funcionamiento de este tipo de control visual [28].

Figura 38. Control visual basado en imagen.



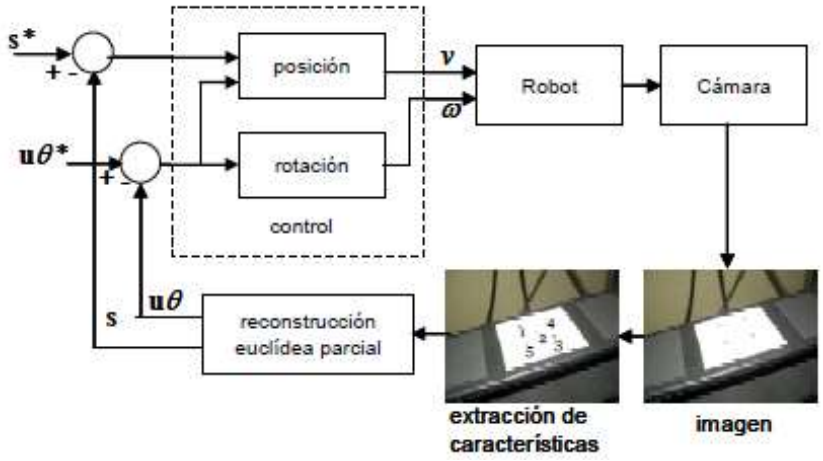
Fuente: [28].

- **CONTROL VISUAL HÍBRIDO**

El tipo de control visual hibrido adquiere las ventajas de los otros tipos de configuraciones, es el control visual con mejores condiciones de todos, y su funcionamiento básicamente se trata de tomar dos imágenes distintas, luego realiza una comparación de características entre ellas, lo que da resultado una relación entre posición y orientación del robot actual con la deseada. Se basan en el método de

Lyapunov para construir una estimación de rutas, de manera que con esto se compensa la posible falta de información que se puede dar por el uso de distintas cámaras [25].

Figura 39. Control visual Híbrido.



Fuente: [28].

5.2. VISIÓN ARTIFICIAL

Los robots móviles pueden equiparse con sistemas de visión, con esto adquieren la capacidad de “ver”, es decir que tendrán una inteligencia artificial, de modo que puede ser usada para poder desplazarse en cualquier entorno, en el que requiera ejecutar su tarea. El uso de visión artificial en robots se da por la necesidad de mejorar procesos, en donde es necesario un control complejo y una supervisión, la visión se da por etapas de obtención, caracterización e interpretación de la información obtenida gracias a las imágenes que se toman del entorno, a continuación, se explican las etapas:

5.2.1. FORMACIÓN DE IMÁGENES Y PREPROCESAMIENTO

Las imágenes se obtienen a través de los dispositivos como lo son cámaras, Kinect, GoPro entre otras, la calidad de la imagen capturada en ocasiones puede no ser la mejor, por lo anterior la información de las imágenes puede ser más compleja al momento de obtenerla, no obstante se puede mejorar la captura, para esto es necesario de dos procesos, el primero denominado “suavizado”, en el que se hace la eliminación de ruido que pueda existir en la imagen, mientras que el segundo llamado “realzado”, este elimina sombras y reflejos que no dejan obtener la información de la imagen clara [29].

5.2.2. SEGMENTACIÓN

El proceso de segmentación se trata de la división de la imagen obtenida en sus diferentes regiones, para poder realizar de manera correcta la segmentación se pueden emplear una serie de algoritmos, los cuales se basan en dos principios que son la similitud y la discontinuidad. Una región es un área de la imagen en donde se pueden presentar propiedades similares, así como colores, intensidad, los bordes son líneas que separan regiones de diferentes propiedades [29].

Para detectar en una imagen sus diferentes bordes o regiones, se necesita realizar una manipulación a la imagen, lo cual implica que cada píxel tendrá cierto valor para poder desagruparlos. Existen unos operadores usados para encontrar las discontinuidades o bordes en una imagen y son los siguientes:

- **OPERADORES PRIMERA DERIVADA Y SEGUNDA DERIVADA**

Los operadores de primera derivada realizan el trabajo de resaltar las zonas de la imagen en donde la intensidad no es uniforme, por otra parte, cuando existen lugares donde la intensidad es uniforme el operador es invalido o nulo. Mientras que los operadores de segunda derivada cumplen la función de cambiar el signo en los lugares de la imagen en donde puedan existir bordes, ahora bien, los lugares donde no existen bordes quedan con signo positivo [29].

- **OPERADORES MORFOLÓGICOS**

Los operadores morfológicos usan operaciones matemáticas, con esto pueden aprovechar las características encontradas por los operadores de primera y segunda derivada, también los resultados de la geometría integral y topológica. Con lo anterior, las imágenes ya se encuentran de una manera más simplificada y aun preservan las formas principales de los objetos [29]. Para describir las regiones de usan estas técnicas basadas en la similitud:

- **BINARIZACIÓN BASADA EN EL USO DE UMBRALES**

El procedimiento de binarización usa los niveles de umbrales, así podrá diferenciar los distintos tonos de la imagen, usando diferentes tipos de umbrales para cada tono en objetos y fondos de la imagen.

- **CRECIMIENTO DE REGIONES MEDIANTE LA ADICIÓN DE PIXELES**

El crecimiento de regiones se trata de agrupar pixeles y regiones de esta manera poder generar unas regiones de mayor tamaño, el procedimiento que realiza se basa en generar unos puntos centrales en los cuales se revisan los pixeles, con esto poder verificar si los pixeles vecinos tienen alguna característica similar que los haga unir, las características que revisa son los colores, texturas, intensidades entre otras, y con esto poder generar regiones más grandes.

- **DIVISIÓN DE REGIONES**

El proceso de división de regiones consiste en fraccionar la imagen en varias regiones al azar, de esta manera poder fusionar o separar las regiones, con lo anterior, poder obtener la información de la región completa.

- **SIMILITUD DE TEXTURA, COLOR O NIVEL DE GRIS**

El proceso de similitud realiza la agrupación de pixeles, siempre y cuando contengan características similares o iguales, luego de realizar las agrupaciones pertinentes el algoritmo selecciona los grupos de pixeles con características equivalentes, a diferencia del resto de los pixeles que no están entre los elegidos y son eliminados.

- **EXTRACCIÓN DE CARACTERÍSTICAS**

El proceso de extracción de características complementa lo realizado en el proceso de segmentación, en el anterior proceso se divide la imagen, de la misma forma, en la extracción de características lo que se hace es la obtención de las propiedades de cada división, y así poder proporcionarle una característica única para poder enviar la información al respectivo algoritmo.

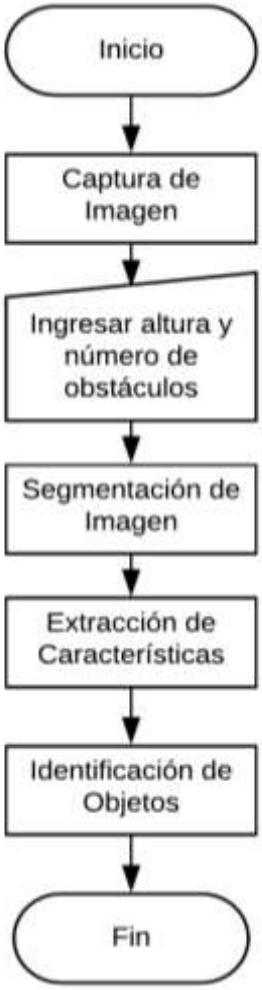
- **REPRESENTACIÓN Y CONOCIMIENTO**

El proceso de representación se hace el reconocimiento de las características obtenidas, para poder formar una composición de la escena, de esta manera luego que obtenga toda la información pueda ser enviada al dispositivo o algoritmo que se esté usando.

5.3. DESARROLLO DE LA INTERFAZ PARA LA OBTENCIÓN DE IMÁGENES

En el desarrollo de la interfaz para obtener las imágenes del entorno en donde se desplazará el robot, será necesario realizar la captura de la imagen, para poder realizar esto es necesario precisar con que dispositivo se hará la tarea, así después la imagen continuara con la segmentación de la misma, de esta manera se obtienen sus características principales, de manera que ya teniendo las identificaciones de los objetos se puede seguir a los respectivos cálculos, a continuación en la Figura 40 se puede apreciar el diagrama de flujo para la obtención de imágenes, la explicación del diagrama se hará posteriormente, en el ANEXO A se encuentra el código desarrollado de la función Capturar imagen.

Figura 40. Diagrama de flujo Función Captura de imagen.



Fuente: Autor.

- **TOOLBOX USADOS PARA EL PROGRAMA**

Una Toolbox es una función de Matlab desarrollada por personas particulares, existen cantidades de Toolbox y cada una fue diseñada para cumplir ciertas funciones, el usuario puede descargarlas e instalarlas en Matlab para su uso, en esta investigación se hará uso de estas herramientas, para así poder lograr una conexión estable entre los dispositivos usados como lo son la cámara, el robot LEGO Mindstorms y el ordenador.

Los Toolboxes que se usaran en este trabajo son:

- Image Acquisition Toolbox
- RWTH - Mindstorms NXT Toolbox
- Image Processing Toolbox

- **IMAGE ACQUISITION TOOLBOX**

Image Acquisition es una Toolbox, contiene todos los soportes o archivos necesarios para que el programa Matlab pueda detectar y reconocer los dispositivos o adaptadores que se requieran conectar para la adquisición de imágenes, esta también contiene los controladores de los dispositivos con las configuraciones necesarias para un correcto uso de los mismos, igualmente el Toolbox está diseñado con un conjunto de funciones que puede ser usadas y aplicadas a distintos algoritmos para el desarrollo de aplicaciones para la adquisición de imágenes.

5.3.1. SELECCIÓN DEL DISPOSITIVO PARA TOMA DE IMAGEN

Luego de realizada la instalación del Toolbox que reconocería los dispositivos, se procede a realizar la conexión de las cámaras con el ordenador, para así poder realizar pruebas, de esta manera seleccionar la cámara que mejor desarrolle las capturas y los videos en tiempo real, en la Figura 41, se aprecian los dispositivos usados en la investigación para poder realizar la captura de imagen, uno de los dispositivos usados es una cámara marca Genius, mientras que el segundo dispositivo es una cámara Creative.

Figura 41. Dispositivos usados para la toma de imagen.

Fuente: [30].

Se realizaron unas pruebas de funcionamiento para seleccionar el mejor dispositivo, una de las pruebas fue poder verificar cuál de las dos cámaras tiene una mejor resolución, otra prueba realizada fue poder observar cuál de las dos cámaras era más precisa en el reconocimiento de las figuras, obstáculos, robot, y meta, además de cuál de las dos cámaras realizaban una conexión más rápida y estable con el ordenador, en la Figura 42 se observa el entorno a trabajar.

Figura 42. Captura de pista sin reconocimiento.

Fuente: Autor.

Se iniciaron las pruebas con la cámara Creative, al realizar la prueba esta arrojo unos resultados erróneos como se puede ver en la Figura 43, en donde se puede observar

que la cámara dejo un obstáculo sin referenciar lo cual es complejo, además de apreciar que la cámara es sensible a tomar sombras como algún tipo de obstáculo falso, esto no se puede admitir en el algoritmo ya que el cálculo de la trayectoria estaría erróneo, por otro lado en la prueba realizada a la cámara Genius la cual se puede observar en la Figura 44, se puede ver que el dispositivo toma de forma correcta todos los obstáculos ubicados en el entorno, esto sin obtener falsos obstáculos por las sombras proyectadas en el espacio, y de esta manera se puede realizar el cálculo correcto de la trayectoria

Figura 43. Captura de pista con Cámara Creative.

Fuente: Autor.

Figura 44. Captura de pista con Cámara Genius

Fuente: Autor.

- **PROCESAMIENTO DE LA IMAGEN**

En el procesamiento de la imagen se realiza la segmentación de la captura, en donde lo que se busca es dividir la imagen para poder asociar las partes que tengan colores o aspectos similares y darles una característica, en este caso, lo que se desea es que el algoritmo identifique los obstáculos, la meta, y el robot, cada uno con un color distinto, para poder realizar esto se deben usar unas funciones que se encuentran incluidas en el programa Matlab estas son, medfilt2, imsubtract, imbinarize.

Las funciones anteriores son las que ayudan a la segmentación y reconocimiento de la misma, para realizar la caracterización de las distintas partes, a continuación, se explicaran de una manera más detallada.

- **MEDFILT2:**

Realiza un filtrado en la imagen capturada, en donde los pixeles de salida obtienen cierto valor promedio entre las casillas vecinas de los alrededores del pixel de entrada, en la Figura 45 se puede apreciar el funcionamiento del filtrado en 2 dimensiones, la imagen de la izquierda es la de entrada, mientras que la imagen de la parte derecha ya se puede observar con la función aplicada, se logra ver los colores más , suavizado en los bordes así como la diferenciación que se le realiza a los obstáculos, robot y meta [31].

Figura 45. Imagen segmentada usando la función Medfilt2.

Fuente: Autor.

- **IMSUBTRACT:**

La función Imsubtract toma elementos de la imagen y los resta con otra imagen, para así poder tener un elemento común o una constante de una nueva imagen, en la Figura 46 se puede apreciar el funcionamiento de la función, en donde se pueden ver resaltados los objetos usados para la generación de trayectorias, como lo son los obstáculos, la meta y el robot [32].

Figura 46 Imagen segmentada usando la función Imsubtract.

Fuente: Autor.

- **IMBINARIZE**

La función imbinarize genera una imagen con una escala de grises puede ser en 2 o 3 dimensiones, esto lo hace cambiando los valores que existen encima de un umbral general que se crea en la función globalmente, colocando un valor de 1s a los que están por encima, de esta forma todos los demás valores que no estén en ese valor serán 0s.

La función usa el método de Otsu, para minimizar la varianza de los píxeles en blanco y negro de umbral, en la Figura 47 se puede observar el funcionamiento de esta función, en la imagen izquierda se puede ver que tiene una cantidad de pixeles de distintos tonos, mientras que luego de usar la función se convierten únicamente en blanco y negro, demostrando que se eliminó la varianza de tonos e los pixeles, para que el algoritmo tome de manera más fácil los obstáculos y componentes del entrono [33].

Figura 47 Imagen segmentada usando la función Imbinarize.

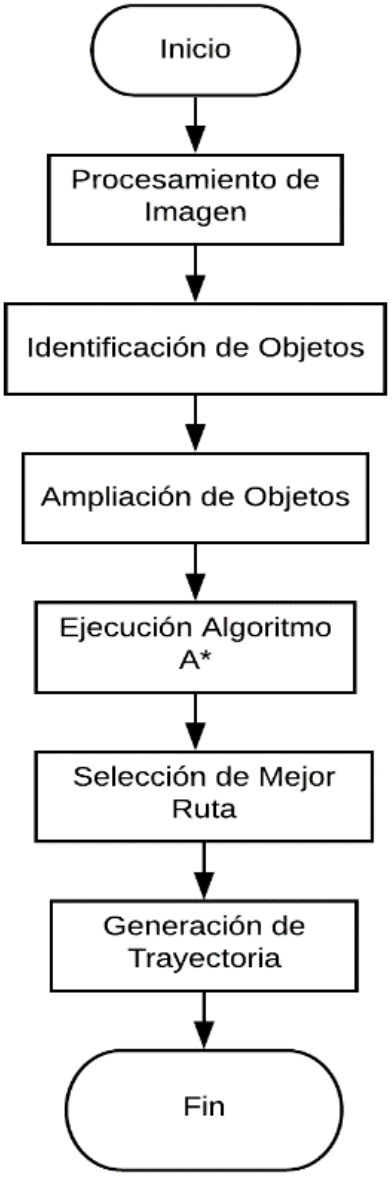
Fuente: Autor.

5.3.2. GENERACIÓN DE TRAYECTORIA

Luego de procesar la imagen se procede a realizar el código para generar la trayectoria que el robot va a ejecutar para llegar a su destino final, para esto fue necesario hacer el uso del algoritmo A*, junto con varias características que fueron necesarias para que el programa escogiera la ruta que realmente fuera la más eficiente, en la Figura 48 se puede observar el diagrama de flujo del algoritmo para la generación de trayectorias, en el ANEXO B, se puede observar el código desarrollado para la función cálculo de trayectoria.

A continuación, se describe el funcionamiento general del algoritmo y posteriormente se dará una explicación de los procedimientos realizados por el algoritmo.

Figura 48. Diagrama de Flujo Generación de Trayectoria.



Fuente: Autor.

- **IDENTIFICACIÓN DE LOS OBJETOS**

La identificación de los objetos se realiza generando un código específico para cada objeto, así pues el objetivo, los obstáculos, por último el robot, cada uno tendrá su respectiva identificación, partiendo de figuras geométricas se caracteriza cada uno de ellos, el programa se va a encargar de encerrar cada figura, y así poder hallar su centroide, este centroide tiene las coordenadas x, y del objeto en el entorno, estos datos son tomados por el algoritmo para saber la posición de cada uno de ellos.

Se tuvo en cuenta que cada objeto debe tener un color distinto, esto para que al momento de procesar la imagen fuera más cómodo realizar la diferenciación de cada objeto, de manera que el programa pueda tener la certeza en las decisiones que deberá tomar, también se incluyó en el programa un contador por cada color distinto, esto para evitar ciertos errores que podrían evitar el correcto funcionamiento.

La identificación de los objetos se realizó por colores y fue de la siguiente manera: el color rojo para el objetivo, el color negro para los obstáculos, los colores azul y verde para el robot que tiene dos partes distintas, en la Figura 49 se puede observar encerrado en el círculo rojo el contador de objetos del entorno, mientras que encerado de color amarillo se encuentra la identificación del centroide y ubicación en el entorno de la figura, y por ultimo encerrado de verde se encuentra la caracterización de las partes delantera y trasera del robot con sus respectivas ubicaciones.

Figura 49. Identificación de los objetos encerrados en los círculos

Fuente: Autor.

- **AMPLIACIÓN DE LOS OBJETOS**

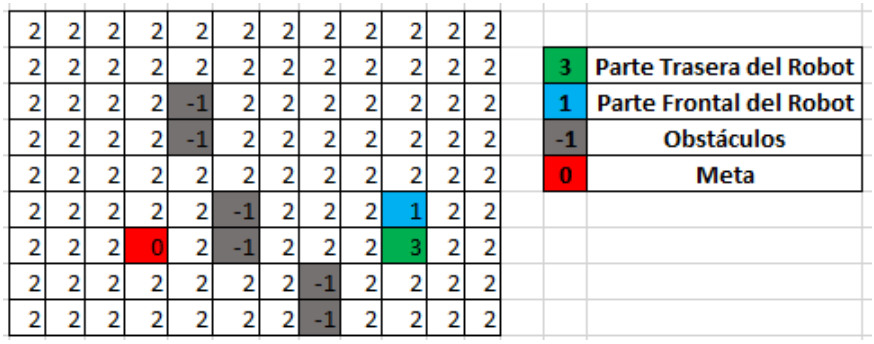
Un detalle para tener en cuenta es que el robot no puede tocar ninguno de los obstáculos que se encuentran en la pista, así que para que no ocurra esto se realizara una ampliación de los obstáculos, esto quiere decir que el programa entenderá que los obstáculos son más grandes y le dará al robot un espacio más amplio en el cual podrá superar la trayectoria sin ningún error.

Luego de realizar el procesamiento de imagen se genera una matriz la cual se llena con los datos en donde se ubican los objetos, cada objeto está siendo identificado con un número de la siguiente manera, los obstáculos con -1, el objetivo con 0 y por último el robot con los números 1 y 3, en la Figura 50 se puede apreciar la matriz con todos los objetos reconocidos y sin ampliación de obstáculos.

En el momento siguiente se realiza la ampliación de obstáculos, así la matriz cambia y se puede observar cómo los obstáculos ocupan más espacio en ella, haciendo referencia al espacio en donde podrá moverse el robot sin tocar los obstáculos, en la Figura 51 se puede observar la matriz luego de que el programa realizara la ampliación de los obstáculos.

- **Matriz sin Ampliación de obstáculos:**

Figura 50. Matriz sin ampliación de los obstáculos en negro

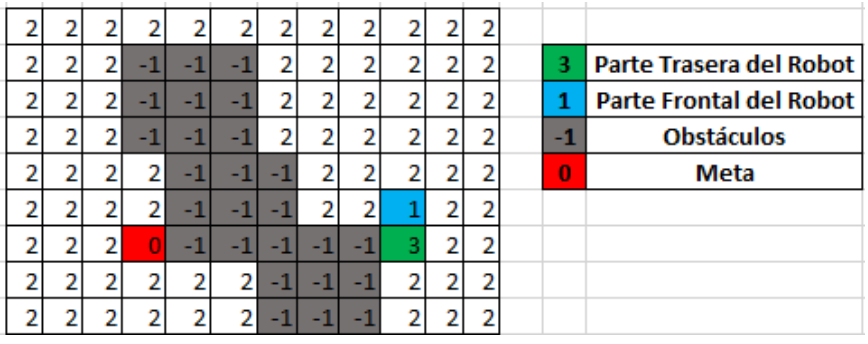


3	Parte Trasera del Robot
1	Parte Frontal del Robot
-1	Obstáculos
0	Meta

Fuente: Autor.

- **Matriz con Ampliación de obstáculos:**

Figura 51. Matriz con ampliación de los obstáculos en negro



3	Parte Trasera del Robot
1	Parte Frontal del Robot
-1	Obstáculos
0	Meta

Fuente: Autor.

5.3.3. ALGORITMO A*

Ahora bien cuando se tiene la información de los elementos, como su ubicación, también la matriz de ampliación de los objetos, con esto se puede determinar realmente cual es el espacio que tiene el robot para moverse, es el momento de incluir en el programa el algoritmo A*, el cual realiza la función de encontrar el camino optimo o más eficiente desde un punto inicial a uno final, para poder realizar esto el algoritmo genera distancias entre nodos, verificando cual es el nodo más económico para su movimiento, con esto que va guardándolo en una matriz a medida que avanza en su estimación, el algoritmo hace uso de la ecuación (1) para realizar los cálculos:

Función algoritmo Astar *

$$f(n) = g(n) + h(n) \qquad (1)$$

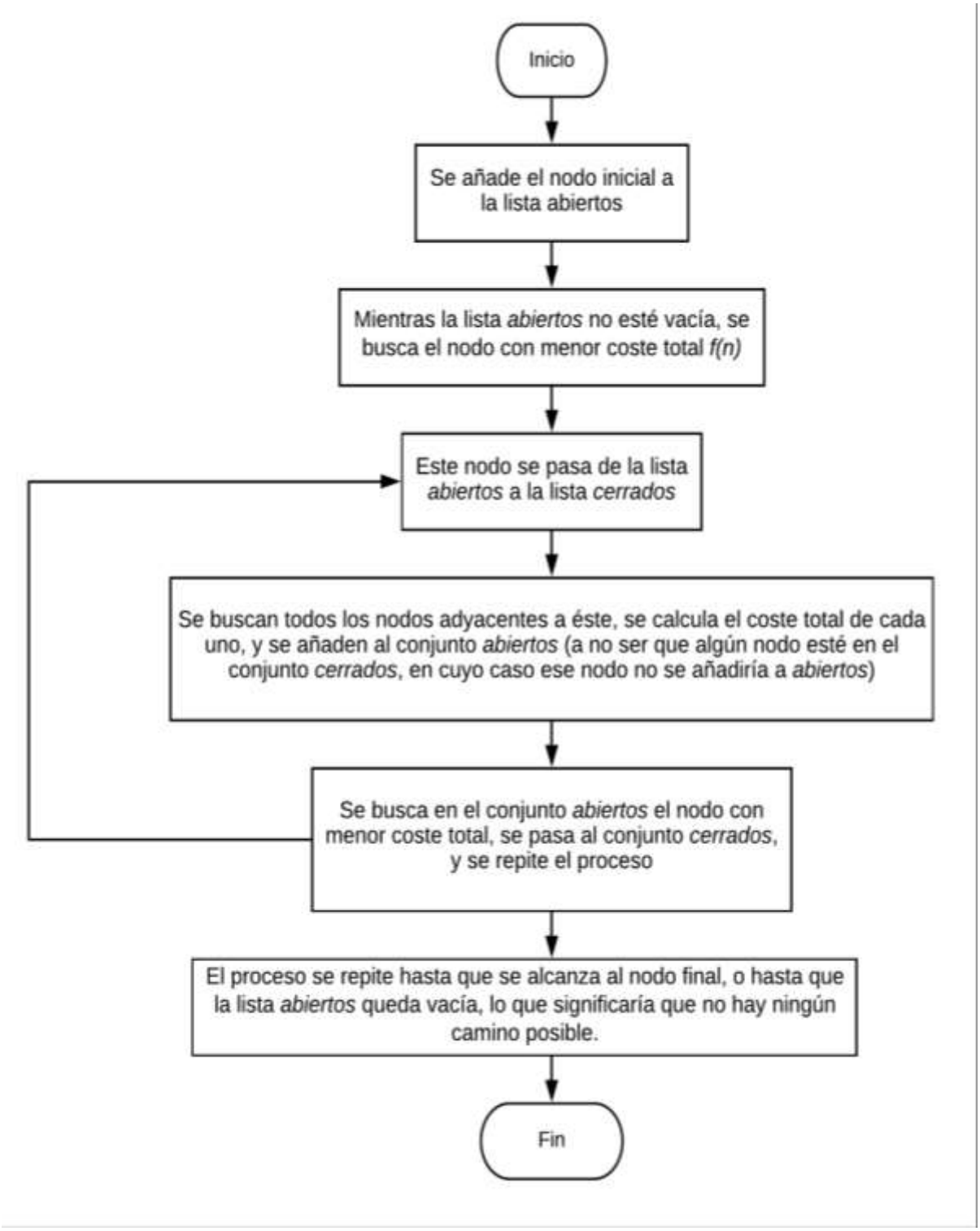
En donde:
f(n)= Suma de los costos.
g(n)= Indica la distancia del camino desde el nodo origen s al n.
h(n)= Expresa la distancia estimada desde el nodo n hasta el nodo destino t.

En la ecuación tenemos que h(n) es la heurística o distancia que existe entre el nodo que se está evaluando y la posición final o nodo final, mientras que g(n) es la sumatoria de los costos sabiendo que los costos significan el valor de ir del nodo inicial al nodo n, y se finaliza con f(n) que se calcula con la suma de la heurística con el costo que equivale ir hasta el nodo n, al final la trayectoria son todas las casillas que contengan los menores f(n).

El funcionamiento de el algoritmo A* se basa en tener 2 conjuntos en estos es donde se va a almacenar la información obtenida en cada iteración, uno de los conjuntos es abierto y este será en donde se almacenan los nodos que podrían ser usados en la trayectoria, aquellos que pueden ser visitados por el nodo anterior, por el contrario el segundo conjunto es cerrado este será el que guarda los nodos ya visitados, pero que pueden ser usados como trayectoria, a partir de lo anterior se planteó el diagrama de

flujo en el que el objetivo es cumplir con lo previamente expuesto, el diagrama planteado se puede ver en la Figura 52, y en el ANEXO C se puede observar el código desarrollado para el funcionamiento del algoritmo.

Figura 52. Diagrama de Flujo Algoritmo A*



Fuente: Autor.

- **GENERACIÓN DE MAPA**

Después de realizarse el procesamiento de la imagen y de la misma manera la ejecución del algoritmo A*, ya se tiene la información de la trayectoria más rápida para ir del inicio al objetivo, esta trayectoria se muestra inicialmente en una matriz en donde se tiene la información de los obstáculos, robot, objetivo, y ahora un nuevo elemento creado, se trata de la trayectoria elegida por el algoritmo identificada con el número 10, así como se puede observar en la Figura 53.

Figura 53. Matriz de programa generando trayectoria

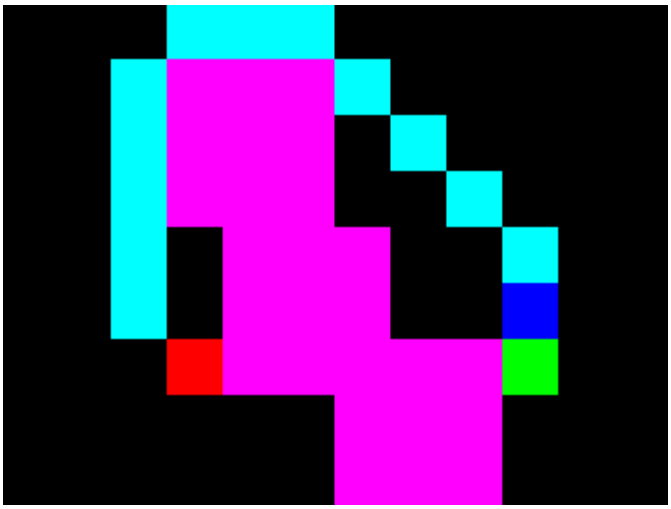
2	2	2	10	10	10	2	2	2	2	2	2
2	2	10	-1	-1	-1	10	2	2	2	2	2
2	2	10	-1	-1	-1	2	10	2	2	2	2
2	2	10	-1	-1	-1	2	2	10	2	2	2
2	2	10	2	-1	-1	-1	2	2	10	2	2
2	2	10	2	-1	-1	-1	2	2	1	2	2
2	2	2	0	-1	-1	-1	-1	-1	3	2	2
2	2	2	2	2	2	-1	-1	-1	2	2	2
2	2	2	2	2	2	-1	-1	-1	2	2	2

3	Parte Trasera del Robot
1	Parte Frontal del Robot
-1	Obstáculos
0	Meta
10	Trayectoria Robot

Fuente: Autor.

Pero para mostrar la trayectoria de una manera más entendible ya que ver una matriz de solo números podría ser no significativa, para poder mostrarla se usa una función llamada “imshow”, con esta función graficamos la matriz final o trayectoria, en forma de imagen procesada, igual que antes cada elemento tiene su distintivo color o caracterización para evitar confusiones, así se ve la ruta graficada.

Figura 54. Trayectoria generada en programa



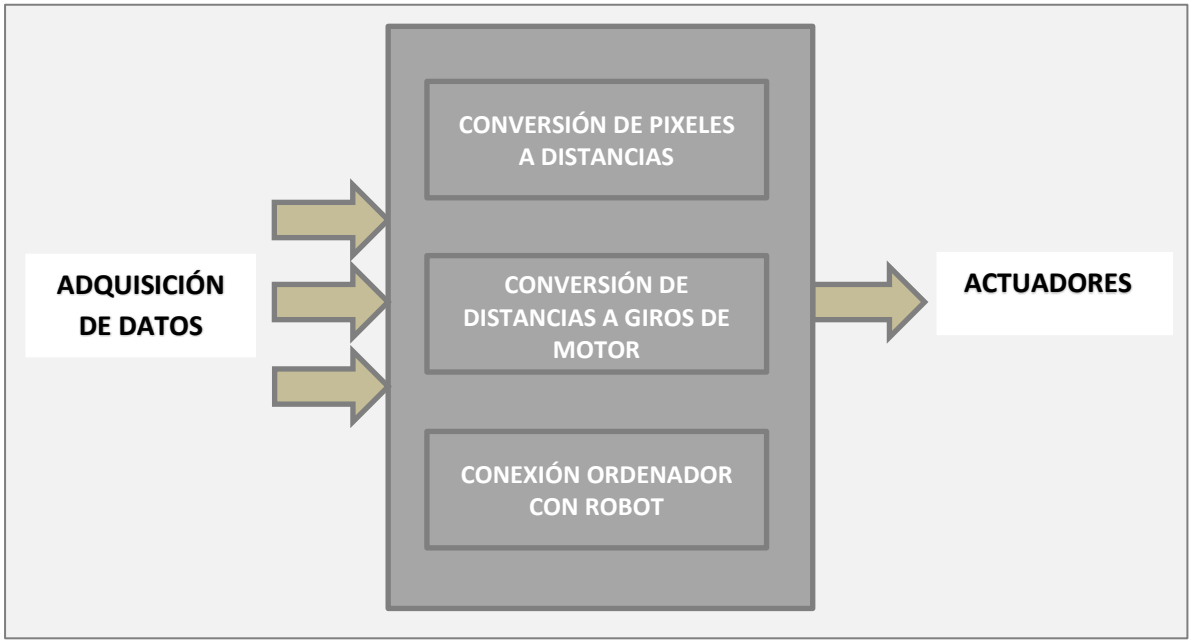
Fuente: Autor.

De la Figura 54 tenemos que el cuadro de color rojo es el objetivo, mientras que el azul es la parte delantera del robot, el verde es la parte trasera del robot, la parte fucsia son los obstáculos con la ampliación realizada, por último la parte negra hace parte del espacio libre el cual no se usa para nada.

6. DESARROLLO DE UN SISTEMA E INTERFAZ DE CONTROL PARA LOS ACTUADORES DEL ROBOT.

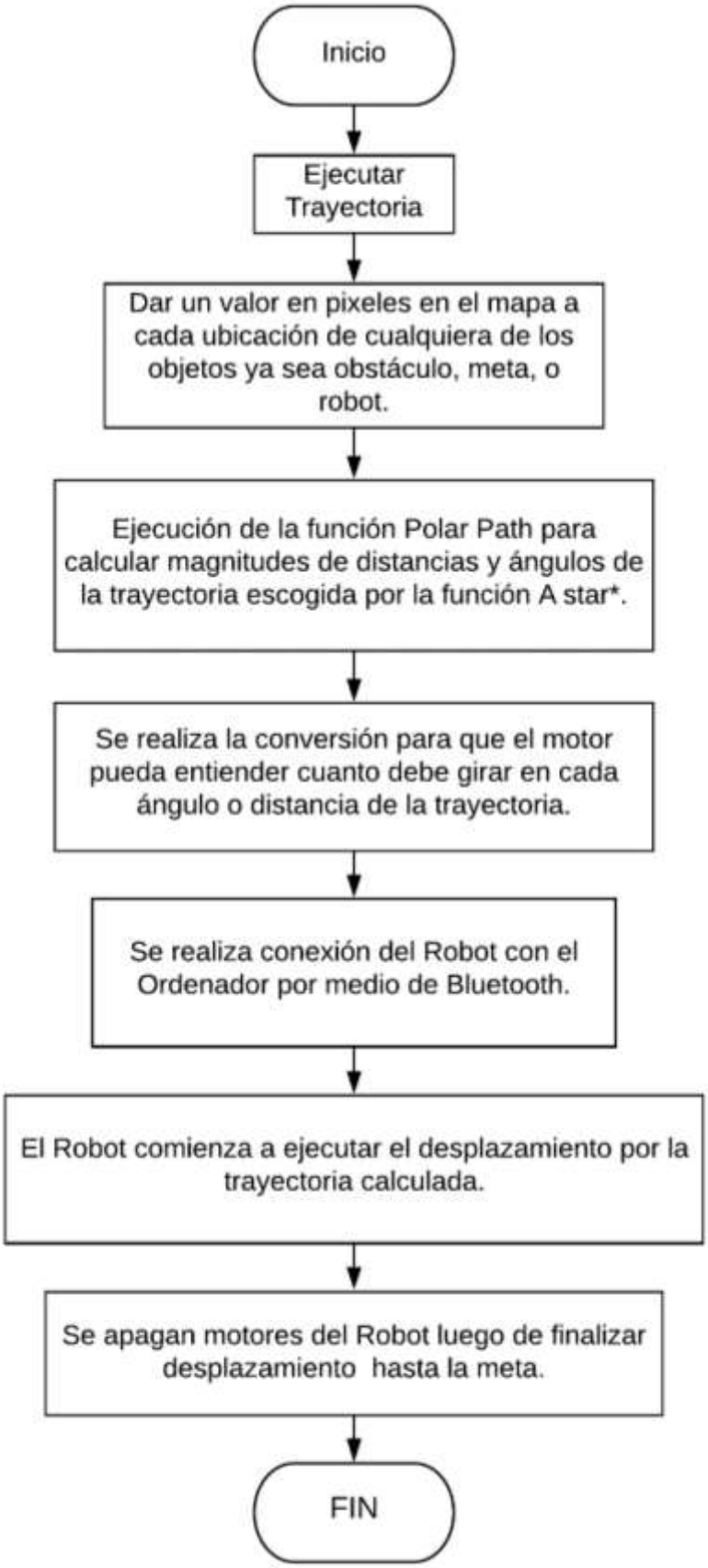
Para el desarrollo del sistema de control de los actuadores del robot es necesario la ejecución de funciones creadas, en la Figura 55 se puede observar el diagrama de la arquitectura de la función, con la adquisición de datos realizada anteriormente se puede realizar la conversión de pixeles a distancias reales, de esta manera después de tener todas las distancias realizar la conversión a giros del motor, esto para que el robot entienda cuanto debe desplazarse, así continuar con la conexión con el robot para poder enviar los comandos necesarios que el robot necesita, para terminar con la ejecución de la trayectoria y finalizar la ejecución del programa, en la Figura 56 se puede ver el diagrama de flujo del algoritmo usado para la ejecución ejecutar, y en el ANEXO D se puede observar el código desarrollado para la conexión de los actuadores del robot con el ordenador y la función Ejecutar trayectoria.

Figura 55. Diagrama de Función Ejecutar.



Fuente: Autor.

Figura 56. Diagrama de Flujo Función Ejecutar



Fuente: Autor.

- **HARDWARE DEL SISTEMA**

Los elementos que componen el hardware del sistema son el ordenador, el robot móvil, y la cámara. El ordenador quien es el que se encarga de ejecutar los programas necesarios para la conexión con el robot, el robot quien es el encargado de trazar la trayectoria que envía el ordenador, y la cámara quien es la encargada de tomar las imágenes y capturar el espacio en donde se desarrollara el evento [1].

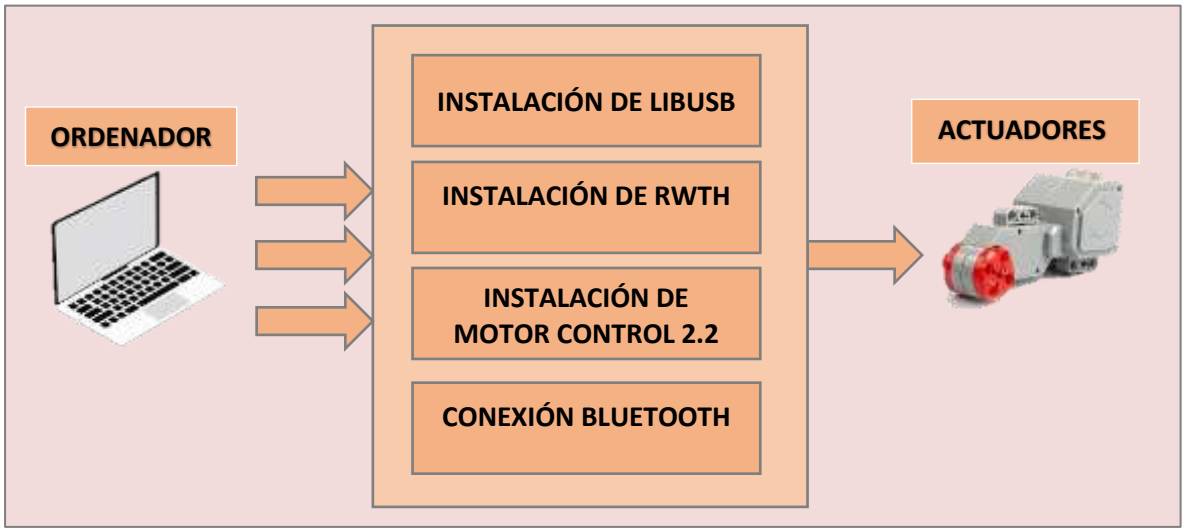
Los requerimientos mínimos del ordenador para poder ejecutar todos los programas y enviar las ordenes al robot de manera correcta son:

- Procesador Core2 Dúo 2 GHz o Amd Athlon.
- Tarjeta gráfica dedicada de 128 MB
- Tarjeta de conexión Bluetooth
- Sistema Operativo Windows XP o superior.
- 1 Gbyte de memoria RAM.

6.1. COMUNICACIÓN DEL ROBOT CON EL ORDENADOR

Para realizar la conexión entre el robot y el ordenador se necesita de un Toolbox, desarrollado especialmente para hacer la conexión entre el ladrillo LEGO Mindstorms NTX con los ordenadores, llamado RWTH - Mindstorms NXT Toolbox, además del Toolbox se necesita de un programa llamado Motor Control 2.2, el cual es usado para controlar los motores del robot desde Matlab, en la Figura 57 se puede observar el diagrama de comunicación entre el ordenador y el robot, además que la conexión se realizará inalámbricamente mediante comunicación Bluetooth.

Figura 57. Diagrama comunicación entre ordenador y Robot.



Fuente: Autor.

- **INSTALACIÓN LIBUSB**

Para que el ordenador pueda comunicarse con el Robot NXT es necesario realizar la instalación del programa LIBUSB, este programa permite el reconocimiento del ladrillo NXT en el ordenador y con esto pueda conectarse de manera directa cuando esté disponible.

- **INSTALACIÓN TOOLBOX RWTH- MINDSTORMS NXT**

El Toolbox RWTH – Mindstorms NXT contiene las funciones de MATLAB, usadas para realizar la conexión y comunicación del ordenador con el ladrillo NXT, esto permite una infinidad de posibilidades en creación de programas, ya que pueden combinarse varias Toolbox ya creadas en MATLAB de esta manera crear un código o programa completo, y poder ser usado por el robot, por medio del Toolbox y su conexión con el Ladrillo NXT, a continuación se pueden ver unas de las características que tiene el Toolbox:

- Abrir y cerrar conexiones Bluetooth o USB
- Enviar y recibir datos entre robots y MATLAB
- Control de motor NXT de alto nivel
- Lecturas de alto nivel del sensor NXT
- Comandos del sistema NXT

- Comandos directos de NXT
- Utilizar hardware común adicional, por ejemplo, cámaras web, joysticks, etc.
- Ejecutar programas MATLAB remotos para observar sensores, etc. mientras los programas NXT clásicos se ejecutan al mismo tiempo [34].

- **INSTALACIÓN MOTOR CONTROL 2.2**

Motor Control 22 es un programa encargado de comunicar los motores conectados en el Ladrillo NXT con el Toolbox RWTH Mindstorms, este se ejecuta cuando se le da la tarea desde MATLAB, para poder tener el programa se debe cargar primero al ladrillo.

- **COMUNICACIÓN ROBOT ORDENADOR BLUETOOTH**

El ladrillo inteligente NXT Mindstorms cuenta con un dispositivo de Bluetooth 2.0 en su configuración por defecto, esto le permite comunicarse con los distintos dispositivos como ordenadores, celulares, Tablet, también con otros robots los cuales también deben tener una versión de Bluetooth 2.0 o más avanzada para poder comunicarse, ya que el ladrillo solo admite esas versiones.

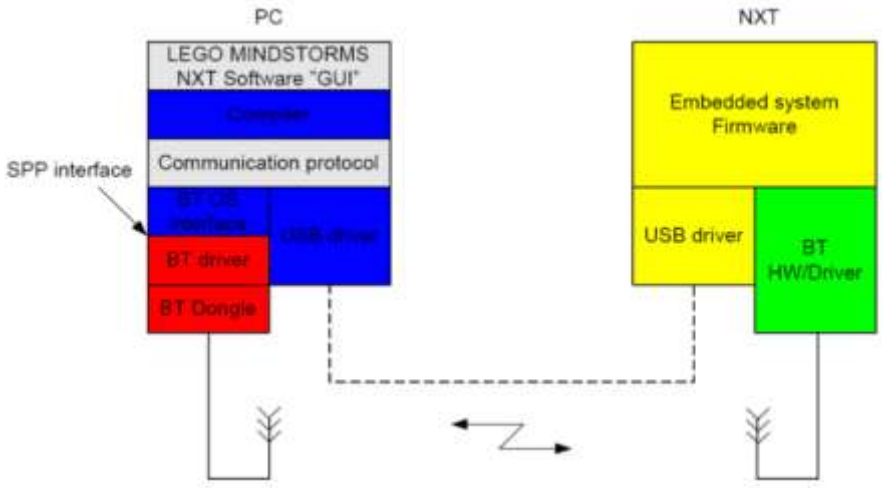
Para conectarlo al ordenador solo basta con prender el ladrillo, se procede a buscar en las configuraciones del ordenador y encender la búsqueda de los dispositivos, en el ordenador se selecciona el Ladrillo NXT, se instala el dispositivo de manera que se ve como en la Figura 58, con esto queda listo para realizar su función, no hay necesidad de instalar drivers ya que se instalan con el Toolbox anteriormente mencionado.

Figura 58. Ladrillo NXT conectado por Bluetooth.

Fuente: Autor.

Ya en la interfaz de Matlab se usa un código específico para iniciar la conexión entre el ordenador y el ladrillo NXT, en ese código se especifica si la conexión que se quiere realizar es mediante USB o Bluetooth, en la Figura 59 se puede observar el protocolo de comunicación entre ordenador y ladrillo NXT, en el proyecto se utilizara la conexión de tipo inalámbrica o Bluetooth.

Figura 59. Diagrama de bloques de comunicación entre Ordenador y Ladrillo NXT.



Fuente: [35].

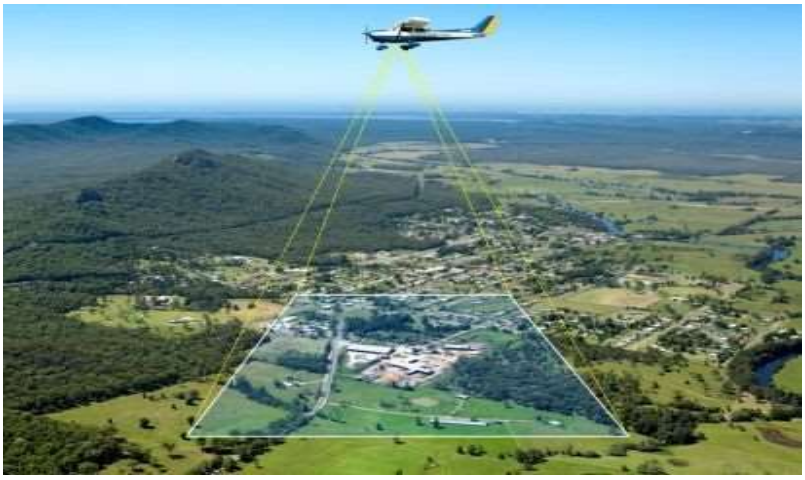
6.2. PROGRAMACIÓN DE SISTEMA DE CONTROL PARA ACTUADORES

Luego de realizada la conexión entre ordenador y Robot, se procede a realizar la programación del sistema de control, el cual va a tomar la información recolectada en los cálculos de la trayectoria, de esta manera poder convertir esa información en los movimientos que va a realizar el robot, para poder desplazarse a través del entorno sin ningún problema.

6.2.1. FOTOGRAMETRÍA

Fotogrametría es la técnica de obtención de información de una superficie física mediante la obtención de imágenes fotográficas y mediciones georreferenciadas del terreno, logrando precisión en la forma y dimensiones del espacio en estudio. En la Figura 60 se puede ver un ejemplo de fotogrametría aérea usada para la generación de mapas topográficos [36].

Figura 60. Ejemplo de Fotogrametría Aérea.



Fuente: Autor.

- **ESCALA FOTOGRAFÍA**

La escala es una relación existente entre las dimensiones de la fotografía tomada con las dimensiones reales del espacio estudiado, las fotografías se pueden considerar de escala pequeña o de gran escala, esto depende de un valor, las fotografías son de gran escala cuando el valor que hay en el denominador es menor a 10.000, por el contrario cuando este valor es mayor se dice que la fotografía es de una escala pequeña [37].

Los datos que se necesitan para obtener la escala de una fotografía aérea son: la distancia focal, y la altura del vuelo, con esto se puede obtener la escala haciendo uso de la ecuación (2):

Ecuación para hallar escala de la imagen o distancia focal

$$\frac{1}{E} = \frac{Df}{H} \qquad (2)$$

En donde:

Df= Distancia Focal.

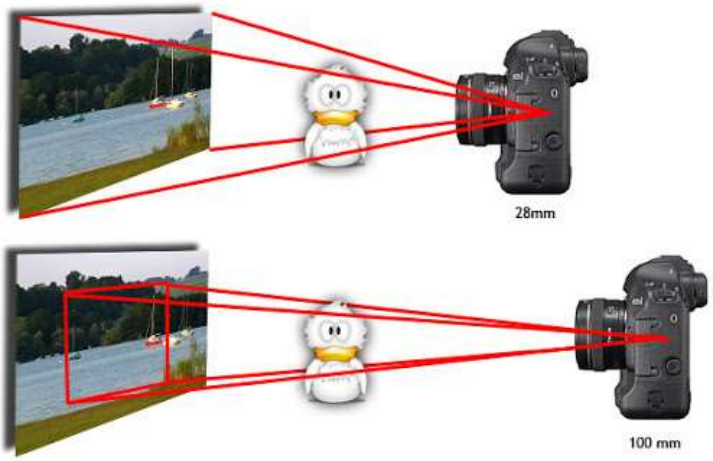
E= Escala

H= Altura de Vuelo.

- **DISTANCIA FOCAL**

La distancia focal es la distancia que hay entre el plano focal y el foco de la lente lugar en donde se cruzan los rayos de luz, entre más grande sea la distancia focal el zoom de la cámara será mayor, de esta manera tomara menos área de la captura o escena, por el contrario, si la distancia focal es pequeña eso significa que el zoom será menor y dejara ver mucha más área de la escena, a continuación, se muestra una imagen para explicar mejor esto:

Figura 61. Ejemplo de distancia focal



Fuente: [38].

En la Figura 61 se ve como una distancia focal menor deja ver toda la escena mientras que una distancia focal mayor deja ver menos escena.

6.2.2. FOTOGRAMETRÍA EN PROGRAMA

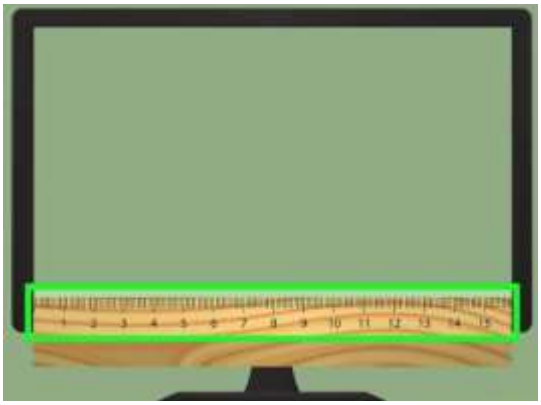
El algoritmo se basa en la fotogrametría aérea, de manera que la cámara estará localizada en una posición alta, haciendo relación como si se tratara de una fotografía desde un avión o dron, de esta manera poder realizar la toma de la escena en donde se encuentran los objetos ubicados.

Para obtener la escala de la imagen se registraron las medidas de la captura en la pantalla del computador, en la Figura 62 se puede ver como se realizó la medición en la pantalla del ordenador para tener una medida del espacio visto desde la interfaz del

ordenador, de la misma manera se realizó la medida del espacio real realizando la medición del espacio que tomaba la cámara, en el momento cuando el programa se encontraba en ejecución, en la Figura 63 se observa que se realiza la medición del espacio según el área captada por la cámara, luego de obtener las medidas el algoritmo las almacena.

- **MEDICIÓN DE CAPTURA EN PANTALLA**

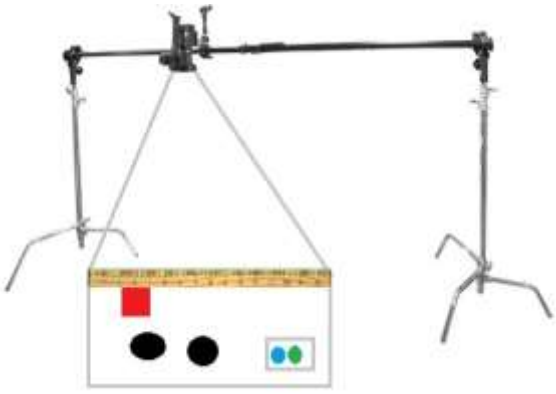
Figura 62. Medición del área de observación de la cámara en tamaño de pantalla.



Fuente: Autor.

- **MEDICIÓN DE CAPTURA EN EL PISO**

Figura 63. Medición del área de observación de la cámara en longitud real.



Fuente: Autor.

Luego de tener las medidas almacenadas en la memoria del algoritmo, se procede a introducir los datos en Excel para obtener una escala promedio entre la medida real y la medida en la interfaz, luego de tener la escala de la imagen se usa la ecuación (2) para hallar la distancia focal de la cámara, para hacer esto fue necesario realizar pruebas, con distintas alturas además de tener diferentes escalas esto para obtener un

valor correcto y general que funcionara con cualquier altura que el usuario requiera, con esto el algoritmo genere las mediciones de manera correcta.

Tabla 1. Tabla de primera prueba con altura de 1272 mm.

Altura Cámara, Z (mm)	X	Y	X	Y	X	Y	Escala Promedio	Distancia focal (f)
	Papel (mm)		Real (mm)		Pixeles			
1272	161	121	1020	765	640	480	6,3	200,98

Fuente: Autor.

Tabla 2. Tabla de segunda prueba con altura de 900 mm.

Altura Cámara, Z (mm)	X	Y	X	Y	X	Y	Escala Promedio	Distancia focal (f)
	Papel (mm)		Real (mm)		Escala			
900	161	121	721,0	541,8	4,5	4,5	4,5	200,98

Fuente: Autor.

Luego de realizadas las pruebas para hallar la distancia focal de la cámara como se puede ver en Tabla 1 y Tabla 2, se obtuvo el valor de 200.98 mm, este valor hace referencia a la distancia focal de la cámara, de esta manera este será el valor que se usara en el algoritmo para la correcta ejecución del programa.

- **FACTOR DE CONVERSIÓN PIXELES A MILÍMETROS**

Para poder hacer los cálculos de distancias reales que el robot va a tener que ejecutar, en el programa se va a realizar un factor de conversión para hacer el cambio de pixeles a milímetros, con esto poder obtener las distancias reales que el robot va a desplazarse a través de la pista.

Factor de conversión pixeles a milímetros

$$Fconv = \frac{Distancia\ real}{Resolucion\ (X,Y)} \quad (3)$$

En donde:

Fconv= Es el Factor de conversión de pixeles a milímetros.

Distancia Real= Distancia real del entorno.

Resolución (X, Y) = Distancias medidas en pixeles.

Tabla 3. Cálculo de factor de conversión con altura de cámara de 900 mm.

Altura Cámara, Z (mm)	X	Y	X	Y	X	Y	Escala Promedio	Distancia focal (f)	X	Y	Factor conv prm (mm/pixeles)
	Medidas pantalla (mm)		Real (mm)		Pixeles				Factor conv (mm/pixeles)		
900	161	121	721,0	541,8	640	480	4,5	200,98	1,13	1,13	1,13

Fuente: Autor.

El valor del factor se obtiene al hacer uso de la ecuación (3), así que después de obtener el factor de conversión en cada componente del sistema, se realiza un promedio de estas dos para obtener un valor único, este cálculo se añade al código del programa, de manera que este promedio se va a calcular en cada altura de cámara, ya que el valor promedio cambia respecto a la medida real espacio que se captura, esto quiere decir que en cada altura hay un factor de conversión distinto, esto se verifica al realizar las pruebas que se pueden ver en la Tabla 3 y Tabla 4, teniendo la certeza se procede a programar el algoritmo, el programa va a pedir los datos de la altura de localización en la que se encuentre la cámara y de esta manera poder calcular su factor de conversión.

Tabla 4. Cálculo de factor de conversión con altura de cámara de 1272 mm.

Altura Cámara, Z (mm)	X	Y	X	Y	X	Y	Escala Promedio	Distancia focal (f)	X	Y	Factor conv prm (mm/pixeles)
	Medidas pantalla (mm)		Real (mm)		Pixeles				Factor conv (mm/pixeles)		
1200	161	121	1020	765	640	480	6,0	200,98	1,59	1,59	1,59

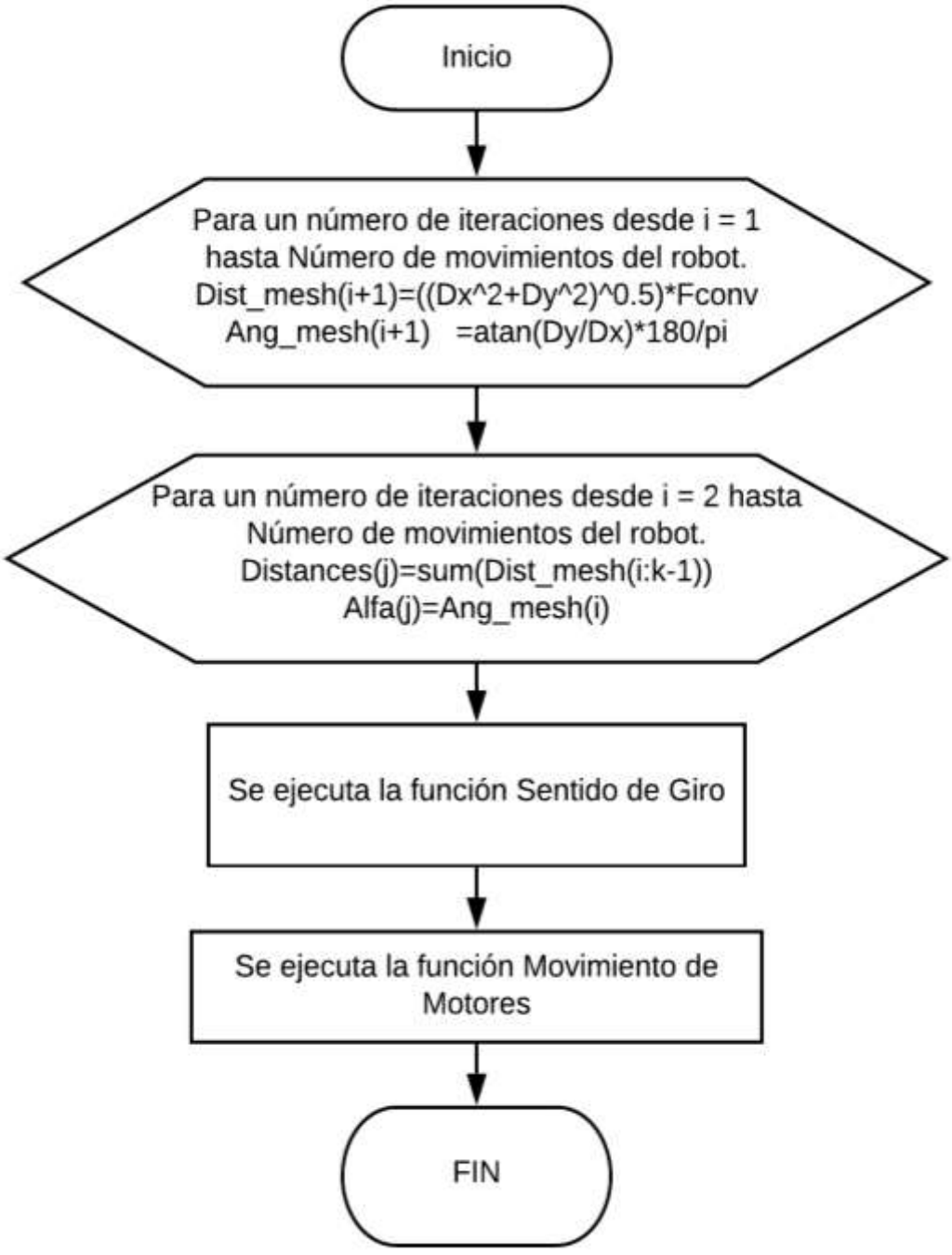
Fuente: Autor.

6.2.3. FUNCIÓN POLAR PATH

La función Polar Path es la encargada de calcular las magnitudes de: las distancias, los ángulos, de igual manera en que cuadrante se encuentra ubicado el robot y por ultimo hacia dónde va a moverse el robot, toda esta información la almacena en vectores, los cuales son creados en el algoritmo y el tamaño está relacionado con la cantidad de movimientos que va a realizar el robot, los vectores se guardan para después ser utilizados en nuevas funciones, usadas en la programación del algoritmo, en la Figura 64 se observa el diagrama de flujo de la función Polar Path, y en el ANEXO E se encuentra el código desarrollado para la función.

A continuación, se describe el funcionamiento general de la función Polar Path, posteriormente se dará una explicación de los procedimientos realizados por la función.

Figura 64. Diagrama de flujo función Polar Path.

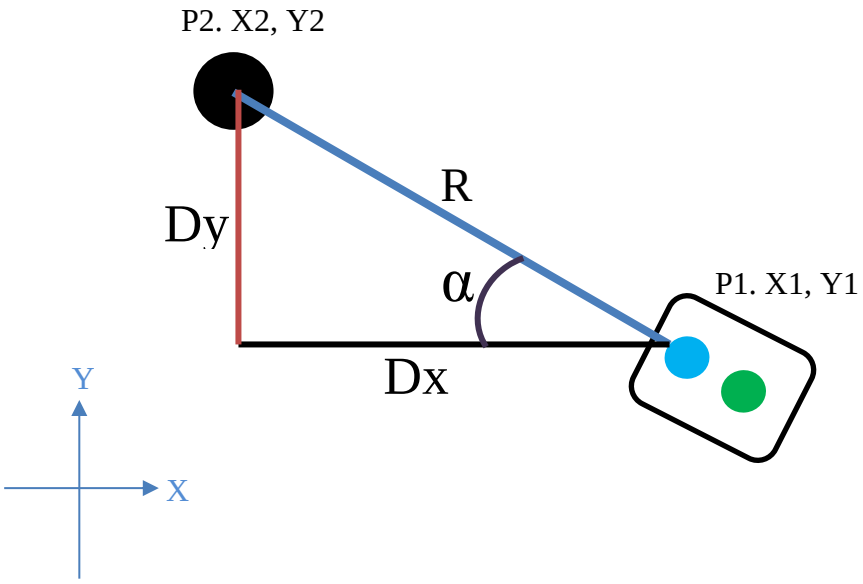


Fuente: Autor.

La función consiste en 5 casos especiales, los casos fueron analizados para así poder obtener toda la información necesaria del robot, de esta manera poder calcular todas las componentes necesarias para sus desplazamientos, los casos encontrados fueron:

1. Caso de cálculo en nodo delante del robot
2. Caso de cálculo para que el robot pise el nodo ubicado en la meta
3. Caso de cálculo para saber ubicación y cuadrantes del robot
4. Caso de cálculo para saber el nodo anterior que realizo el robot
5. Caso de cálculo giro inicial del robot

Figura 65. Cálculo de distancia R y ángulo α.



Fuente: Autor.

Como se puede observar la Figura 65, en esta función se hace el cálculo de la distancia R desde el punto frontal del robot (P1. X1, Y1), al siguiente nodo de la trayectoria (P2. X2, Y2), igualmente se hace el cálculo del ángulo (α), lo anterior se realiza por cada nodo que será visitado por el robot, todos estos valores parcialmente se encuentran en pixeles, luego de ser guardados en los vectores programados desde el inicio de la función, el programa automáticamente usa el factor de conversión para convertir estos valores a coordenadas polares (ángulos en ° y distancias en mm), ya realizado el cambio se guardan los vectores nuevamente con los valores reales.

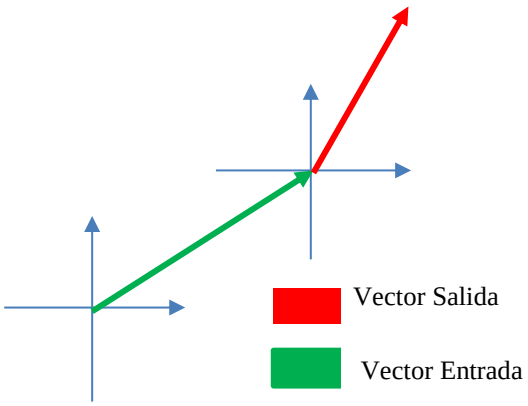
Después de realizar los cálculos el programa hace el llamado de 2 funciones más llamadas Sentido de giro, y Movimiento motor, las cuales serán explicadas más adelante.

- **FUNCIÓN SENTIDO DE GIRO**

Luego de obtener los vectores de distancias R y ángulos α en la función Polar Path, se procede a realizar los cálculos de ángulos Δ, para poder darle el sentido de giro a cada ángulo, para que de esta manera el robot conozca los desplazamientos que debe realizar, para poder obtener los datos se realizó una serie de rutas o trayectorias distintas en papel, una de las rutas se puede observar en la Figura 72, esto se hizo para buscar algún patrón que hiciera más fácil la programación del algoritmo, realizadas todas las pruebas de papel, se encontraron 5 casos y sus respectivas ecuaciones las cuales se repetían en varias de las trayectorias, de manera que se encontró un patrón, estas ecuaciones serán usadas en el programa para encontrar los datos buscados, a continuación se van a exponer a fondo los 5 casos encontrados, en el ANEXO G se encuentra el código desarrollado para la función Polar Path.

1) Si el vector de llegada como el vector de salida se encuentran en el mismo cuadrante, hacer uso de la ecuación (4) para hallar Δ:

Figura 66. Vector de llegada en cuadrante 1 y vector de salida en el mismo cuadrante.



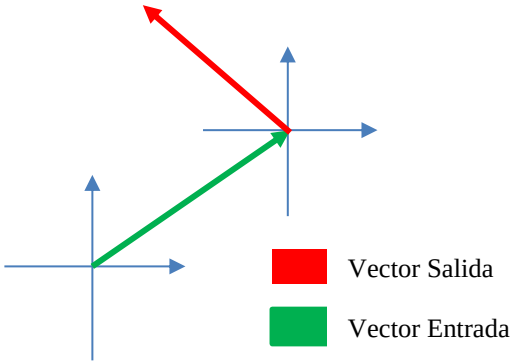
Fuente: Autor.

Primer caso de función Polar Path.

$$\Delta = \alpha_{salida} - \alpha_{llegada} \quad (4)$$

2) Si el vector de llegada o el vector de salida se encuentran en los cuadrantes (1 y 2) o (3 y 4), hacer uso de la ecuación (5) para hallar ∆:

Figura 67. Vector de llegada en cuadrante 1 y vector de salida en cuadrante 2.



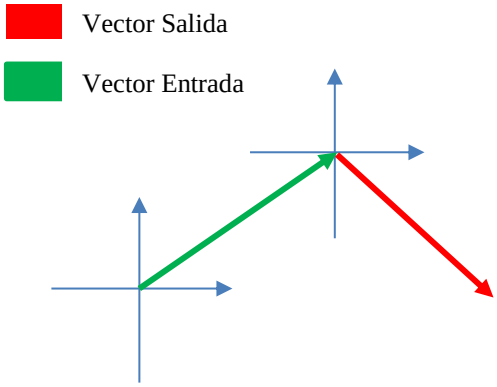
Fuente: Autor.

Segundo caso de función Polar Path.

$$\Delta = \alpha_{salida} + \alpha_{llegada} \quad (5)$$

3) Si los vectores de llegada o vector de salida se encuentran en los cuadrantes (2 y 3) o (1 y 4), hacer uso de la ecuación (6) para hallar ∆:

Figura 68. Vector de llegada en cuadrante 1 y de salida en cuadrante 4.



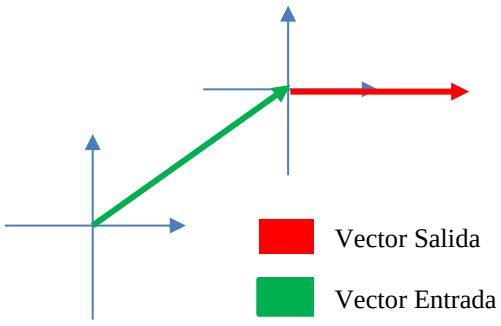
Fuente: Autor.

Tercer caso de función Polar Path.

$$\Delta = 180 - \alpha_{salida} - \alpha_{llegada} \quad (6)$$

4) Si alguno de los vectores viene o va de un eje coordenado y el otro vector viene o va hacia un cuadrante, hacer uso de la ecuación (7) para hallar ∆:

Figura 69. Vector de llegada en cuadrante 1 y vector de salida en eje coordenado.



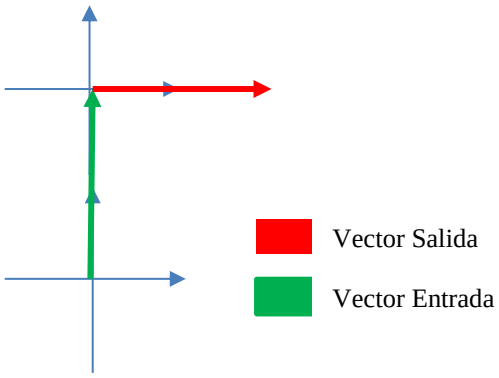
Fuente: Autor.

Cuarto caso de función Polar Path.

$$\Delta = \alpha_{llegada} - \alpha_{salida} \quad (7)$$

5) Si los vectores vienen o van de ejes coordenados a ejes coordenados, hacer uso de la ecuación (8) para hallar Δ:

Figura 70. Vector de llegada y de salida en Ejes coordenados.



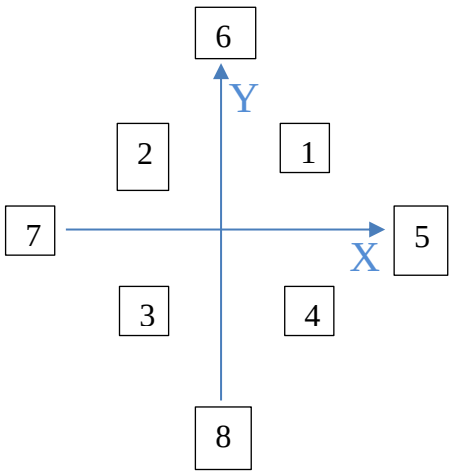
Fuente: Autor.

Quinto caso de función Polar Path.

$$\Delta = 90° \qquad (8)$$

Estos son los 5 casos de comportamientos que puede tener el robot en esta investigación, como se pudo observar en las imágenes anteriores se habló de vectores y de cuadrantes, esta información es importante, ya que en el programa existe una función, esta es la que se encarga de guardar la información de los cuadrantes, de entrada y de salida de la trayectoria, es decir en donde se encuentra el robot y el siguiente nodo a donde debe moverse, esta función está regida por una identificación que se le da a los cuadrantes, en el ANEXO F se puede observar el código desarrollado para la función, y en la Figura 71 se puede observar la identificación dada, esta función es importante, puesto que según la ubicación actual y el cuadrante hacia donde se deba mover el robot se dará el sentido de giro a cada ángulo calculado anteriormente.

Figura 71. Numeración de cuadrantes en programa.



Fuente: Autor.

- **SENTIDO DE GIRO SEGÚN CUADRANTE**

En la Figura 72, se observa una ruta en la que se presentan los principales casos del código, allí se observa el funcionamiento del código, lo primero que se necesita hacer es que el robot realice un giro inicial para configurar su posición respecto al siguiente nodo, luego de realizar ese giro se revisa el vector de ángulos α generado en la función Polar Path y en la función Sentido de Giro, los ángulos que se van a usar son el ángulo α de llegada con el ángulo α de salida, así como se puede observar en la Tabla 5 en donde se ve los distintos sentidos de giro que puede realizar el robot según sus cuadrantes.

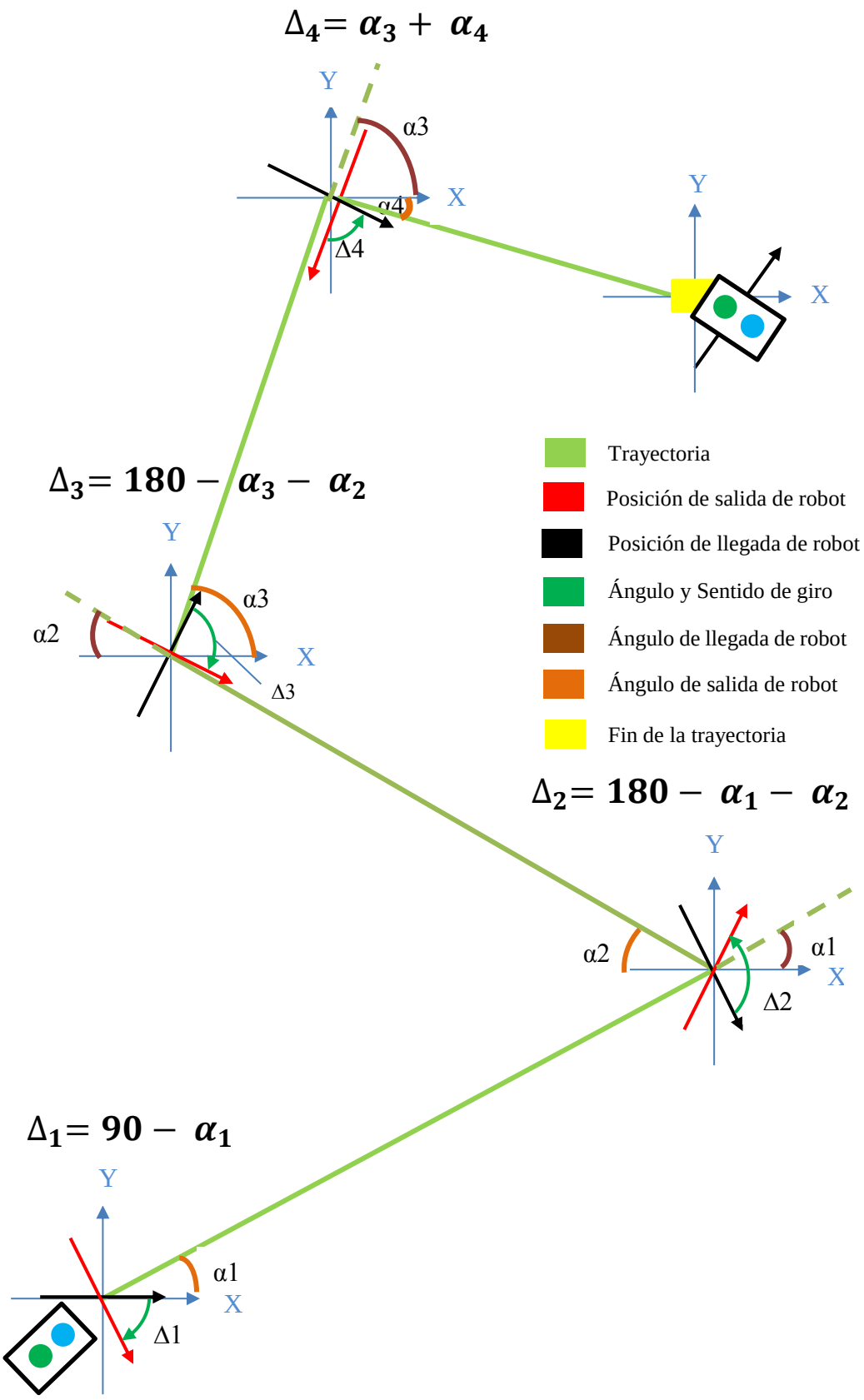
Para obtener los datos se usó una lógica, la cual trata de que en cada nodo se hace el desplazamiento del eje coordenado, con esto se obtienen los datos del ángulo α de llegada junto con el ángulo α de salida, además de saber los cuadrantes que usara en cada tramo de trayectoria, esta información se reúne de esta manera se procede a calcular los ángulos totales o Δ, estos serán los ángulos que va a girar el Robot, los ángulos se calculan según el caso que corresponda usando la ecuación correspondiente al caso, siempre están comprometidos el ángulo de llegada junto con el ángulo de salida, esto dará como resultado el ángulo Δ que va girar el robot.

Tabla 5. Sentidos de giro del Robot

Cuadrante Llegada	Cuadrante Salida	Sentido de Giro
1	2,6,7	Giro con Sentido Anti horario
1	4,5,8	Giro con Sentido Horario
2	3,7,8	Giro con Sentido Anti horario
2	1,5,6	Giro con Sentido Horario
3	4,5,8	Giro con Sentido Anti horario
3	2,6,7	Giro con Sentido Horario
4	1,5,6	Giro con Sentido Anti horario
4	3,7,8	Giro con Sentido Horario
5	1,2,6	Giro con Sentido Anti horario
5	3,4,8	Giro con Sentido Horario
6	2,3,7	Giro con Sentido Anti horario
6	1,4,5	Giro con Sentido Horario
7	3,4,8	Giro con Sentido Anti horario
7	1,2,6	Giro con Sentido Horario
8	1,4,7	Giro con Sentido Anti horario
8	2,3,5	Giro con Sentido Horario
5	5	Movimiento a 0° Sin giros
6	6	Movimiento a 0° Sin giros
7	7	Movimiento a 0° Sin giros
8	8	Movimiento a 0° Sin giros
1	1	Movimiento que puede tener alguno de los 2 sentidos
2	2	Movimiento que puede tener alguno de los 2 sentidos
3	3	Movimiento que puede tener alguno de los 2 sentidos
4	4	Movimiento que puede tener alguno de los 2 sentidos

Fuente: Autor.

Figura 72. Una de las rutas realizadas para hallar ecuaciones y de allí sacar ángulos y sentidos de giro del Robot.



Fuente: Autor.

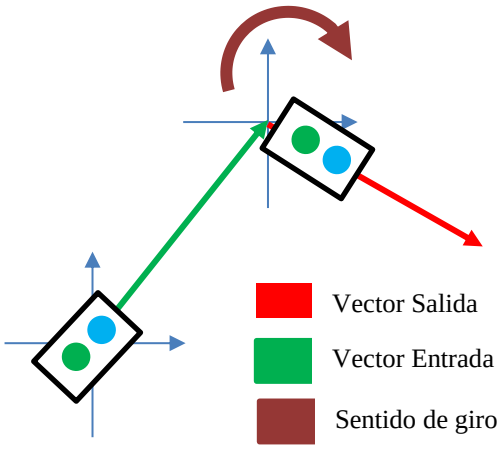
- **FUNCIÓN MOVIMIENTOS DE MOTORES**

La función se ejecuta luego de tener los datos de los vectores de distancias R y ángulos ∆, obtenidos con las funciones anteriores, sabiendo el sentido de giro se procede a crear el vector de movimientos de motores, el cual tendrá la información de cuál de los motores debe prender en cada caso, en la Tabla 6 se ve la manera correcta de encender los motores según el cuadrante de llegada y de salida, en el ANEXO H se encuentra el código desarrollado para la función Movimiento de motores.

Se debe aclarar que los movimientos del Robot serán de la siguiente manera, el robot debe: girar para alinearse con la trayectoria, frena cuando llegue a la distancia propuesta, de nuevo se moverá el tramo necesario según el algoritmo, así sucesivamente hasta finalizar el trayecto, así pues, obteniendo los valores de que motor debe encender ya puede comenzar a ejecutar la trayectoria deseada, para poder programar el algoritmo se escogió una nomenclatura para cada movimiento como se ve a continuación.

- **NOMENCLATURA MOTORES:**
  - Para encender el motor derecho se usó “1”
  - Para encender el motor izquierdo se usó “-1”
  - Para encender los dos motores se usó “0”

Figura 73. Explicación del sentido de giro del Robot.



Fuente: Autor.

En la Figura 73, se puede ver que el vector de entrada está en el cuadrante 1 y el vector de salida está en el cuadrante 4, si vemos la Tabla 6, podemos observar que se trata del segundo caso, en el que se solamente encendería el motor izquierdo, para así poder realizar el giro deseado para continuar con la trayectoria el Robot.

Tabla 6. Encendido o Apagado de motores según el giro y el cuadrante

Cuadrante Llegada	Cuadrante Salida	Motor Izquierdo	Motor Derecho
1	2,6,7	Apagado	Encendido
1	4,5,8	Encendido	Apagado
2	3,7,8	Apagado	Encendido
2	1,5,6	Encendido	Apagado
3	4,5,8	Apagado	Encendido
3	2,6,7	Encendido	Apagado
4	1,5,6	Apagado	Encendido
4	3,7,8	Encendido	Apagado
5	1,2,6	Apagado	Encendido
5	3,4,8	Encendido	Apagado
6	2,3,7	Apagado	Encendido
6	1,4,5	Encendido	Apagado
7	3,4,8	Apagado	Encendido
7	1,2,6	Encendido	Apagado
8	1,4,7	Apagado	Encendido
8	2,3,5	Encendido	Apagado
5	5	Encendido	Encendido
6	6	Encendido	Encendido
7	7	Encendido	Encendido
8	8	Encendido	Encendido
1	1	Puede encender o quedar apagado	Puede encender o quedar apagado
2	2	Puede encender o quedar apagado	Puede encender o quedar apagado
3	3	Puede encender o quedar apagado	Puede encender o quedar apagado
4	4	Puede encender o quedar apagado	Puede encender o quedar apagado

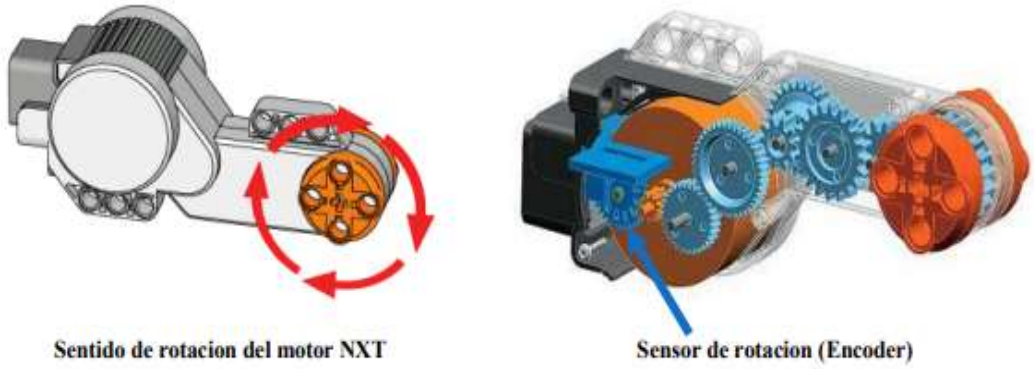
Fuente: Autor.

6.2.4. CONVERSIÓN TICKS DE MOTOR A LONGITUD Y GRADOS

Obteniendo los vectores de Distancias y Ángulos completamente llenos con los valores de cada desplazamiento que realizará el Robot, se procede a convertir la información recolectada en valores o datos que los motores puedan entender, y para esto se usará un factor de conversión tanto para los ángulos como para las distancias.

Este factor de conversión se obtuvo ya que los motores LEGO tienen un encoder o sensor, este viene en el interior del motor como se ve en la Figura 74, con el cual se puede hacer la medición de los grados o vueltas que hace el motor en cada movimiento, en el programa se usa una variable llamada “QuarterTurnTicks”, esta es la variable que será manipulada para encontrar el factor de conversión preciso para que los movimientos del Robot sean precisos.

Figura 74. Ilustración de sensor de rotación o encoder en un Motor Lego



Fuente: [24].

Para poder realizar la conversión de Ticks del motor a longitud de la misma manera a ángulos, fue necesario realizar unas pruebas, las cuales consistían en ubicar el robot en una misma posición inicial en todas las pruebas, se colocaba un flexómetro en la parte lateral de la pista, se comienza a dar valores a la variable QuarterTurnTicks en el programa, se ejecutaba a continuación se verificaba cuanta distancia se desplazaba el Robot, de la misma manera se obtuvieron datos para los ángulos, de esta manera teniendo unos valores se realizó una regla de tres para obtener el valor utilizado para realizar las conversiones, con estos valores se generaron la Tabla 7 y la Tabla 8 en donde se observan algunos valores de distancia y ángulos con sus respectivos Ticks para los motores.

La prueba realizada para obtener el factor de conversión de longitud fue:

Si 24 cm corresponden a 500 Ticks del motor entonces se realiza regla de tres para verificar cuantos Ticks correspondes a 1 cm y este fue el resultado:

Factor de conversión de longitud a Ticks.

$$Factor\ de\ conversión\ longitud = \frac{1\ cm * 500\ Ticks}{24\ cm} \quad (9)$$

$$Factor\ de\ conversión\ longitud = 20.83\ Ticks/cm$$

Tabla 7. Diferentes longitudes con su equivalencia en Ticks

Longitud (mm)	Factor Conversión (Ticks/mm)	QuarterTurnTicks
1	2,083	2,083
5	2,083	10,415
10	2,083	20,830
15	2,083	31,245
20	2,083	41,660
25	2,083	52,075
30	2,083	62,490
35	2,083	72,905
40	2,083	83,320
45	2,083	93,735
50	2,083	104,150

Fuente: Autor.

La prueba realizada para obtener el factor de conversión de los ángulos fue:

Si 90° corresponden a 540 Ticks del motor entonces se realiza regla de tres para verificar cuantos Ticks correspondes a 1° y este fue el resultado:

Factor de conversión de ángulos a Ticks.

$$Factor\ de\ conversión\ ángulos = \frac{1\ ° * 540\ Ticks}{90°} \quad (10)$$

$$Factor\ de\ conversión\ ángulos = 6\ Ticks/°$$

Tabla 8. Diferentes números de ángulos con su equivalencia en Ticks

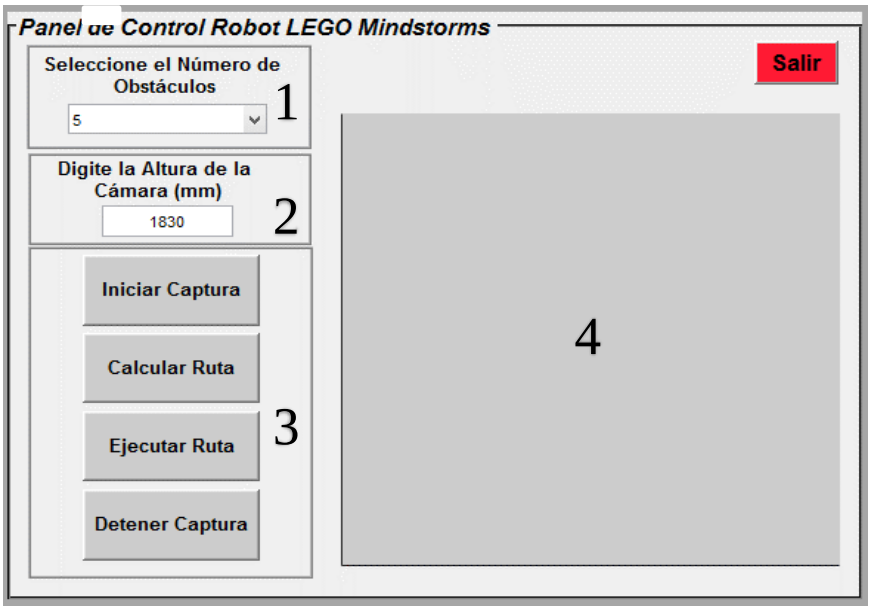
Ángulos (°)	Factor Conversión (Ticks/°)	QuarterTurnTicks
15	6	90
30	6	180
45	6	270
60	6	360
75	6	450
90	6	540
105	6	630
120	6	720
135	6	810
150	6	900
165	6	990
180	6	1080

Fuente: Autor.

6.3. DESARROLLO DE LA INTERFAZ DE USUARIO PARA LA APLICACIÓN

En el desarrollo de la interfaz se reúnen todas las funciones o algoritmos realizados en cada capítulo, cada función principal tiene un botón para la interacción del usuario con el programa, la interfaz de usuario desarrollada para la aplicación se puede observar en la Figura 75, esta interfaz consta de 4 partes, cada una tiene su respectiva función, la primera es la zona en donde el usuario podrá seleccionar la cantidad de obstáculos a utilizar en la trayectoria, la segunda parte es la zona en donde el usuario escribe la altura a la cual estará ubicada la cámara respecto al espacio, esta medida estará dada en milímetros, en la tercera zona se tienen los botones para accionar los dispositivos usados en el programa como los son, el robot, la cámara, y en la cuarta zona de la interfaz se observa la parte en donde se mostrara el diagrama de la trayectoria optima que será realizada por el robot.

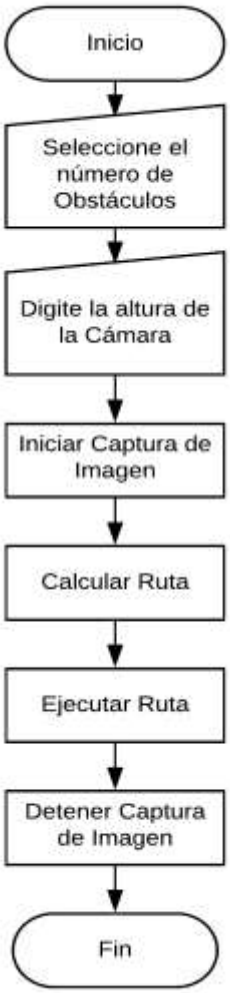
Figura 75. Interfaz de usuario del programa.



Fuente: Autor.

El diagrama de flujo de la interfaz se puede observar en la Figura 76, se aprecia que primero se debe seleccionar el número de obstáculos, seguido de la digitación de la altura a la cual se encuentra la cámara, luego se procede a adquirir la imagen, para luego calcular la ruta y de esta manera poder ejecutarla, por ultimo detener la adquisición de imágenes para finalizar el programa, en el ANEXO I se encuentra el código realizado para el desarrollo de la interfaz grafica de la aplicación.

Figura 76. Diagrama de flujo Interfaz de Usuario.



Fuente: Autor.

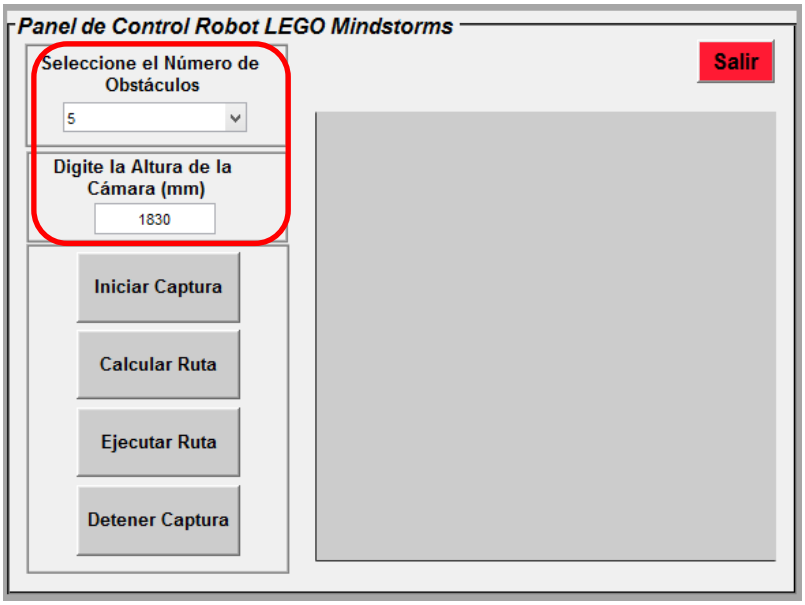
A continuación, se mostrará el funcionamiento de cada objeto de la interfaz de usuario.

6.4. FUNCIONAMIENTO DE LOS OBJETOS DE LA INTERFAZ DE USUARIO

1. Prueba con 5 Obstáculos.

Lo primero que se hace en el software es seleccionar la cantidad de obstáculos que van a intervenir en la trayectoria, en la Figura 77 se puede observar el programa con la cantidad de obstáculos necesaria y la altura de la cámara usada en la prueba con un valor de 1.830 mm.

Figura 77. Interfaz de usuario con características necesarias.



Fuente: Autor.

En la Figura 78 se puede observar la adquisición de imagen con el contador de obstáculos en funcionamiento, en la figura de la izquierda se puede observar el entorno de prueba, en la parte derecha se muestra el mismo entorno, solo que en esta ocasión se tienen 2 obstáculos de más, de manera que se puede observar como el programa entiende que solo debe tomar 5 obstáculos, así que los otros 2 los deja sin caracterización.

Figura 78. Programa con detección de obstáculos en ejecución.

Fuente: Autor.

Luego de adquirir la imagen se continua con el cálculo de la ruta, para que esto suceda en este botón intervienen las funciones y algoritmos generados, esto para realizar el cálculo correctamente. Luego de obtener la información de los objetos se procede a generar el mapeo y la ubicación de estos, en la Figura 79 se puede observar la matriz con ampliación de obstáculos que genera Matlab, en ella se puede ver las características de cada uno, siendo -1 la ubicación de los obstáculos, el 1 y 3 la ubicación del robot, el 0 para la ubicación del objetivo y el 2 para mostrar las zonas con espacio para que el robot pueda moverse.

Figura 79. Matriz con ampliación de obstáculos en ejecución.

```
mapa: 9x12 double =

     2     2     2     2    -1    -1    -1     2     2     2     2     2
     2     2     2     2    -1    -1    -1     2     2     2     2     2
     2     2     2     2    -1    -1    -1    -1    -1    -1    -1     2
     2     2     2     0     2    -1    -1    -1    -1    -1    -1     2
     2     2     2     2    -1    -1    -1    -1    -1    -1    -1     2
     2     2    -1    -1    -1    -1    -1     2     2     2     2     2
     2     2    -1    -1    -1    -1    -1     2     2     2     2     2
     2     2    -1    -1    -1     2     2     2     2     1     2     2
     2     2     2     2     2     2     2     2     2     3     2     2
```

Fuente: Autor.

Luego de generar la matriz con obstáculos el programa continua ejecutando el algoritmo A*, este selecciona la ruta optima, en la Figura 80 se puede observar el vector generado por el algoritmo, este tiene la ubicación de cada casilla por la que el robot va a realizar la trayectoria óptima.

Figura 80. Vector con las ubicaciones de las casillas de la trayectoria óptima.

```
Optimal_path: 11x2 double

     5     3
     6     2
     7     2
     8     2
     9     3
     9     4
     9     5
     9     6
     9     7
     9     8
     9     9
```

Fuente: Autor.

En la Figura 81 se puede apreciar la matriz con la trayectoria calculada, esta con una característica especial ya que se identifica con el número 10, también se puede verificar las casillas con el vector generado en la Figura 80.

Figura 81. Matriz con trayectoria calculada.

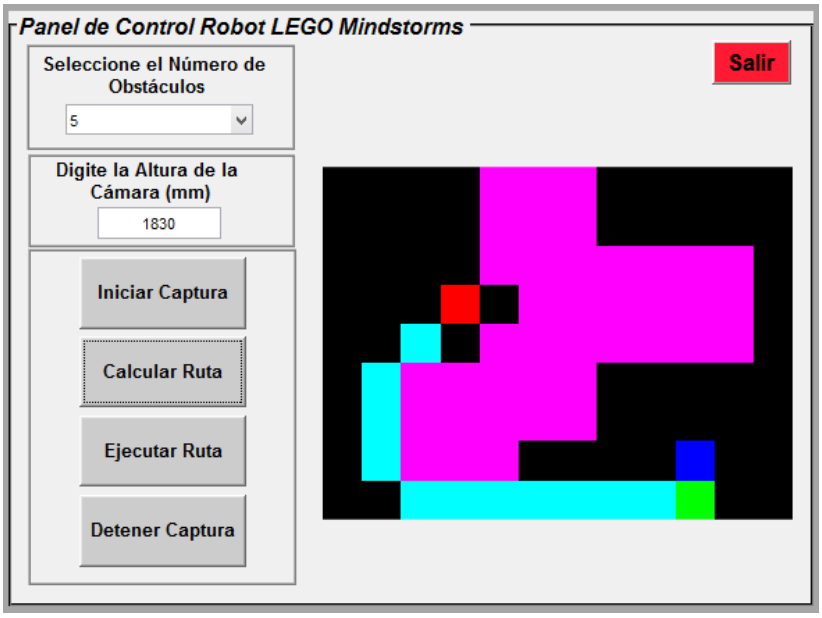
```
map: 9x12 double =

     2     2     2     2    -1    -1    -1     2     2     2     2     2
     2     2     2     2    -1    -1    -1     2     2     2     2     2
     2     2     2     2    -1    -1    -1    -1    -1    -1    -1     2
     2     2     2     0     2    -1    -1    -1    -1    -1    -1     2
     2     2    10     2    -1    -1    -1    -1    -1    -1    -1     2
     2    10    -1    -1    -1    -1    -1     2     2     2     2     2
     2    10    -1    -1    -1    -1    -1     2     2     2     2     2
     2    10    -1    -1    -1     2     2     2     2     1     2     2
     2     2    10    10    10    10    10    10    10     3     2     2
```

Fuente: Autor.

En la Figura 82 se puede observar la trayectoria calculada, generada en la zona cuatro de la interfaz de usuario, esta vez es mostrada de manera más explícita con un diagrama en donde se observa la trayectoria en la que se va a desplazar el robot.

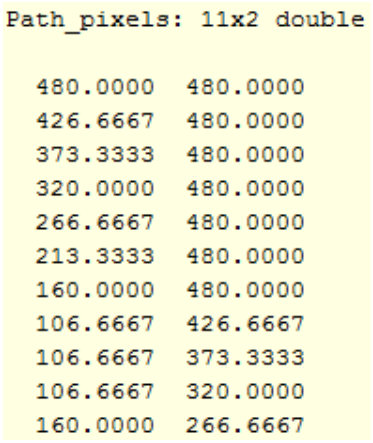
Figura 82. Interfaz de usuario con diagrama de trayectoria calculada.



Fuente: Autor.

Luego de calculada la trayectoria se ejecuta la ruta, para poder ejecutarse se realizan las conversiones necesarias, ya que la información que se tiene no puede ser enviada al robot ya que no será entendida, por ejemplo en la Figura 83 se puede apreciar la conversión realizada a los valores de los vectores de la Figura 80, en donde pasaron de ser una simple casilla, a tener un valor en pixeles y ubicación en el entorno.

Figura 83. Conversión de casillas a pixeles.



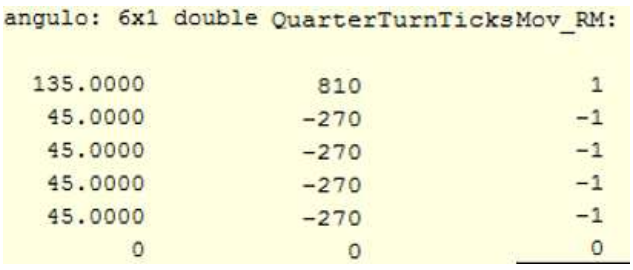
Path_pixels: 11x2 double

480.0000	480.0000
426.6667	480.0000
373.3333	480.0000
320.0000	480.0000
266.6667	480.0000
213.3333	480.0000
160.0000	480.0000
106.6667	426.6667
106.6667	373.3333
106.6667	320.0000
160.0000	266.6667

Fuente: Autor.

Lo siguiente a realizar es la conversión de ángulos y distancias a comandos que el robot pueda entender, en la Figura 84 se puede apreciar la figura que consta de 3 partes, la primer parte que se encuentra a la izquierda se ven los valores de los ángulos en grados, en la segunda parte se observa el valor de los ángulos en Ticks del motor, mientras que en la parte final se observa cual debe ser el motor que debe encender para realizar el giro, de manera que si el valor es 0 enciendes los 2 motores al tiempo, si es -1 solo enciende el motor izquierdo y si es 1 solo enciende el motor derecho. Esta es la información que será enviada al motor para que ejecute la trayectoria.

Figura 84. Conversión de ángulos a movimientos de motores



angulo: 6x1 double QuarterTurnTicksMov_RM:

135.0000	810	1
45.0000	-270	-1
45.0000	-270	-1
45.0000	-270	-1
45.0000	-270	-1
0	0	0

Fuente: Autor.

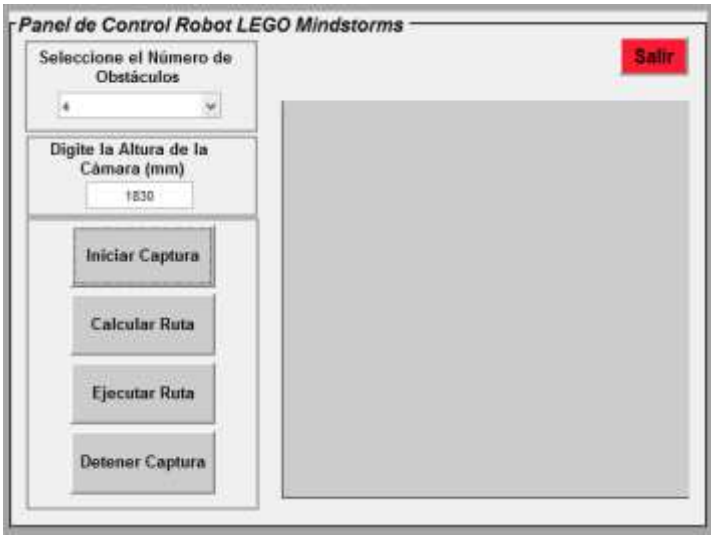
En la Figura 85 se observa la ubicación final del robot, cuando el finaliza su trayectoria encima del objetivo, de manera que el robot realizo su trayectoria correctamente, el programa aun continua en ejecución y para terminar completamente se oprime el botón de detención de captura y se cierra el programa para finalizar, la ejecución de esta prueba con 5 obstáculos se realizó con un tiempo de 1 minuto con 5 segundos en total, esto es mientras se realiza la toma de imagen, el cálculo de la trayectoria, la conexión del ordenador con el robot y por ultimo la ejecución de la trayectoria.

Figura 85. Ubicación del Robot al finalizar la trayectoria.

Fuente: Autor.

2. Prueba con 4 Obstáculos.

Figura 86. Interfaz de usuario con 4 obstáculos y altura de 1830 mm.

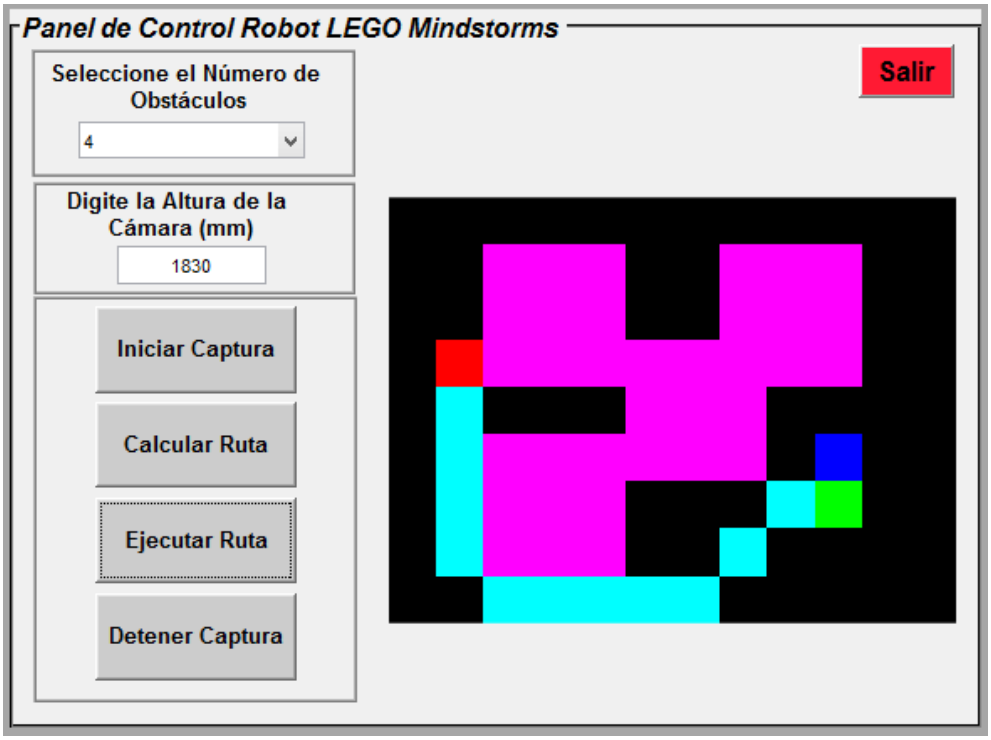


Fuente: Autor.

Figura 87. Programa con adquisición de imágenes en ejecución.

Fuente: Autor.

Figura 88. Interfaz con diagrama de trayectoria calculada para 4 obstáculos.



Fuente: Autor.

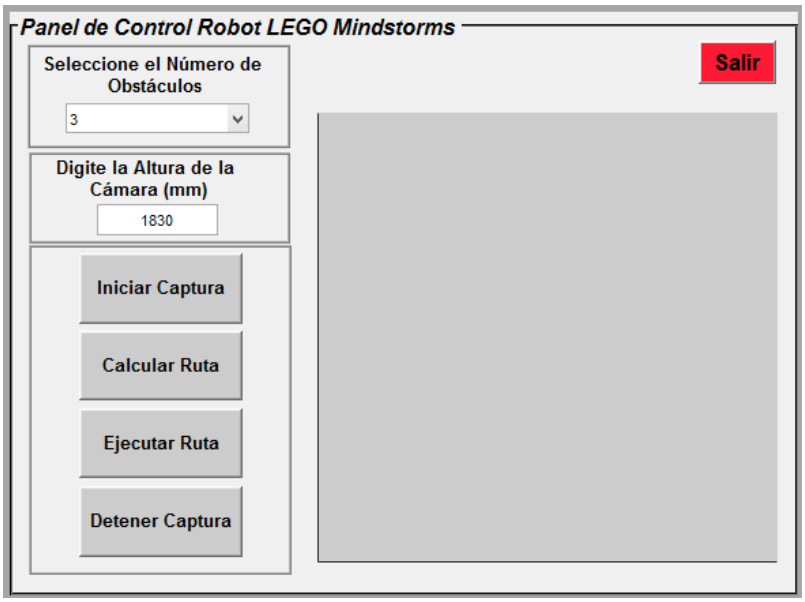
Figura 89. Posición del Robot al finalizar la trayectoria.

Fuente: Autor.

La ejecución de esta prueba con 4 obstáculos se realizó con un tiempo de 1 minuto con 15 segundos en total, esto es mientras se realiza la toma de imagen, el cálculo de la trayectoria, la conexión del ordenador con el robot y por ultimo la ejecución de la trayectoria.

3. Prueba con 3 Obstáculos.

Figura 90. Interfaz de usuario con 3 obstáculos y altura de 1830 mm.

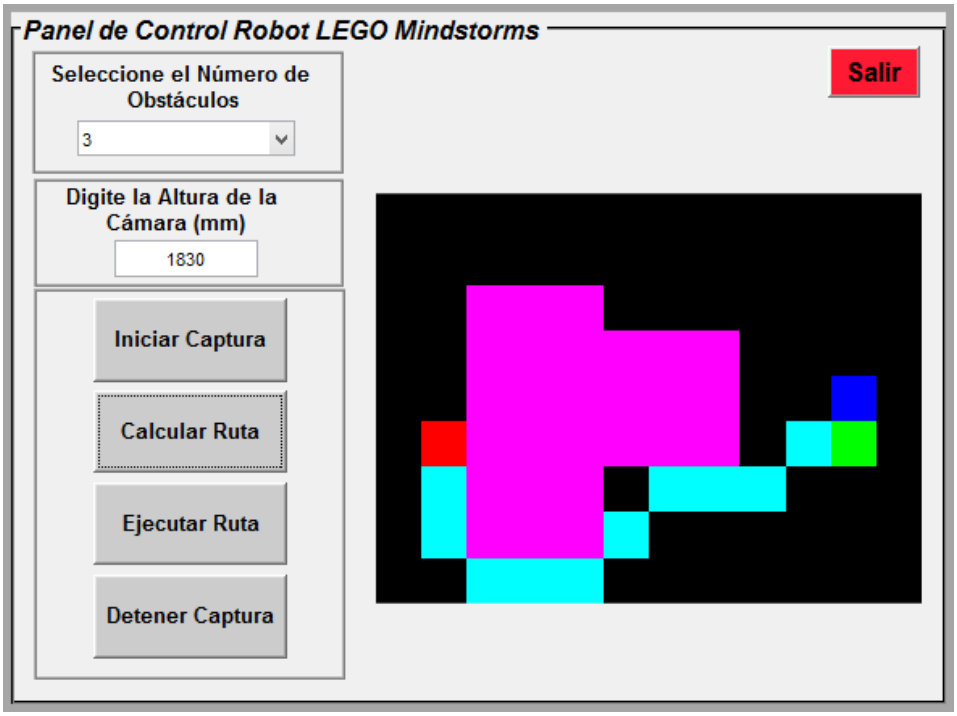


Fuente: Autor.

Figura 91. Programa con adquisición de imágenes en ejecución.

Fuente: Autor.

Figura 92. Interfaz con diagrama de trayectoria calculada para 3 obstáculos.



Fuente: Autor.

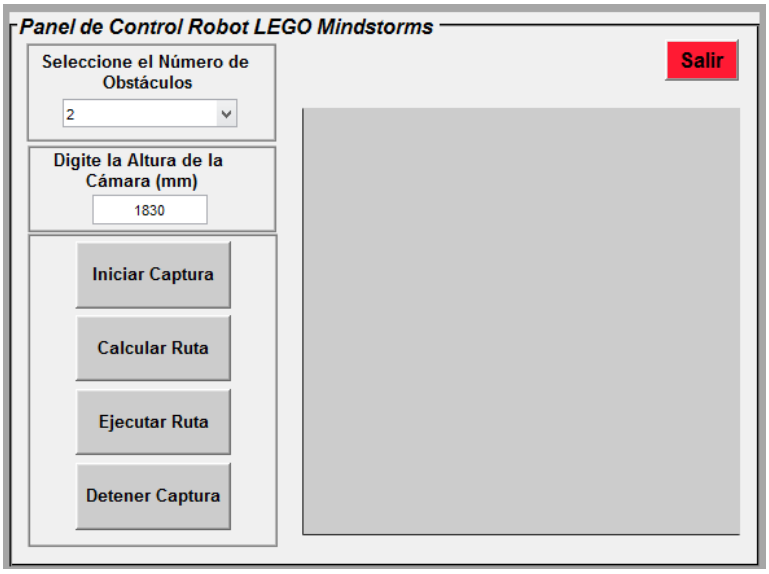
Figura 93. Posición del Robot al finalizar la trayectoria.

Fuente: Autor.

La ejecución de esta prueba con 3 obstáculos se realizó con un tiempo de 1 minuto con 30 segundos en total, esto es mientras se realiza la toma de imagen, el cálculo de la trayectoria, la conexión del ordenador con el robot y por ultimo la ejecución de la trayectoria.

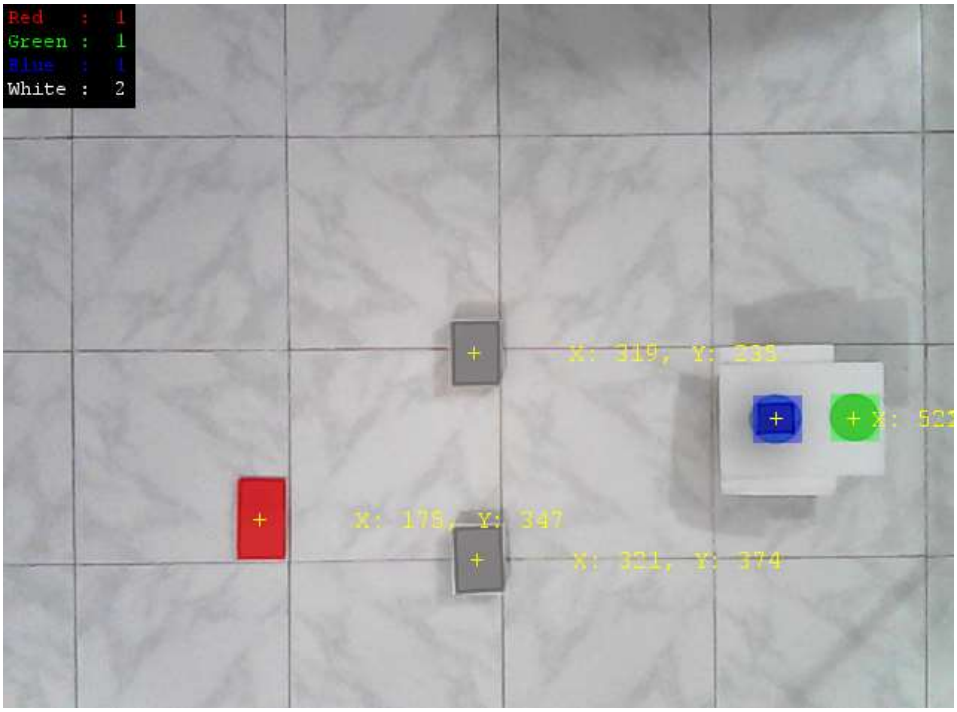
4. Prueba con 2 Obstáculos.

Figura 94. Interfaz de usuario con 2 obstáculos y altura de 1830 mm.



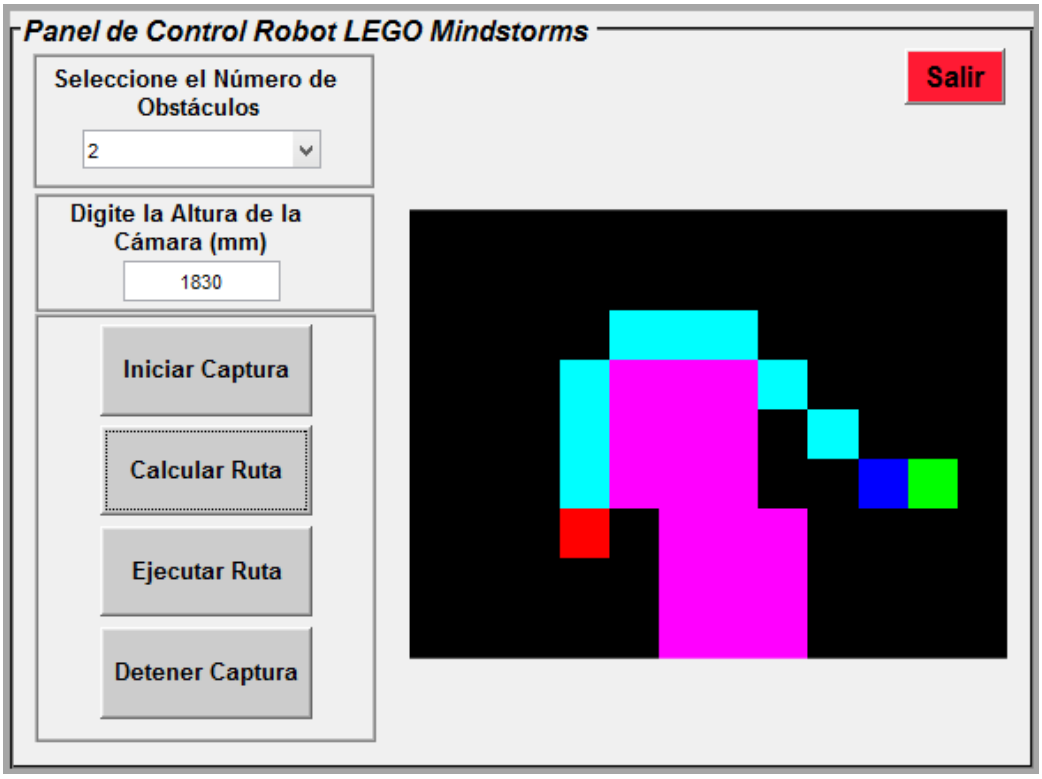
Fuente: Autor.

Figura 95. Programa con adquisición de imágenes en ejecución.



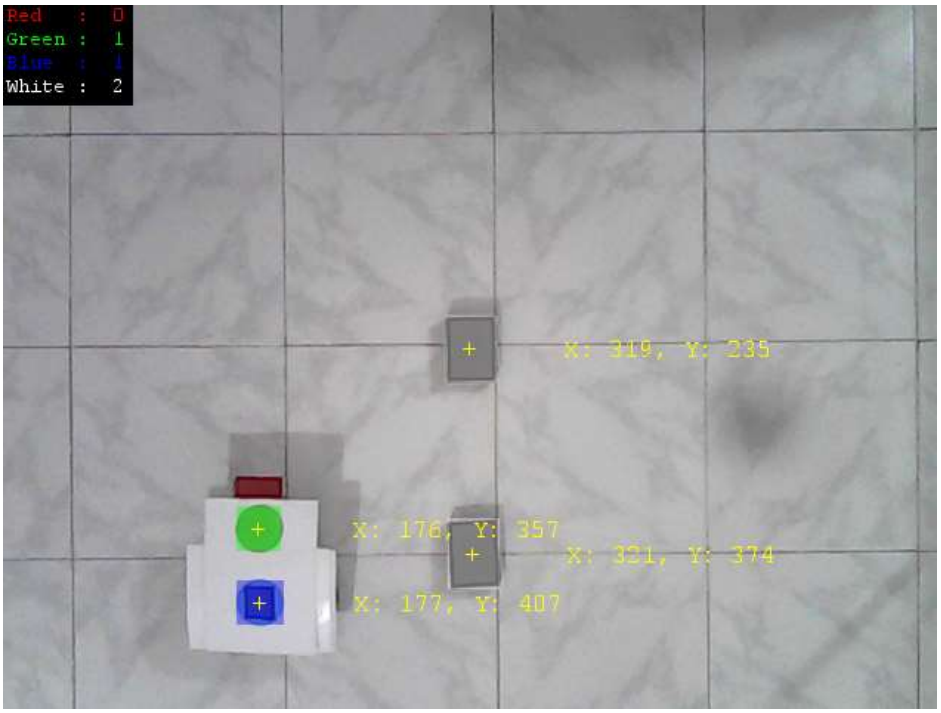
Fuente: Autor.

Figura 96. Interfaz con diagrama de trayectoria calculada para 2 obstáculos.



Fuente: Autor.

Figura 97. Posición del Robot al finalizar la trayectoria.



Fuente: Autor.

La ejecución de esta prueba con 2 obstáculos se realizó con un tiempo de 55 segundos segundos en total, esto es mientras se realiza la toma de imagen, el cálculo de la trayectoria, la conexión del ordenador con el robot y por ultimo la ejecución de la trayectoria.

Luego de realizadas las pruebas se encontró que el robot en algunas de ellas presenta una desviación en su trayectoria, esto se da en gran medida por la configuración diferencial implementada en el robot, ya que en este tipo de configuración es de gran importancia poder controlar la velocidad de cada rueda, ya que se pueden presentar variaciones de las velocidades debido a deslizamientos, fricción entre las ruedas y el terreno en el que se desarrolla el desplazamiento o perturbaciones en cada uno de los motores.

7. CONCLUSIONES Y RECOMENDACIONES

7.1. CONCLUSIONES

- Fue posible generar trayectorias haciendo uso del algoritmo desarrollado, para ambientes con las siguientes condiciones; entornos de tipo estructurado (por ejemplo, bodegas de almacenamiento) y áreas de trabajo interior (por ejemplo, un laboratorio de experimentos).

- Gracias a la implementación del algoritmo fue posible seleccionar la cantidad de obstáculos que se tienen en el terreno; asimismo, a través del uso de fotogrametría se pudo calcular la escala de la imagen, logrando determinar las medidas de objetos por medio de una conversión de pixeles a medidas reales en milímetros, teniendo en cuenta la altura y la distancia focal de la cámara. Lo anterior, fue muy importante para el desarrollo del software.

- El desarrollo de trayectorias se genero con base en el algoritmo de busqueda denominado A*, el cual encuentra la ruta de menor valor o costo, entre las casillas en las que se discretiza el entorno, teniendo de referencia el punto de partida y llegada del robot, obteniendo la ruta mas optima (más corta) que existe entre todas las posibles.

- Para el desarrollo de las funciones que activan los actuadores del robot, se tuvo en cuenta diferentes variables para obtener los desplazamientos requeridos. Entre estos se destaca la conversión de distancias, ángulos y movimientos, a giros del motor LEGO, garantizando que las instrucciones que se envían al robot, permiten ejecutar la trayectoria determinada por el algoritmo A*.

- A través de una cámara web se logró realizar la detección de obstáculos, la meta y el robot, logrando que el algoritmo configurara la malla de circulación en función de los pixeles de la cámara, generando un sistema coordenado, donde se ubican espacialmente los elementos, obteniendo los datos de entrada para la generación de la trayectoria en la pantalla de la GUIDE.

- La implementación de este software funciono de manera óptima con una conexión vía Bluetooth, este tipo de enlace con el robot presenta ventajas para el manejo de este, ya que no es necesario disponer de medios de comunicación físicos como cables, los cuales afectarían la maniobrabilidad del robot.

7.2. RECOMENDACIONES

- En futuras investigaciones se podrían realizar estudios usando terrenos de tipo no estructurado, en los cuales se introduzcan obstáculos móviles como por ejemplo una línea de producción, además de trabajar en ambientes exteriores con terrenos irregulares.

- En futuros proyectos se podría implementar una comunicación vía Wireless para entrega de datos más rápida, de manera que no existan tiempos de espera como en algunas ocasiones ocurre vía Bluetooth.

- Para futuras investigaciones se podrían implementar espacios mucho más amplios y complejos, de manera que seria posible implementar cámaras con mejores características como mayor y mejor resolución, distancia focal, además de poder ubicarlas en el robot o a mayor altura, con esto generar laberintos o ambientes mas complejos.

- Para futuras investigaciones se propone desarrollar un control de la trayectoria, de manera que trabaje en tiempo real mientras el robot ejecuta la ruta esto para evitar que se desvie en el transcurso de la ruta, enviando alguna señal a los motores o a la aplicación para que el robot corrija su trayectoria, esto debido a que en el proyecto en algunas pruebas se encontró que el robot se desviaba un poco de la trayectoria, esto puede ser causado por factores como: el terreno, la conexión, la batería del robot, entre otros.

8. REFERENCIAS

[1] M. Á. Fernández Lancha, D. Fernández Sanz, y C. Valmaseda Plasencia, “Planificacion De Trayectorias Para Un Robot Movil”, *147.96.80.209*, p. 125, 2010.

[2] “CAPÍTULO 1 PLANEACIÓN DE MOVIMIENTOS Una de las metas en robótica es la creación de”.

[3] J. Aracil, “Introducción a Matlab y Simulink Introducción Componentes de Matlab”, pp. 1–24, 2007.

[4] A. Behrens, L. Atorf, y T. Aach, “Teaching practical engineering for freshman students using the RWTH–Mindstorms NXT toolbox for MATLAB”, *Matlab-Modelling, Program. Simulations*, núm. June 2014, p. 41, 2010.

[5] F. Reyes, *Robótica. Control de robots manipuladores*, Primera Ed. México, 2011.

[6] A. Yandún, “Planeacion y seguimiento de trayectorias para un robot móvil”, 2011.

[7] “Semana de la Electrónica Facultad de Ciencias de la Electrónica BUAP Dr. Daniel Mocencahua Mora Alfonso M. Cuesta Hernández Métodos de planificación de. - ppt descargar”. [En línea]. Disponible en: https://slideplayer.es/slide/3519177/. [Consultado: 05-oct-2019].

[8] “Algoritmo de Dijkstra - Proyecto Inicio”. [En línea]. Disponible en: https://sites.google.com/site/grupo28upm/home/algoritmo-de-dijkstra. [Consultado: 10-ene-2020].

[9] B. Lopez Salazar, “Algoritmo de Dijkstra”, *12-junio-2019*, 2019. [En línea]. Disponible en: https://www.ingenieriaindustrialonline.com/investigacion-de-operaciones/algoritmo-de-dijkstra/. [Consultado: 08-nov-2019].

[10] O. Ortiz y A. Santana, “Planificación de caminos para Robots en Realidad Virtual”, 2017.

[11] “Performance Comparison of BUG Algorithms for Mobile Robots - DISTBUG Algorithm - YouTube”. [En línea]. Disponible en: https://www.youtube.com/watch?v=Sg0JiwCj5tU. [Consultado: 05-sep-2019].

[12] M. (Iowa S. U. Steven y J. (University of T. James, “Rapidly Exploring Random Trees-Progress and Prospects”, núm. 1, pp. 6–8, 2003.

[13] V. Jorge, “Algoritmo A* [código]”. [En línea]. Disponible en: http://jc-info.blogspot.com/2009/05/algoritmo-codigo.html. [Consultado: 20-may-2019].

[14] J. A. C. E. Víctor R. González Fdez, Antonio López Cruzado, “Robots móviles”, 2018. [En línea]. Disponible en: http://platea.pntic.mec.es/vgonzale/cyr_0708/index.htm.

[15] I. Bambino, “Una Introducción a los Robots Móviles”, 2008.

[16] “Cómo aumentar la productividad en sus líneas de producción”. [En línea]. Disponible en: https://ayhconsultoria.com/lineas-de-produccion/. [Consultado: 26-jun-2019].

[17] “La importancia de conocer los tipos de bodegas”. [En línea]. Disponible en:

http://clis.co/la-importancia-conocer-los-tipos-bodegas/. [Consultado: 15-mar-2020].

[18] “Mejores robot aspirador para hogares que tengan mascotas”. [En línea]. Disponible en: https://topesdegama.com/listas/gadgets/robot-aspirador-pelo-mascotas. [Consultado: 10-ago-2019].

[19] “TALON | ceta”. [En línea]. Disponible en: https://www.ceta.aero/copia-de-dragon-runner-dr20. [Consultado: 05-sep-2019].

[20] JUAN DAVID PRECIADO AGUILAR, “PLANIFICACIÓN DE TRAYECTORIAS DE ROBOTS MÓVILES DE DIFERENTES ARQUITECTURAS EN ENTORNOS DINÁMICOS.”, p. 105, 2018.

[21] “Una esfera que choca con un escalón”. [En línea]. Disponible en: http://www.sc.ehu.es/sbweb/fisica3/solido/escalon_esfera/escalon_esfera.html. [Consultado: 18-ago-2019].

[22] “Robot móvil controlado por bluetooth”. [En línea]. Disponible en: https://naylampmechatronics.com/blog/53_robot-movil-controlado-por-bluetooth.html. [Consultado: 13-ene-2020].

[23] “RB-CAR | Robotnik”. [En línea]. Disponible en: https://www.roscomponents.com/es/robots-moviles/19-rbcar.html#/equipo_de_comunicacion_inalambrico_de_larga_distancia-no/robotnik_gps-no/summit_xl_zona_rgbd-no/summit_xl_zone_2-no/summit_xl_zone_3-no. [Consultado: 23-ene-2020].

[24] Á. Soriano Vigueras y Á. V. Fernández, “Diseño y Programación de Algoritmos para Robot Móviles. Aplicación al robot LEGO-NXT”, *Univ. Politec. Val.*, p. 117.

[25] L. Á. Silva, “INFORMÁTICA INDUSTRIAL Escuela Técnica Superior de Ingenieros Industriales CONTROL VISUAL DE ROBOTS PARALELOS . Tesis Doctoral Autor José Maria Sebastián y Zúñiga Doctor Ingeniero Industrial”, núm. April, 2014.

[26] “Robot de Inspección iPEK ROVION”. [En línea]. Disponible en: http://www.panatec-agua.com/robot-inspeccion-ipek.php. [Consultado: 15-jul-2019].

[27] “Que debería saber antes de elegir un sistema de visión artificial (PARTE I) - Blog de BcnvisionBlog de Bcnvision”. [En línea]. Disponible en: https://www.bcnvision.es/blog-vision-artificial/algunas-preguntas-que-deberia-hacerse-antes-de-comprar-un-sistema-de-vision-artificial-parte-i/. [Consultado: 30-sep-2019].

[28] M. E. Aircraft y A. M. Marcolini, “Escuela técnica superior de ingenieros industriales”, núm. June, p. 106, 2014.

[29] M. Baquero-su, “Generación de Trayectorias Y Procesamiento Digital de Imágenes para la Navegación de un Robot Móvil”, núm. September 2007, 2016.

[30] “Cámara Web Genius Facecam 1000 Webcam”. [En línea]. Disponible en: https://www.linio.com.co/p/ca-mara-web-genius-facecam-1000x-webcam-hd-720p-chat-skype-qbiqun. [Consultado: 20-sep-2019].

[31] I. The MathWorks, “Filtrado de mediana 2D - MATLAB medfilt2 - MathWorks

América Latina". [En línea]. Disponible en: https://la.mathworks.com/help/images/ref/medfilt2.html. [Consultado: 28-oct-2019].

[32] I. The MathWorks, "Restar una imagen de otra o restar constante de la imagen - MATLAB imsubtract - MathWorks América Latina". [En línea]. Disponible en: https://la.mathworks.com/help/images/ref/imsubtract.html. [Consultado: 28-may-2020].

[33] I. The MathWorks, "Binarize 2-D grayscale image or 3-D volume by thresholding - MATLAB imbinarize - MathWorks". [En línea]. Disponible en: https://la.mathworks.com/help/images/ref/imbinarize.html?lang=en. [Consultado: 28-may-2020].

[34] L. Atorf, "PC - NXT Communication", *7 de julio*, 2008. [En línea]. Disponible en: https://www.mindstorms.rwth-aachen.de/documents/downloads/doc/version-4.07/pc_nxt_communication.html#1. [Consultado: 15-sep-2019].

[35] P. F. De Carrera, "Desarrollo multiplataforma de aplicaciones de control y comunicación para robots móviles", 2012.

[36] R. Orellana, "Apuntes de Fotogrametría", *System*, pp. 1–48, 2006.

[37] S. Pablo y C. Posada, "Fotogrametría básica: cálculo de la escala y el paralaje", 2013. .

[38] "Distancia focal | Curso de fotografía digital TheWebfoto". [En línea]. Disponible en: http://www.thewebfoto.com/2-hacer-fotos/203-distancia-focal. [Consultado: 09-oct-2019].

9. ANEXOS

9.1. ANEXO A

CÓDIGO BOTÓN CAPTURAR IMAGEN

---------- EJECUTA PRESIONANDO EL BOTÓN CAPTURAR ----------

```
function Capturar_Callback (hObject, eventdata, handles)
texto=get (handles. edit1,'String');
Dist_cam=str2num(texto); %Escritura de distancia de Cámara
X=strcat(handles.c) %Selección de Cantidad de obstáculos
O=double(X);
redThresh = 0.2; % Cálculo de umbrales para la binarización del mapa de grises
greenThresh = 0.05; %
blueThresh = 0.15; %
whiteThresh = 0.4;

global ResC_x;
global ResC_y;
global Dist_f;
global Z_cam;
global Fconv;
global apagar;
global Inicio;
global vidDevice;
global hVideoIn;
global vidIn;

%Variables de Cámara
ResC_x=640;
ResC_y=480;
Dwind_x=161; %Tamaño X de la ventana de la interfaz en mm
Dwind_y=121; %Tamaño Y de la ventana de la interfaz en mm
Dist_f=200.9840988; %Distancia focal
Z_cam=Dist_cam; %Altura de cámara
Esc=Z_cam/Dist_f; %Escala de fotografía
Dist_x=Dwind_x*Esc; %Distancia aplicando factor de conversión
Dist_y=Dwind_y*Esc; %Distancia aplicando factor de conversión
Fconv= mean ([Dist_x/ResC_x, Dist_y/ResC_y]); %Cálculo de factor de conversión
según altura
Inicio=1;

while(Inicio)

   vidDevice = imaq. VideoDevice ('winvideo', 3, 'YUY2_640x480', 'ROI', [1 1 ResC_x
ResC_y], 'ReturnedColorSpace', 'rgb');
   frame = step(vidDevice);
   vidInfo = imaqhwinfo(vidDevice);
```

```
   hblob = vision. BlobAnalysis ('AreaOutputPort', false, 'CentroidOutputPort', true, 'BoundingBoxOutputPort', true', 'MinimumBlobArea', 100, 'MaximumBlobArea', 3000, 'MaximumCount', 1); % Configuración del reconocimiento de objetos para colores
   hblobWhite = vision. BlobAnalysis ('AreaOutputPort', false, 'CentroidOutputPort', true, 'BoundingBoxOutputPort', true', 'MinimumBlobArea', 300, 'MaximumBlobArea', 3000, 'MaximumCount', O); % Configuración del reconocimiento de objetos para negro
   hshapeinsBox = vision. ShapeInserter ('BorderColorSource', 'Input port', 'Fill', true, 'FillColorSource', 'Input port', 'Opacity', 0.4); %para el texto en la ventana de video
   htextinsRed = vision. TextInserter ('Text', 'Red: %2d', 'Location', [5 2], 'Color', [1 0 0], 'Font', 'Courier New', 'FontSize', 14);
   htextinsGreen = vision. TextInserter ('Text', 'Green: %2d', 'Location', [5 18], 'Color', [0 1 0], 'Font', 'Courier New', 'FontSize', 14);
   htextinsBlue = vision. TextInserter ('Text', 'Blue: %2d', 'Location', [5 34], 'Color', [0 0 1], 'Font', 'Courier New', 'FontSize', 14);
   htextinsWhite = vision. TextInserter ('Text', 'White: %2d', 'Location', [5 50], 'Color', [1 1 1], 'Font', 'Courier New', 'FontSize', 14);

   htextinsCent = vision. TextInserter ('Text', '+      X: %4d, Y: %4d', ... % para el centroide al lado de cada figura
       'LocationSource', 'Input port', ...
       'Color', [1 1 0], ... // yellow color
       'Font', 'Courier New', ...
       'FontSize', 17);
   hVideoIn = vision. VideoPlayer ('Name', 'Final Video', 'Position', [100 100 vidInfo.MaxWidth+20 vidInfo.MaxHeight+30]); % Salida de Video

   %% filtrado de colores
   global ejecutar;
   global centXRed;
   global centYRed;
   global centXGreen;
   global centYGreen;
   global centXBlue;
   global centYBlue;
   global centXWhite;
   global centYWhite;

   ejecutar=1;
   apagar=1;

   while(ejecutar)
      rgbFrame = step(vidDevice); % adquirir el frame

      diffFrameRed = imsubtract (rgbFrame (:1), rgb2gray(rgbFrame)); % toma el componente rojo
      diffFrameRed = medfilt2(diffFrameRed, [3 3]); % filtra
      binFrameRed = imbinarize (diffFrameRed, redThresh); % blanco y negro

      diffFrameGreen = imsubtract (rgbFrame (:2), rgb2gray(rgbFrame));
      diffFrameGreen = medfilt2(diffFrameGreen, [3 3]);
      binFrameGreen = imbinarize (diffFrameGreen, greenThresh);
```

```
        diffFrameBlue = imsubtract (rgbFrame (:3), rgb2gray(rgbFrame));
        diffFrameBlue = medfilt2(diffFrameBlue, [3 3]);
        binFrameBlue = imbinarize (diffFrameBlue, blueThresh);

        bwredFrame = imbinarize (rgbFrame (:1), whiteThresh); %obtiene blanco y negro
para el umbral que mejor funcionó
        bwgreenFrame = imbinarize (rgbFrame (:2), whiteThresh);
        bwblueFrame = imbinarize (rgbFrame (:3), whiteThresh);

        binFrameWhite = bwredFrame | bwgreenFrame | bwblueFrame; %las combino
        binFrameWhite = medfilt2(binFrameWhite, [3 3]);
        binFrameWhite=imcomplement(binFrameWhite); %intercambia blanco con negro

        [centroidRed, bboxRed] = step (hblob, binFrameRed); % se obtienen los
centroides para el rojo y la caja que lo encierra con vision. BlobAnalysis
        centroidRed = uint16(centroidRed); % se convierte a entero

        [centroidGreen, bboxGreen] = step (hblob, binFrameGreen);
        centroidGreen = uint16(centroidGreen);

        [centroidBlue, bboxBlue] = step (hblob, binFrameBlue);
        centroidBlue = uint16(centroidBlue);

        [centroidWhite, bboxWhite] = step (hblobWhite, binFrameWhite);
        centroidWhite = uint16(centroidWhite);

        %Para escribir en la imagen los centroides y dibujar los rectangulos de cada color

        rgbFrame (1:70,1:90, :) = 0;
        vidIn = step (hshapeinsBox, rgbFrame, bboxRed, single ([1 0 0])); % agrega
cuadro rojo
        vidIn = step (hshapeinsBox, vidIn, bboxGreen, single ([0 1 0])); % agrega cuadro
verde
        vidIn = step (hshapeinsBox, vidIn, bboxBlue, single ([0 0 1])); % agrega cuadro
azul
        vidIn = step (hshapeinsBox, vidIn, bboxWhite, single ([1 1 1])); % agrega cuadro
blanco

        for object = 1:1: length (bboxRed (:1)) % escribe el centroide para el elemento
rojo
            centXRed = centroidRed(object,1);
            centYRed = centroidRed(object,2);

            vidIn = step (htextinsCent, vidIn, [centXRed centYRed], [centXRed-6 centYRed-
9]);
        end

        for object = 1:1: length (bboxGreen (:1)) %
            centXGreen = centroidGreen(object,1);
            centYGreen = centroidGreen(object,2);
```

```
        vidIn = step (htextinsCent, vidIn, [centXGreen centYGreen], [centXGreen-6
centYGreen-9]);
    end

    for object = 1:1: length (bboxBlue (:1)) %
        centXBlue = centroidBlue(object,1);
        centYBlue = centroidBlue(object,2);

        vidIn = step (htextinsCent, vidIn, [centXBlue centYBlue], [centXBlue-6
centYBlue-9]);
    end

    centXWhite= [];
    centXWhite=uint16(centXWhite);
    centYWhite= [];
    centYWhite=uint16(centYWhite);

    for object = 1:1: length (bboxWhite (:1))
        centXWhite(object) = centroidWhite(object,1);
        centYWhite(object) = centroidWhite(object,2);
        vidIn = step (htextinsCent, vidIn, [centXWhite(object) centYWhite(object)],
[centXWhite(object)-6 centYWhite(object)-9]);
    end

    vidIn = step (htextinsRed, vidIn, uint8(length (bboxRed (:1)))); % Contador de
objetos Rojos
    vidIn = step (htextinsGreen, vidIn, uint8(length (bboxGreen (:1)))); % Contador
de objetos Verdes
    vidIn = step (htextinsBlue, vidIn, uint8(length (bboxBlue (:1)))); % Contador de
objetos Azules
    vidIn = step (htextinsWhite, vidIn, uint8(length (bboxWhite (:1)))); % Contador
de objetos Negros
    step (hVideoIn, vidIn); % Salida de video

    if apagar == 0
        break
    end
  end

  if Inicio == 0
    release(vidDevice);
    release(hVideoIn);
    close ('Final Video');
    break
  end
end
guidata (hObject, handles);
```

9.2. ANEXO B

CÓDIGO BOTÓN CALCULAR RUTA

----------SE EJECUTA PRESIONANDO EL BOTÓN CALCULAR ----------

```
function Calcular_Callback (hObject, eventdata, handles)

global centXRed; % objetivo
global centYRed;
global centXGreen; % robot
global centYGreen;
global centXBlue; % robot
global centYBlue;
global centXWhite; %Obstáculos
global centYWhite;
global mapa;
global ruta;
global ResC_x;
global ResC_y;
global Optimal_path;
global lim_x;
global lim_y;
global Target;
global Rob_back;
global Rob_front;

%% Tamaño de los obstáculos y cuadricula del mapa

disp(length(centXWhite'))
eje_y = 9;
eje_x=floor (((640/480) *eje_y));
lim_y=ResC_y/eje_y;
lim_x=ResC_x/eje_x;
mapa=2*ones (eje_y, eje_x);

for i=1: length(centXWhite')
   disp(i)
   NewRes_y=ceil(double(centYWhite(i)) /lim_y);
   NewRes_x=ceil(double(centXWhite(i)) /lim_x);
   mapa (NewRes_y, NewRes_x) =-1; %Resolución/10

   if NewRes_y<eje_y
      mapa (NewRes_y+1, NewRes_x) =-1;
   end

   if NewRes_y>1
      mapa (NewRes_y-1, NewRes_x) =-1;
   end

   if NewRes_x<eje_x
```

```
        mapa (NewRes_y, NewRes_x+1) =-1;
    end

    if NewRes_x>1
        mapa (NewRes_y, NewRes_x-1) =-1;
    end

    if NewRes_y<eje_y & NewRes_x<eje_x
        mapa (NewRes_y+1, NewRes_x+1) =-1;
    end

    if NewRes_y<eje_y & NewRes_x>1
        mapa (NewRes_y+1, NewRes_x-1) =-1;
    end

    if NewRes_y>1 & NewRes_x<eje_x
        mapa (NewRes_y-1, NewRes_x+1) =-1;
    end

    if NewRes_y>1 & NewRes_x>1
        mapa (NewRes_y-1, NewRes_x-1) =-1;
    end

end

mapa(ceil(double(centYRed)/lim_y), ceil(double(centXRed)/lim_x)) =0;
mapa(ceil(double(centYBlue)/lim_y), ceil(double(centXBlue)/lim_x)) =1;
mapa(ceil(double(centYGreen)/lim_y), ceil(double(centXGreen)/lim_x)) =3;
map=mapa;
```

---------- SE EJECUTA FUNCIÓN A STAR* ----------

```
 [xTarget, yTarget]=find(map==0);
Target= [xTarget, yTarget];
[xvalb, yvalb]=find(map==3);
Rob_back= [xvalb, yvalb];
[xval, yval]=find(map==1);
Rob_front= [xval, yval];

[ Optimal_path, bandera] = astar (map, eje_x, eje_y);

%% Cálculo de trayectoria
ruta = Optimal_path;
if(bandera)
    ruta2=ruta; %Para el dibujo no es necesario dibujar el último paso por eso se hace
necesario borrar ese último cuadro
    ruta2(1, :)= []; %
    for i=1: size(ruta2)
        map(ruta2(i,1), ruta2(i,2)) =10; % coloca 10 en las casillas correspondientes a la
ruta
    end
```

```
end
Optimal_path=ruta2;

%% Graficar ruta
r=map;
g=map;
b=map;
e=map;

r(r==0|r==-1) =255;
r(r~=255) =0;
g(g==-1|g==10) =255;
g(g~=255) =0;
b(b==1|b==-1|b==10) =255;
b(b~=255) =0;
e(e==10|e==3) =255;
e(e~=255) =0;

col (:1) =r;
col (:2) =g;
col (:3) =b;
col (:2) =e;

imshow(col,'InitialMagnification','Fit');
```

9.3. ANEXO C

CÓDIGO FUNCIÓN A STAR*

```
---------------------- SE EJECUTA FUNCIÓN A STAR*-----------------------

function [ Optimal_path, bandera] = astar (map, eje_x, eje_y)
MAX_X=eje_y;
MAX_Y=eje_x;
MAX_VAL=eje_x;
bandera=1;

   %DEFINE THE 2-D MAP ARRAY
   [xTarget, yTarget]=find(map==0);
   [xval, yval]=find(map==1);
   [xvalb, yvalb]=find(map==3);

   xStart=xval; %Starting Position
   yStart=yval; %Starting Position

OPEN= [];
% Estructura de la lista cerrada

%X val | Y val |

% CLOSED=zeros(MAX_VAL,2);
CLOSED= [];

 %% Con esto se colocan los obstáculos en la lista cerrada

 k=1; % Contador Dummy
for i=1: MAX_X
   for j=1: MAX_Y
      if (map (i, j) == -1)
         CLOSED(k,1) =i;
         CLOSED(k,2) =j;
         k=k+1;
      end
   end
end
CLOSED_COUNT=size(CLOSED,1);
% Coloca el primer nodo como el inicial
xNode=xval;
yNode=yval;
OPEN_COUNT=1;
path_cost=0;
goal_distance=distance (xNode, yNode, xTarget, yTarget);
OPEN (OPEN_COUNT, :)=insert_open (xNode, yNode, xNode, yNode, path_cost,
goal_distance, goal_distance);
```

```
OPEN(OPEN_COUNT,1) =0;
CLOSED_COUNT=CLOSED_COUNT+1;
CLOSED(CLOSED_COUNT,1) =xNode;
CLOSED(CLOSED_COUNT,2) =yNode;
NoPath=1;

%% Inicio del algoritmo

while ((xNode ~= xTarget || yNode ~= yTarget) && NoPath == 1)
% plot (xNode+.5, yNode+.5,'go');
 exp_array=expand_array (xNode, yNode, path_cost, xTarget, yTarget, CLOSED,
MAX_X, MAX_Y);
 exp_count=size(exp_array,1);
 %UPDATE LIST OPEN WITH THE SUCCESSOR NODES
 %OPEN LIST FORMAT
 %-----------------------------------------------------------------------
 %IS ON LIST 1/0 |X val |Y val |Parent X val |Parent Y val |h(n) |g(n)|f(n)|
 %-----------------------------------------------------------------------
 %EXPANDED ARRAY FORMAT
 %-------------------------------
 %|X val |Y val ||h(n) |g(n)|f(n)|
 %-------------------------------
 for i=1: exp_count
   flag=0;
   for j=1: OPEN_COUNT
      if(exp_array(i,1) == OPEN(j,2) && exp_array(i,2) == OPEN(j,3))
         OPEN(j,8) =min(OPEN(j,8), exp_array(i,5)); %#ok<*SAGROW>
         if OPEN(j,8) == exp_array(i,5)
            %UPDATE PARENTS, gn, hn
            OPEN(j,4) =xNode;
            OPEN(j,5) =yNode;
            OPEN(j,6) =exp_array(i,3);
            OPEN(j,7) =exp_array(i,4);
         end %End of mínimum fn check
         flag=1;
      end%End of node check
%        if flag == 1
%           break;
   end%End of j for
   if flag == 0
      OPEN_COUNT = OPEN_COUNT+1;
      OPEN (OPEN_COUNT, :)=insert_open(exp_array(i,1), exp_array(i,2), xNode,
yNode, exp_array(i,3), exp_array(i,4), exp_array(i,5));
   end%End of insert new element into the OPEN list
 end%End of i for

 %END OF WHILE LOOP

 %% Aquí se encuentra el camino con el menor costo (f(n))
```

```
  index_min_node = min_fn (OPEN, OPEN_COUNT, xTarget, yTarget);
  if (index_min_node ~= -1)
   % Selecciona la posición de los nodos con menor costo
   xNode=OPEN(index_min_node,2);
   yNode=OPEN(index_min_node,3);
   % Carga o suma el costo al nodo padre
   path_cost=OPEN(index_min_node,6);
  % Mueve el nodo a la lista cerrada para saber que ya uso ese nodo
  CLOSED_COUNT=CLOSED_COUNT+1;
  CLOSED(CLOSED_COUNT,1) =xNode;
  CLOSED(CLOSED_COUNT,2) =yNode;
  OPEN(index_min_node,1) =0;
  else
     %No Path exists to the Target!!
     NoPath=0; %Exits the loop!
  end%End of index_min_node check
end%End of While Loop

%%
%%Once algorithm has run The optimal path is generated by starting of at the
%last node (if it is the target node) and then identifying its parent node
%until it reaches the start node. This is the optimal path

i=size(CLOSED,1);
Optimal_path= [];
xval=CLOSED(i,1);
yval=CLOSED(i,2);
i=1;
Optimal_path(i,1) =xval;
Optimal_path(i,2) =yval;
i=i+1;

if ((xval == xTarget) && (yval == yTarget))
    inode=0;
   % Recorre los nodos disponibles para determinar los nodos parientes
   parent_x=OPEN (node_index (OPEN, xval, yval),4); %node_index returns the
index of the node
   parent_y=OPEN (node_index (OPEN, xval, yval),5);

   while (parent_x ~= xStart || parent_y ~= yStart)
         Optimal_path(i,1) = parent_x;
         Optimal_path(i,2) = parent_y;
         % Consigue los nodos parientes
         inode=node_index (OPEN, parent_x, parent_y);
         parent_x=OPEN(inode,4); %node_index returns the index of the node
         parent_y=OPEN(inode,5);
         i=i+1;
   end
 j=size(Optimal_path,1);
```

```
 %% Encuentra la ruta Óptima
j=j-1;

 % Hace el recorrido de la ruta óptima
 for i=j: -1:1
end

else
    bandera=0;
end
```

9.4. ANEXO D

CÓDIGO BOTÓN EJECUTAR RUTA

---------- SE EJECUTA PRESIONANDO EL BOTÓN EJECUTAR ----------

```
function Ejecutar_Callback (hObject, eventdata, handles)

greenThresh = 0.05; % Umbral para detección color verde
blueThresh = 0.15; % Umbral para la detección color azul

global ResC_x;
global ResC_y;
ResC_x=640;
ResC_y=480;

global izquierda;
global derecha;
global centXRed;
global centYRed;
global centXGreen;
global centYGreen;
global centXBlue;
global centYBlue;
global Target;
global Rob_back;
global Rob_front;
global umbralAng;
global umbralDis;
global Optimal_path;
global lim_x;
global lim_y;
global Fconv;

izquierda=NXTMotor('C');
derecha=NXTMotor('B');
umbralAng=15;
umbralDis=20;

Rob_back_pixels= [Rob_back (2) *lim_x, Rob_back (1) *lim_y];
Rob_front_pixels= [Rob_front (2) *lim_x, Rob_front (1) *lim_y];
Target_pixels= [Target (2) *lim_x, Target (1) *lim_y];
Path_pixels= flip ([Optimal_path (:2). * lim_x, Optimal_path (:1). *lim_y]);
```

```
---------- SE EJECUTA FUNCIÓN POLAR PATH----------

[ distancia, angulo, Mov_RM] = f_Polar_Path (Rob_back_pixels, Rob_front_pixels,
Target_pixels, Path_pixels, Fconv);
Num_paths=length(angulo); % Cantidad de trayectorias que va a realizar el robot
conv_angmot_mm=2.083; % Conversión °rot motor/mm
conv_angmot_ang=6.0; % Conversión °rot motor/sexagesimales
ang_rot_motor=round (distancia. *conv_angmot_mm); % En grados de rotaciones
de motor
QuarterTurnTicks=Mov_RM. *round (angulo. *conv_angmot_ang); % En grados de
motor

ctr_ang=-ones (Num_paths, 1);

for i=1: Num_paths
   if mod(i,2) ==0
      ctr_ang(i)=1;
   end
end

%% Comunicación de Trayectoria Calculada al Robot
COM_CloseNXT all
Ports = [MOTOR_B; MOTOR_C]; % Puertos del motor para rueda izquierda y derecha
DrivingSpeed     = 50; %Velocidad del Robot

% Abre conexión Bluetooth
h = COM_OpenNXT('bluetooth.ini');
COM_SetDefaultNXT(h);

% Inicia las características del motor
for i=1: Num_paths
   if i==1
      %Velocidad de giro del Robot
      if QuarterTurnTicks(i)==0
         TurningSpeed     = 40;
      else
         TurningSpeed     = 40*(QuarterTurnTicks(i)/abs(QuarterTurnTicks(i)));
      end
      mStraight                = NXTMotor(Ports);
      mStraight.SpeedRegulation   = false; % not for sync mode
      mStraight.Power             = DrivingSpeed;
      mStraight.TachoLimit        = ang_rot_motor(i);
      mStraight.ActionAtTachoLimit = 'Brake';
      mTurn1                   = NXTMotor (Ports (2)); % ports swapped because it's
nicer
      mTurn1.SpeedRegulation      = false; % we could use it if we wanted
      mTurn1.Power             = -abs(TurningSpeed);
      mTurn1.TachoLimit           = abs(QuarterTurnTicks(i));
```

```
        mStraight.ActionAtTachoLimit = 'Brake';
        mTurn2          = mTurn1;
        mTurn2.Port     = Ports (1); % ports swapped again...
        mTurn2.Power    = mTurn1.Power;
        % Inicializa los motores
       % Se envía este comando en caso de que aún esten girando los motores
        mStraight.Stop('off');

        if QuarterTurnTicks(i)~=0
           if TurningSpeed<0
              mTurn1.SendToNXT();
              mTurn1.WaitFor();

           elseif TurningSpeed>0
              mTurn2.SendToNXT();
              mTurn2.WaitFor();
           end
        end
        % Ejecuta movimiento
        mStraight.SendToNXT();
        % Espera que finalice el movimiento
        mStraight.WaitFor();
        % Ahora espera para realizar el giro
    else

        %Velocidad de giro del Robot
        if QuarterTurnTicks(i)==0
           TurningSpeed     = 40;
        else
           TurningSpeed     = 40*(QuarterTurnTicks(i)/abs(QuarterTurnTicks(i)));
        end
        mStraight                = NXTMotor(Ports);
        mStraight.SpeedRegulation   = false; % not for sync mode
        mStraight.Power          = DrivingSpeed;
        mStraight.TachoLimit       = ang_rot_motor(i);
        mStraight.ActionAtTachoLimit = 'Brake';
        mTurn1                   = NXTMotor (Ports (2)); % ports swapped because it's
nicer
        mTurn1.SpeedRegulation      = false; % we could use it if we wanted
        mTurn1.Power             = ctr_ang(i)*abs(TurningSpeed);
        mTurn1.TachoLimit          = abs(QuarterTurnTicks(i));
        mStraight.ActionAtTachoLimit = 'Brake';
        mTurn2          = mTurn1;
        mTurn2.Port     = Ports (1); % ports swapped again...
        mTurn2.Power    = mTurn1.Power;
        % Inicializa los motores
       % Se envía este comando en caso de que aún esten girando los motores
        mStraight.Stop('off');
        if ctr_ang(i)==1
           if QuarterTurnTicks(i)~=0
```

```
            if TurningSpeed>0
                mTurn1.SendToNXT();
                mTurn1.WaitFor();

            elseif TurningSpeed<0
                mTurn2.SendToNXT();
                mTurn2.WaitFor();
            end
        end
    else
        if QuarterTurnTicks(i)~=0
            if TurningSpeed<0
                mTurn1.SendToNXT();
                mTurn1.WaitFor();

            elseif TurningSpeed>0
                mTurn2.SendToNXT();
                mTurn2.WaitFor();
            end
        end

    end
    % Ejecuta movimiento
    mStraight.SendToNXT();
    % Espera que finalice el movimiento
    mStraight.WaitFor();
    % Ahora espera para realizar el giro
  end
end

% Inicializa los motores
% Se envía este comando en caso de que aún esten girando los motores
mStraight.Stop('off');
% Se apagan los motores
mStraight.Stop('off');
% Se cierra la conexión Bluetooth
COM_CloseNXT(h);
```

9.5. ANEXO E

CÓDIGO FUNCIÓN POLAR PATH

---------- SE EJECUTA FUNCIÓN POLAR PATH----------

```
function [ Distances, Delta, Mov_RotMotor] = f_Polar_Path (Rob_back, Rob_front, Target, Path, Fconv)
Función Trayectoria en coordenadas Polares
La función calcula las magnitudes y ángulos de la trayectoria del robot

                INPUTS
================================================
Rob_back    =   Coordenadas del robot atrás (formato pixeles).
Rob_front   =   Coordenadas del robot adelante (formato pixeles).
Target      =   Coordenadas del punto objetivo (formato pixeles).
Path        =   Trayectoria calculada (formato pixeles).
Fconv       =   Factor de conversión de mm/pixeles

                OUTPUT
================================================
Distances   =   Distancia de cada punto en milímetros.
Delta       =   Ángulo de giro por cada punto en grados sexagesimales.

Autor: Santiago Morales

Num_move=length (Path (:1)) +3;
Dist_mesh= zeros (Num_move, 1);
Ang_mesh   = zeros (Num_move, 1);
Cuadrante = zeros (Num_move, 1);

%% Calcular Distancias, ángulos y Cuadrantes para cada pixel
for i=1: Num_move-1
switch i

   case 1
   Dx=Path(i,1)-Rob_front (1);
   Dy=-(Path(i,2)-Rob_front (2));
   Dist_mesh(i+1) =((Dx^2+Dy^2) ^0.5) *Fconv;
   Ang_mesh(i+1)   =atan(Dy/Dx) *180/pi;
   Cuadrante(i+1) = f_cuadrante (Dx, Dy);

   case Num_move-2
   Dx=Target(1)-Path(end,1);
   Dy=-(Target(2)-Path(end,2));
   Dist_mesh(i+1) =(Dx^2+Dy^2) ^0.5*Fconv;
   Ang_mesh(i+1)   =atan(Dy/Dx) *180/pi;
   Cuadrante(i+1) = f_cuadrante (Dx, Dy);

   case Num_move-1
   Dx=Rob_front (1)-Rob_back (1);
```

```
    Dy=-(Rob_front (2)-Rob_back (2));
    Dist_mesh(i+1) =((Dx^2+Dy^2) ^0.5) *Fconv;
    Ang_mesh(i+1)   =Ang_mesh(i);
    Cuadrante(i+1) = f_cuadrante (Dx, Dy);
    otherwise

    Dx=Path(i,1)-Path(i-1,1);
    Dy=-(Path(i,2)-Path(i-1,2));
    Dist_mesh(i+1) =((Dx^2+Dy^2) ^0.5) *Fconv;
    Ang_mesh(i+1) =atan(Dy/Dx) *180/pi;
    Cuadrante(i+1) = f_cuadrante (Dx, Dy);

End

end

%% Calcular la posición de giro inicial del robot
Ang_mesh (1) =Ang_mesh (2);
j=1;
Dx=Rob_front (1)-Rob_back (1);
Dy=-(Rob_front (2)-Rob_back (2));
Alfa (1) =Ang_mesh (1);
Alfa_rob=atan(Dy/Dx) *180/pi;
Cuadrante (1) = Cuadrante (2);

---------- SE EJECUTA FUNCIÓN CUADRANTES----------

Quad_rbt=f_cuadrante (Dx, Dy);
%% Simplificación de las distancias, ángulos y Cuadrantes iguales
m=2;
for i=2: Num_move-1
k=i+1;
    if i==m
    while    and(round(Dist_mesh(i))    ==round(Dist_mesh(k)),    round(Ang_mesh(i))
==round(Ang_mesh(k)))
    k=k+1;
    if k>Num_move
        break
    end
    end

    if i==2
    Distances(j)=sum (Dist_mesh (i: k-1));
    Cuadrantes(j)=Cuadrante(i);

    Else

    Distances(j)=sum (Dist_mesh (i: k-1));
    Alfa(j)=Ang_mesh(i);
    Cuadrantes(j)=Cuadrante(i);
%    Angles(j)=Ang_mesh(i)+Angles(j-1);
```

```
    end

    j=j+1;
    m=k;
    end
end

---------- SE EJECUTA FUNCIÓN SENTIDO DE GIRO----------

[ Delta] = f_sentido_giro (Cuadrantes, Alfa, Quad_rbt, Alfa_rob);

---------- SE EJECUTA FUNCIÓN MOVIMIENTO MOTORES----------

[ Mov_RotMotor] = f_mov_motor (Cuadrantes, Alfa, Quad_rbt, Alfa_rob);
Distances(end+1) =5;
end
```

9.6. ANEXO F

CÓDIGO FUNCIÓN CUADRANTES

```
---------- SE EJECUTA FUNCIÓN CUADRANTES----------

function [ cuadrante] = f_cuadrante (Dx, Dy)
%Autor: Santiago Morales
if and((Dx>0), (Dy>0))
        cuadrante = 1;
    elseif and((Dx<0), (Dy>0))
        cuadrante = 2;
    elseif and((Dx<0), (Dy<0))
        cuadrante = 3;
    elseif and((Dx>0), (Dy<0))
        cuadrante = 4;
    elseif and((Dx>0), (Dy==0))
        cuadrante = 5;
    elseif and((Dx==0), (Dy>0))
        cuadrante = 6;
    elseif and((Dx<0), (Dy==0))
        cuadrante = 7;
    elseif and((Dx==0), (Dy<0))
        cuadrante = 8;

end
```

9.7. ANEXO G

CÓDIGO FUNCIÓN SENTIDO DE GIRO

```
---------- SE EJECUTA FUNCIÓN SENTIDO DE GIRO----------

function [ Delta] = f_sentido_giro (Quad_fwd, Alfa, Quad_rbt, Alfa_rob)
Función Sentido de Giro
La función calcula los sentidos de giro ya sea horario o anti horario

                INPUTS
=================================================
Quad_fwd    =   Cuadrante hacia adelante de la ruta.
Alfa        =   Ángulo polar de cada vector.

                OUTPUT
=================================================
Delta    =   Sentido de giro del carro.

Autor: Santiago Morales

%% Precargar variables
Alfa=abs(Alfa);
Delta=zeros(length(Quad_fwd) +1,1);
cond=zeros(length(Quad_fwd) +1,1);
Alfa_rob=abs(Alfa_rob);

%% Calculo del sentido del giro

for i=1: length(Quad_fwd)
   if i==1
      %% Condición 1
      op_log=zeros (8,1);

      % Condición 1a (Entre cuadrante)
        op_log (1) =and (Quad_rbt==1, Quad_fwd(i)==3);
        op_log (2) =and (Quad_rbt==3, Quad_fwd(i)==1);
        op_log (3) =and (Quad_rbt==2, Quad_fwd(i)==4);
        op_log (4) =and (Quad_rbt==4, Quad_fwd(i)==2);

      % Condición 1b (Entre ejes verticales a cuadrantes)
        op_log (5) =and (Quad_rbt==1, Quad_fwd(i)==6);
        op_log (6) =and (Quad_rbt==2, Quad_fwd(i)==6);
        op_log (7) =and (Quad_rbt==3, Quad_fwd(i)==8);
        op_log (8) =and (Quad_rbt==4, Quad_fwd(i)==8);
      cond_parc (1) =max(op_log);

      %% Condición 2
      op_log=zeros (16,1);

      % Condición 2a (Entre cuadrantes)
```

```
  op_log (1) =and (Quad_rbt==1, Quad_fwd(i)==2);
  op_log (2) =and (Quad_rbt==2, Quad_fwd(i)==1);
  op_log (3) =and (Quad_rbt==3, Quad_fwd(i)==4);
  op_log (4) =and (Quad_rbt==4, Quad_fwd(i)==3);

% Condición 2b (Entre ejes horizontales a cuadrantes)
  op_log (5) = and (Quad_rbt==5, Quad_fwd(i)==1);
  op_log (6) =and (Quad_rbt==5, Quad_fwd(i)==4);
  op_log (7) =and (Quad_rbt==7, Quad_fwd(i)==2);
  op_log (8) =and (Quad_rbt==7, Quad_fwd(i)==3);

% Condición 2c (Entre ejes verticales a cuadrantes)
  op_log (9) =and (Quad_rbt==6, Quad_fwd(i)==4);
  op_log (10) =and (Quad_rbt==6, Quad_fwd(i)==3);
  op_log (11) =and (Quad_rbt==8, Quad_fwd(i)==1);
  op_log (12) =and (Quad_rbt==8, Quad_fwd(i)==2);

% Condición 2d (Entre ejes verticales a cuadrantes)
  op_log (13) =and (Quad_rbt==1, Quad_fwd(i)==5);
  op_log (14) =and (Quad_rbt==4, Quad_fwd(i)==5);
  op_log (15) =and (Quad_rbt==2, Quad_fwd(i)==7);
  op_log (16) =and (Quad_rbt==3, Quad_fwd(i)==7);
cond_parc (2) =max(op_log);


%% Condición 3
op_log=zeros (12,1);

% Condición 3a (Entre cuadrantes)
  op_log (1) = and (Quad_rbt==2, Quad_fwd(i)==3);
  op_log (2) =and (Quad_rbt==3, Quad_fwd(i)==2);
  op_log (3) =and (Quad_rbt==1, Quad_fwd(i)==4);
  op_log (4) =and (Quad_rbt==4, Quad_fwd(i)==1);

% Condición 3b (Entre ejes horizontales a cuadrantes)
  op_log (5) = and (Quad_rbt==4, Quad_fwd(i)==7);
  op_log (6) =and (Quad_rbt==1, Quad_fwd(i)==7);
  op_log (7) =and (Quad_rbt==3, Quad_fwd(i)==5);
  op_log (8) =and (Quad_rbt==2, Quad_fwd(i)==5);

% Condición 3c (Entre ejes horizontales a cuadrantes)
  op_log (9) =and (Quad_rbt==7, Quad_fwd(i)==4);
  op_log (10) =and (Quad_rbt==7, Quad_fwd(i)==1);
  op_log (11) =and (Quad_rbt==5, Quad_fwd(i)==2);
  op_log (12) =and (Quad_rbt==5, Quad_fwd(i)==3);
cond_parc (3) =max(op_log);

%% Condición 4 (Entre ejes verticales a cuadrantes)
op_log=zeros (4,1);
  op_log (1) =and (Quad_rbt==6, Quad_fwd(i)==1);
  op_log (2) =and (Quad_rbt==6, Quad_fwd(i)==2);
```

```
        op_log (3) =and (Quad_rbt==8, Quad_fwd(i)==3);
        op_log (4) =and (Quad_rbt==8, Quad_fwd(i)==4);
      cond_parc (4) =max(op_log);

      %% Condición 5 (Entre ejes coordenados, ángulos rectos)
      op_log=zeros (8,1);
        op_log (1) =and (Quad_rbt==5, Quad_fwd(i)==6);
        op_log (2) =and (Quad_rbt==5, Quad_fwd(i)==8);
        op_log (3) =and (Quad_rbt==7, Quad_fwd(i)==6);
        op_log (4) =and (Quad_rbt==7, Quad_fwd(i)==8);
        op_log (5) =and (Quad_rbt==6, Quad_fwd(i)==5);
        op_log (6) =and (Quad_rbt==6, Quad_fwd(i)==7);
        op_log (7) =and (Quad_rbt==8, Quad_fwd(i)==5);
        op_log (8) =and (Quad_rbt==8, Quad_fwd(i)==7);
      cond_parc (5) =max(op_log);

       %% Cálculo del ángulo de giro del robot (Delta)
        if cond_parc (1) ==1
            Delta(i)=Alfa(i)-Alfa_rob;
            cond(i)=1;
        elseif cond_parc (2) ==1
            Delta(i)=Alfa(i)+Alfa_rob;
            cond(i)=2;
        elseif cond_parc (3) ==1
            Delta(i)=180-Alfa(i)-Alfa_rob;
            cond(i)=3;
        elseif cond_parc (4) ==1
            Delta(i)=Alfa_rob-Alfa(i);
            cond(i)=4;
         elseif cond_parc (5) ==1
            Delta(i)=90;
            cond(i)=5;

        end

    else

      %% Condición 1
      op_log=zeros (8,1);

      % Condición 1a (Entre cuadrante)
        op_log (1) =and(Quad_fwd(i-1) ==1, Quad_fwd(i)==3);
        op_log (2) =and(Quad_fwd(i-1) ==3, Quad_fwd(i)==1);
        op_log (3) =and(Quad_fwd(i-1) ==2, Quad_fwd(i)==4);
        op_log (4) =and(Quad_fwd(i-1) ==4, Quad_fwd(i)==2);

      % Condición 1b (Entre ejes verticales a cuadrantes)
        op_log (5) =and(Quad_fwd(i-1) ==1, Quad_fwd(i)==6);
        op_log (6) =and(Quad_fwd(i-1) ==2, Quad_fwd(i)==6);
        op_log (7) =and(Quad_fwd(i-1) ==3, Quad_fwd(i)==8);
        op_log (8) =and(Quad_fwd(i-1) ==4, Quad_fwd(i)==8);
```

```
cond_parc (1) =max(op_log);

%% Condición 2
op_log=zeros (16,1);

% Condición 2a (Entre cuadrantes)
  op_log (1) =and(Quad_fwd(i-1) ==1, Quad_fwd(i)==2);
  op_log (2) =and(Quad_fwd(i-1) ==2, Quad_fwd(i)==1);
  op_log (3) =and(Quad_fwd(i-1) ==3, Quad_fwd(i)==4);
  op_log (4) =and(Quad_fwd(i-1) ==4, Quad_fwd(i)==3);

% Condición 2b (Entre ejes horizontales a cuadrantes)
  op_log (5) = and(Quad_fwd(i-1) ==5, Quad_fwd(i)==1);
  op_log (6) =and(Quad_fwd(i-1) ==5, Quad_fwd(i)==4);
  op_log (7) =and(Quad_fwd(i-1) ==7, Quad_fwd(i)==2);
  op_log (8) =and(Quad_fwd(i-1) ==7, Quad_fwd(i)==3);

% Condición 2c (Entre ejes verticales a cuadrantes)
  op_log (9) =and(Quad_fwd(i-1) ==6, Quad_fwd(i)==4);
  op_log (10) =and(Quad_fwd(i-1) ==6, Quad_fwd(i)==3);
  op_log (11) =and(Quad_fwd(i-1) ==8, Quad_fwd(i)==1);
  op_log (12) =and(Quad_fwd(i-1) ==8, Quad_fwd(i)==2);



% Condición 2d (Entre ejes verticales a cuadrantes)
  op_log (13) =and(Quad_fwd(i-1) ==1, Quad_fwd(i)==5);
  op_log (14) =and(Quad_fwd(i-1) ==4, Quad_fwd(i)==5);
  op_log (15) =and(Quad_fwd(i-1) ==2, Quad_fwd(i)==7);
  op_log (16) =and(Quad_fwd(i-1) ==3, Quad_fwd(i)==7);
cond_parc (2) =max(op_log);

%% Condición 3
op_log=zeros (12,1);

% Condición 3a (Entre cuadrantes)
  op_log (1) = and(Quad_fwd(i-1) ==2, Quad_fwd(i)==3);
  op_log (2) =and(Quad_fwd(i-1) ==3, Quad_fwd(i)==2);
  op_log (3) =and(Quad_fwd(i-1) ==1, Quad_fwd(i)==4);
  op_log (4) =and(Quad_fwd(i-1) ==4, Quad_fwd(i)==1);

% Condición 3b (Entre ejes horizontales a cuadrantes)
  op_log (5) = and(Quad_fwd(i-1) ==4, Quad_fwd(i)==7);
  op_log (6) =and(Quad_fwd(i-1) ==1, Quad_fwd(i)==7);
  op_log (7) =and(Quad_fwd(i-1) ==3, Quad_fwd(i)==5);
  op_log (8) =and(Quad_fwd(i-1) ==2, Quad_fwd(i)==5);

% Condición 3c (Entre ejes horizontales a cuadrantes)
  op_log (9) =and(Quad_fwd(i-1) ==7, Quad_fwd(i)==4);
  op_log (10) =and(Quad_fwd(i-1) ==7, Quad_fwd(i)==1);
  op_log (11) =and(Quad_fwd(i-1) ==5, Quad_fwd(i)==2);
```

```
        op_log (12) =and(Quad_fwd(i-1) ==5, Quad_fwd(i)==3);
      cond_parc (3) =max(op_log);

      %% Condición 4 (Entre ejes verticales a cuadrantes)
      op_log=zeros (4,1);

        op_log (1) =and(Quad_fwd(i-1) ==6, Quad_fwd(i)==1);
        op_log (2) =and(Quad_fwd(i-1) ==6, Quad_fwd(i)==2);
        op_log (3) =and(Quad_fwd(i-1) ==8, Quad_fwd(i)==3);
        op_log (4) =and(Quad_fwd(i-1) ==8, Quad_fwd(i)==4);
      cond_parc (4) =max(op_log);

      %% Condición 5 (Entre ejes coordenados, ángulos rectos)
      op_log=zeros (8,1);
        op_log (1) =and(Quad_fwd(i-1) ==5, Quad_fwd(i)==6);
        op_log (2) =and(Quad_fwd(i-1) ==5, Quad_fwd(i)==8);
        op_log (3) =and(Quad_fwd(i-1) ==7, Quad_fwd(i)==6);
        op_log (4) =and(Quad_fwd(i-1) ==7, Quad_fwd(i)==8);
        op_log (5) =and(Quad_fwd(i-1) ==6, Quad_fwd(i)==5);
        op_log (6) =and(Quad_fwd(i-1) ==6, Quad_fwd(i)==7);
        op_log (7) =and(Quad_fwd(i-1) ==8, Quad_fwd(i)==5);
        op_log (8) =and(Quad_fwd(i-1) ==8, Quad_fwd(i)==7);
      cond_parc (5) =max(op_log);

       %% Cálculo del ángulo de giro del robot (Delta)

        if cond_parc (1) ==1
            Delta(i)=Alfa(i)-Alfa(i-1);
            cond(i)=1;
        elseif cond_parc (2) ==1
            Delta(i)=Alfa(i)+Alfa(i-1);
            cond(i)=2;
        elseif cond_parc (3) ==1
            Delta(i)=180-Alfa(i)-Alfa(i-1);
            cond(i)=3;
        elseif cond_parc (4) ==1
            Delta(i)=Alfa(i-1)-Alfa(i);
            cond(i)=4;
         elseif cond_parc (5) ==1
            Delta(i)=90;
            cond(i)=5;

        end
   end
end
end
end
```

9.8. ANEXO H

CÓDIGO FUNCIÓN MOVIMIENTO MOTORES

```
---------- SE EJECUTA FUNCIÓN MOVIMIENTO MOTORES----------

function [ Mov_RotMotor] = f_mov_motor (Quad_fwd, Alfa, Quad_rbt, Alfa_rob)
Función Movimiento de motores
La función calcula los giros de los motores del robot

                INPUTS
=================================================
Quad_fwd    =   Cuadrante hacia adelante de la ruta.
Quad_rbt    =   Cuadrante hacia atrás de la ruta.
Alfa         =   Ángulo polar de cada vector.
Alfa rob     =   Ángulo polar anterior.

                OUTPUT
=================================================
Mov_Motor   =   Movimiento del motor.

Autor: Santiago Morales

Alfa=abs(Alfa);
Mov_RotMotor=zeros(length(Quad_fwd) +1,1);
cond=zeros(length(Quad_fwd) +1,1);

%%Calculo condición inicial
%% Condición 1 (Motor Derecho)
for i=1: length(Quad_fwd)
   if i==1
      op_log=zeros (24,1);
      % Condición 1er cuadrante
        op_log (1) =and (Quad_rbt==1, Quad_fwd(i)==2);
        op_log (2) =and (Quad_rbt==1, Quad_fwd(i)==6);
        op_log (3) =and (Quad_rbt==1, Quad_fwd(i)==7);

      % Condición 2do cuadrante
        op_log (4) =and (Quad_rbt==2, Quad_fwd(i)==3);
        op_log (5) =and (Quad_rbt==2, Quad_fwd(i)==7);
        op_log (6) =and (Quad_rbt==2, Quad_fwd(i)==8);

      % Condición 3er cuadrante
        op_log (7) =and (Quad_rbt==3, Quad_fwd(i)==4);
        op_log (8) =and (Quad_rbt==3, Quad_fwd(i)==5);
        op_log (9) =and (Quad_rbt==3, Quad_fwd(i)==8);

      % Condición 4to cuadrante
        op_log (10) =and (Quad_rbt==4, Quad_fwd(i)==1);
        op_log (11) =and (Quad_rbt==4, Quad_fwd(i)==5);
        op_log (12) =and (Quad_rbt==4, Quad_fwd(i)==6);
```

```
  % Condición 5to cuadrante
  op_log (13) =and (Quad_rbt==5, Quad_fwd(i)==1);
  op_log (14) =and (Quad_rbt==5, Quad_fwd(i)==2);
  op_log (15) =and (Quad_rbt==5, Quad_fwd(i)==6);

  % Condición 6to cuadrante
  op_log (16) =and (Quad_rbt==6, Quad_fwd(i)==2);
  op_log (17) =and (Quad_rbt==6, Quad_fwd(i)==3);
  op_log (18) =and (Quad_rbt==6, Quad_fwd(i)==7);

  % Condición 7mo cuadrante
  op_log (19) =and (Quad_rbt==7, Quad_fwd(i)==3);
  op_log (20) =and (Quad_rbt==7, Quad_fwd(i)==4);
  op_log (21) =and (Quad_rbt==7, Quad_fwd(i)==8);

  % Condición 8vo cuadrante
  op_log (22) =and (Quad_rbt==8, Quad_fwd(i)==1);
  op_log (23) =and (Quad_rbt==8, Quad_fwd(i)==4);
  op_log (24) =and (Quad_rbt==8, Quad_fwd(i)==7);
 cond_parc (1) =max(op_log);

%% Condición 2 (Motor Izquierdo)
op_log=zeros (24,1);
% Condición 1er cuadrante
  op_log (1) =and (Quad_rbt==1, Quad_fwd(i)==4);
  op_log (2) =and (Quad_rbt==1, Quad_fwd(i)==5);
  op_log (3) =and (Quad_rbt==1, Quad_fwd(i)==8);

% Condición 2do cuadrante
  op_log (4) =and (Quad_rbt==2, Quad_fwd(i)==1);
  op_log (5) =and (Quad_rbt==2, Quad_fwd(i)==5);
  op_log (6) =and (Quad_rbt==2, Quad_fwd(i)==6);

% Condición 3er cuadrante
  op_log (7) =and (Quad_rbt==3, Quad_fwd(i)==2);
  op_log (8) =and (Quad_rbt==3, Quad_fwd(i)==6);
  op_log (9) =and (Quad_rbt==3, Quad_fwd(i)==7);

% Condición 4to cuadrante
  op_log (10) =and (Quad_rbt==4, Quad_fwd(i)==3);
  op_log (11) =and (Quad_rbt==4, Quad_fwd(i)==7);
  op_log (12) =and (Quad_rbt==4, Quad_fwd(i)==8);

  % Condición 5to cuadrante
  op_log (13) =and (Quad_rbt==5, Quad_fwd(i)==3);
  op_log (14) =and (Quad_rbt==5, Quad_fwd(i)==4);
  op_log (15) =and (Quad_rbt==5, Quad_fwd(i)==8);

  % Condición 6to cuadrante
  op_log (16) =and (Quad_rbt==6, Quad_fwd(i)==1);
```

```
  op_log (17) =and (Quad_rbt==6, Quad_fwd(i)==4);
  op_log (18) =and (Quad_rbt==6, Quad_fwd(i)==5);

  % Condición 7mo cuadrante
  op_log (19) =and (Quad_rbt==7, Quad_fwd(i)==1);
  op_log (20) =and (Quad_rbt==7, Quad_fwd(i)==2);
  op_log (21) =and (Quad_rbt==7, Quad_fwd(i)==6);

  % Condición 8vo cuadrante
  op_log (22) =and (Quad_rbt==8, Quad_fwd(i)==2);
  op_log (23) =and (Quad_rbt==8, Quad_fwd(i)==3);
  op_log (24) =and (Quad_rbt==8, Quad_fwd(i)==5);
cond_parc (2) =max(op_log);

  %% Condición 3 (Dos Motores)
 op_log=zeros (4,1);
  op_log (1) =and (Quad_rbt==5, Quad_fwd(i)==5);
  op_log (2) =and (Quad_rbt==6, Quad_fwd(i)==6);
  op_log (3) =and (Quad_rbt==7, Quad_fwd(i)==7);
  op_log (4) =and (Quad_rbt==8, Quad_fwd(i)==8);
 cond_parc (3) =max(op_log);

%% Condición 4 (Motor 45°)
op_log=zeros (4,1);
  op_log (1) =and (Quad_rbt==1, Quad_fwd(i)==1);
  op_log (2) =and (Quad_rbt==2, Quad_fwd(i)==2);
  op_log (3) =and (Quad_rbt==3, Quad_fwd(i)==3);
  op_log (4) =and (Quad_rbt==4, Quad_fwd(i)==4);
cond_parc (4) =max(op_log);

  %% Cálculo movimiento de motores
  % En donde 1=Enciende motor derecho
  %              -1=Enciende motor izquierdo
  %               0=Enciende los dos motores

  if cond_parc (1) ==1
     Mov_RotMotor(i)=1;
      cond(i)=1;

  elseif cond_parc (2) ==1
     Mov_RotMotor(i)=-1;
      cond(i)=2;

  elseif cond_parc (3) ==1
     Mov_RotMotor(i)=0;
      cond(i)=3;

  elseif cond_parc (4) ==1
       if or (op_log (1) ==1, op_log (4) ==1)
          if Alfa (1)>45
          Mov_RotMotor(i)=1;
```

```
                else
                Mov_RotMotor(i)=-1;
                end
            elseif or (op_log (2) ==1, op_log (3) ==1)
                if Alfa (1)>45
                Mov_RotMotor(i)=-1;
                else
                Mov_RotMotor(i)=1;
                end
            end
           cond(i)=4;
        end
else

    %%Calculo del resto de ruta

    %% Condición 1 (Motor Derecho)
    op_log=zeros (24,1);
    % Condición 1er cuadrante
     op_log (1) =and(Quad_fwd(i-1) ==1, Quad_fwd(i)==2);
     op_log (2) =and(Quad_fwd(i-1) ==1, Quad_fwd(i)==6);
     op_log (3) =and(Quad_fwd(i-1) ==1, Quad_fwd(i)==7);

    % Condición 2do cuadrante
     op_log (4) =and(Quad_fwd(i-1) ==2, Quad_fwd(i)==3);
     op_log (5) =and(Quad_fwd(i-1) ==2, Quad_fwd(i)==7);
     op_log (6) =and(Quad_fwd(i-1) ==2, Quad_fwd(i)==8);

    % Condición 3er cuadrante
     op_log (7) =and(Quad_fwd(i-1) ==3, Quad_fwd(i)==4);
     op_log (8) =and(Quad_fwd(i-1) ==3, Quad_fwd(i)==5);
     op_log (9) =and(Quad_fwd(i-1) ==3, Quad_fwd(i)==8);

    % Condición 4to cuadrante
     op_log (10) =and(Quad_fwd(i-1) ==4, Quad_fwd(i)==1);
     op_log (11) =and(Quad_fwd(i-1) ==4, Quad_fwd(i)==5);
     op_log (12) =and(Quad_fwd(i-1) ==4, Quad_fwd(i)==6);

     % Condición 5to cuadrante
     op_log (13) =and(Quad_fwd(i-1) ==5, Quad_fwd(i)==1);
     op_log (14) =and(Quad_fwd(i-1) ==5, Quad_fwd(i)==2);
     op_log (15) =and(Quad_fwd(i-1) ==5, Quad_fwd(i)==6);

     % Condición 6to cuadrante
     op_log (16) =and(Quad_fwd(i-1) ==6, Quad_fwd(i)==2);
     op_log (17) =and(Quad_fwd(i-1) ==6, Quad_fwd(i)==3);
     op_log (18) =and(Quad_fwd(i-1) ==6, Quad_fwd(i)==7);

     % Condición 7mo cuadrante
     op_log (19) =and(Quad_fwd(i-1) ==7, Quad_fwd(i)==3);
     op_log (20) =and(Quad_fwd(i-1) ==7, Quad_fwd(i)==4);
```

```
  op_log (21) =and(Quad_fwd(i-1) ==7, Quad_fwd(i)==8);

  % Condición 8vo cuadrante
  op_log (22) =and(Quad_fwd(i-1) ==8, Quad_fwd(i)==1);
  op_log (23) =and(Quad_fwd(i-1) ==8, Quad_fwd(i)==4);
  op_log (24) =and(Quad_fwd(i-1) ==8, Quad_fwd(i)==7);
 cond_parc (1) =max(op_log);

%% Condición 2 (Motor Izquierdo)
op_log=zeros (24,1);
% Condición 1er cuadrante
  op_log (1) =and(Quad_fwd(i-1) ==1, Quad_fwd(i)==4);
  op_log (2) =and(Quad_fwd(i-1) ==1, Quad_fwd(i)==5);
  op_log (3) =and(Quad_fwd(i-1) ==1, Quad_fwd(i)==8);

% Condición 2do cuadrante
  op_log (4) =and(Quad_fwd(i-1) ==2, Quad_fwd(i)==1);
  op_log (5) =and(Quad_fwd(i-1) ==2, Quad_fwd(i)==5);
  op_log (6) =and(Quad_fwd(i-1) ==2, Quad_fwd(i)==6);

% Condición 3er cuadrante
  op_log (7) =and(Quad_fwd(i-1) ==3, Quad_fwd(i)==2);
  op_log (8) =and(Quad_fwd(i-1) ==3, Quad_fwd(i)==6);
  op_log (9) =and(Quad_fwd(i-1) ==3, Quad_fwd(i)==7);

% Condición 4to cuadrante
  op_log (10) =and(Quad_fwd(i-1) ==4, Quad_fwd(i)==3);
  op_log (11) =and(Quad_fwd(i-1) ==4, Quad_fwd(i)==7);
  op_log (12) =and(Quad_fwd(i-1) ==4, Quad_fwd(i)==8);

  % Condición 5to cuadrante
  op_log (13) =and(Quad_fwd(i-1) ==5, Quad_fwd(i)==3);
  op_log (14) =and(Quad_fwd(i-1) ==5, Quad_fwd(i)==4);
  op_log (15) =and(Quad_fwd(i-1) ==5, Quad_fwd(i)==8);

  % Condición 6to cuadrante
  op_log (16) =and(Quad_fwd(i-1) ==6, Quad_fwd(i)==1);
  op_log (17) =and(Quad_fwd(i-1) ==6, Quad_fwd(i)==4);
  op_log (18) =and(Quad_fwd(i-1) ==6, Quad_fwd(i)==5);

  % Condición 7mo cuadrante
  op_log (19) =and(Quad_fwd(i-1) ==7, Quad_fwd(i)==1);
  op_log (20) =and(Quad_fwd(i-1) ==7, Quad_fwd(i)==2);
  op_log (21) =and(Quad_fwd(i-1) ==7, Quad_fwd(i)==6);

  % Condición 8vo cuadrante
  op_log (22) =and(Quad_fwd(i-1) ==8, Quad_fwd(i)==2);
  op_log (23) =and(Quad_fwd(i-1) ==8, Quad_fwd(i)==3);
  op_log (24) =and(Quad_fwd(i-1) ==8, Quad_fwd(i)==5);
cond_parc (2) =max(op_log);
```

```
        %% Condición 3 (Dos Motores)
       op_log=zeros (4,1);
        op_log (1) =and(Quad_fwd(i-1) ==5, Quad_fwd(i)==5);
        op_log (2) =and(Quad_fwd(i-1) ==6, Quad_fwd(i)==6);
        op_log (3) =and(Quad_fwd(i-1) ==7, Quad_fwd(i)==7);
        op_log (4) =and(Quad_fwd(i-1) ==8, Quad_fwd(i)==8);
       cond_parc (3) =max(op_log);

      %% Condición 4 (Motor 45°)
      op_log=zeros (4,1);
        op_log (1) =and(Quad_fwd(i-1) ==1, Quad_fwd(i)==1);
        op_log (2) =and(Quad_fwd(i-1) ==2, Quad_fwd(i)==2);
        op_log (3) =and(Quad_fwd(i-1) ==3, Quad_fwd(i)==3);
        op_log (4) =and(Quad_fwd(i-1) ==4, Quad_fwd(i)==4);
      cond_parc (4) =max(op_log);

        %% Cálculo movimiento de motores
        % En donde 1=Enciende motor derecho
        %              -1=Enciende motor izquierdo
        %               0=Enciende los dos motores
        if cond_parc (1) ==1
          Mov_RotMotor(i)=1;
           cond(i)=1;

        elseif cond_parc (2) ==1
          Mov_RotMotor(i)=-1;
           cond(i)=2;

        elseif cond_parc (3) ==1
          Mov_RotMotor(i)=0;
           cond(i)=3;

        elseif cond_parc (4) ==1
             if or (op_log (1) ==1, op_log (4) ==1)
                if Alfa(i)>45
                Mov_RotMotor(i)=1;
                else
                Mov_RotMotor(i)=-1;
                end
             elseif or (op_log (2) ==1, op_log (3) ==1)
                if Alfa(i)>45
                Mov_RotMotor(i)=-1;
                else
                Mov_RotMotor(i)=1;
                end
             end
           cond(i)=4;
        end
   end
end
end
```

9.9. ANEXO I

CÓDIGO INTERFAZ GRÁFICA APLICACIÓN

```
Autor: Santiago Morales

function varargout = Guide_Tesis1(varargin)
clc
gui_Singleton = 1;
gui_State = struct ('gui_Name', mfilename, ...
    'gui_Singleton', gui_Singleton, ...
    'gui_OpeningFcn', @Guide_Tesis1_OpeningFcn, ...
    'gui_OutputFcn', @Guide_Tesis1_OutputFcn, ...
    'gui_LayoutFcn', [], ...
    'gui_Callback', []);
if nargin && ischar (varargin {1})
    gui_State.gui_Callback = str2func (varargin {1});
end

if Margot
    [varargout {1: nargout}] = gui_mainfcn (gui_State, varargin {:});
else
    gui_mainfcn (gui_State, varargin {:});
end

% --- Executes just before Guide_Tesis1 is made visible.
function Guide_Tesis1_OpeningFcn (hObject, eventdata, handles, varargin)
handles.c='1';
handles. output = hObject;
guidata (hObject, handles);

function varargout = Guide_Tesis1_OutputFcn (hObject, eventdata, handles)
varargout {1} = handles. output;

% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback (hObject, eventdata, handles)
C=get(hObject,'Value'); %Preguntamos en cuál renglón del popupmenu2 estamos
if C==1 %Cambiamos el color de la gráfica dependiendo de cuál opción elijamos.
    handles.c=1;
elseif C==2
    handles.c=2;
elseif C==3
    handles.c=3;
elseif C==4
    handles.c=4;
elseif C==5
    handles.c=5;
elseif C==6
    handles.c=6;
elseif C==7
    handles.c=7;
```

```
end
guidata (hObject, handles);

% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn (hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit1_Callback (hObject, eventdata, handles)
v=str2num(get(hObject,'String'));
guidata (handles. edit1, handles);

% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn (hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

---------- SE EJECUTA PRESIONANDO EL BOTÓN CAPTURAR ----------

```
function Capturar_Callback (hObject, eventdata, handles)
texto=get (handles. edit1,'String');
Dist_cam=str2num(texto); %Escritura de distancia de Cámara
X=strcat(handles.c) %Seleccion de Cantidad de obstáculos
O=double(X);
redThresh = 0.2; % Cálculo de umbrales para la binarización del mapa de grises
greenThresh = 0.05; %
blueThresh = 0.15; %
whiteThresh = 0.4;

global ResC_x;
global ResC_y;
global Dist_f;
global Z_cam;
global Fconv;
global apagar;
global Inicio;
global vidDevice;
global hVideoIn;
global vidIn;

%Variables de Cámara y cálculo de factor de conversión según altura
ResC_x=640;
ResC_y=480;
Dwind_x=161; %mm
Dwind_y=121; %mm
Dist_f=200.9840988; %mm
Z_cam=Dist_cam; % mm
```

```
Esc=Z_cam/Dist_f;
Dist_x=Dwind_x*Esc;
Dist_y=Dwind_y*Esc;
Fconv= mean ([Dist_x/ResC_x, Dist_y/ResC_y]);
Inicio=1;

while(Inicio)

   vidDevice = imaq. VideoDevice ('winvideo', 3, 'YUY2_640x480', 'ROI', [1 1 ResC_x
ResC_y], 'ReturnedColorSpace', 'rgb');
   frame = step(vidDevice);
   vidInfo = imaqhwinfo(vidDevice);
   hblob = vision. BlobAnalysis ('AreaOutputPort', false, 'CentroidOutputPort', true,
'BoundingBoxOutputPort', true', 'MinimumBlobArea', 100, 'MaximumBlobArea', 3000,
'MaximumCount', 1); % Configuración del reconocimiento de objetos para colores
   hblobWhite = vision. BlobAnalysis ('AreaOutputPort', false, 'CentroidOutputPort', true,
'BoundingBoxOutputPort', true', 'MinimumBlobArea', 300, 'MaximumBlobArea', 3000,
'MaximumCount', O); % Configuración del reconocimiento de objetos para negro
   hshapeinsBox = vision. ShapeInserter ('BorderColorSource', 'Input port', 'Fill', true,
'FillColorSource', 'Input port', 'Opacity', 0.4); %para el texto en la ventana de video
   htextinsRed = vision. TextInserter ('Text', 'Red: %2d', 'Location', [5 2], 'Color', [1 0 0],
'Font', 'Courier New', 'FontSize', 14);
   htextinsGreen = vision. TextInserter ('Text', 'Green: %2d', 'Location', [5 18], 'Color', [0
1 0], 'Font', 'Courier New', 'FontSize', 14);
   htextinsBlue = vision. TextInserter ('Text', 'Blue: %2d', 'Location', [5 34], 'Color', [0 0
1], 'Font', 'Courier New', 'FontSize', 14);
   htextinsWhite = vision. TextInserter ('Text', 'White: %2d', 'Location', [5 50], 'Color', [1
1 1], 'Font', 'Courier New', 'FontSize', 14);
   htextinsCent = vision. TextInserter ('Text', '+      X: %4d, Y: %4d', ... % para el
centroide al lado de cada figura
       'LocationSource', 'Input port', ...
       'Color', [1 1 0], ... // yellow color
       'Font', 'Courier New', ...
       'FontSize', 17);
   hVideoIn = vision. VideoPlayer ('Name', 'Final Video', 'Position', [100 100
vidInfo.MaxWidth+20 vidInfo.MaxHeight+30]); % Salida de Video

   %% filtrado de colores
   global ejecutar;
   global centXRed;
   global centYRed;
   global centXGreen;
   global centYGreen;
   global centXBlue;
   global centYBlue;
   global centXWhite;
   global centYWhite;

   ejecutar=1;
   apagar=1;
```

```
   while(ejecutar)
      rgbFrame = step(vidDevice); % adquirir el frame


      diffFrameRed = imsubtract (rgbFrame (:1), rgb2gray(rgbFrame)); % se toma el
componente rojo
      diffFrameRed = medfilt2(diffFrameRed, [3 3]); % filtra
      binFrameRed = imbinarize (diffFrameRed, redThresh); % blanco y negro

      diffFrameGreen = imsubtract (rgbFrame (:2), rgb2gray(rgbFrame));
      diffFrameGreen = medfilt2(diffFrameGreen, [3 3]);
      binFrameGreen = imbinarize (diffFrameGreen, greenThresh);

      diffFrameBlue = imsubtract (rgbFrame (:3), rgb2gray(rgbFrame));
      diffFrameBlue = medfilt2(diffFrameBlue, [3 3]);
      binFrameBlue = imbinarize (diffFrameBlue, blueThresh);

      bwredFrame = imbinarize (rgbFrame (:1), whiteThresh); %obtiene blanco y negro
para el umbral que mejor funcionó
      bwgreenFrame = imbinarize (rgbFrame (:2), whiteThresh);
      bwblueFrame = imbinarize (rgbFrame (:3), whiteThresh);

      binFrameWhite = bwredFrame | bwgreenFrame | bwblueFrame; %las combino
      binFrameWhite = medfilt2(binFrameWhite, [3 3]);
      binFrameWhite=imcomplement(binFrameWhite); %intercambia blanco con negro

      [centroidRed, bboxRed] = step (hblob, binFrameRed); % se obtienen los centroides
para el rojo y la caja que lo encierra con vision. BlobAnalysis
      centroidRed = uint16(centroidRed); % se convierte a entero

      [centroidGreen, bboxGreen] = step (hblob, binFrameGreen);
      centroidGreen = uint16(centroidGreen);

      [centroidBlue, bboxBlue] = step (hblob, binFrameBlue);
      centroidBlue = uint16(centroidBlue);

      [centroidWhite, bboxWhite] = step (hblobWhite, binFrameWhite);
      centroidWhite = uint16(centroidWhite);

      %Para escribir en la imagen los centroides y dibujar los rectangulos de cada color

      rgbFrame (1:70,1:90, :) = 0;
      vidIn = step (hshapeinsBox, rgbFrame, bboxRed, single ([1 0 0])); % agrega cuadro
rojo
      vidIn = step (hshapeinsBox, vidIn, bboxGreen, single ([0 1 0])); % agrega cuadro
verde
      vidIn = step (hshapeinsBox, vidIn, bboxBlue, single ([0 0 1])); % agrega cuadro azul
      vidIn = step (hshapeinsBox, vidIn, bboxWhite, single ([1 1 1])); % agrega cuadro
blanco
```

```
      for object = 1:1: length (bboxRed (:1)) % escribe el centroide para el elemento rojo
         centXRed = centroidRed(object,1);
         centYRed = centroidRed(object,2);

         vidIn = step (htextinsCent, vidIn, [centXRed centYRed], [centXRed-6 centYRed-
9]);
      end

      for object = 1:1: length (bboxGreen (:1)) %
         centXGreen = centroidGreen(object,1);
         centYGreen = centroidGreen(object,2);

         vidIn = step (htextinsCent, vidIn, [centXGreen centYGreen], [centXGreen-6
centYGreen-9]);
      end

      for object = 1:1: length (bboxBlue (:1)) %
         centXBlue = centroidBlue(object,1);
         centYBlue = centroidBlue(object,2);

         vidIn = step (htextinsCent, vidIn, [centXBlue centYBlue], [centXBlue-6 centYBlue-
9]);
      end

      centXWhite= [];
      centXWhite=uint16(centXWhite);
      centYWhite= [];
      centYWhite=uint16(centYWhite);

      for object = 1:1: length (bboxWhite (:1))
         centXWhite(object) = centroidWhite(object,1);
         centYWhite(object) = centroidWhite(object,2);
         vidIn = step (htextinsCent, vidIn, [centXWhite(object) centYWhite(object)],
[centXWhite(object)-6 centYWhite(object)-9]);
      end

      vidIn = step (htextinsRed, vidIn, uint8(length (bboxRed (:1)))); % Contador de
objetos Rojos
      vidIn = step (htextinsGreen, vidIn, uint8(length (bboxGreen (:1)))); % Contador de
objetos Verdes
      vidIn = step (htextinsBlue, vidIn, uint8(length (bboxBlue (:1)))); % Contador de
objetos Azules
      vidIn = step (htextinsWhite, vidIn, uint8(length (bboxWhite (:1)))); % Contador de
objetos Negros
      step (hVideoIn, vidIn); % Salida de video

      if apagar == 0
         break
      end
   end
```

```
    if Inicio == 0
        release(vidDevice);
        release(hVideoIn);
        close ('Final Video');
        break
    end
end

guidata (hObject, handles);
```

---------- SE EJECUTA PRESIONANDO EL BOTÓN CALCULAR ----------

```
function Calcular_Callback (hObject, eventdata, handles)

global centXRed; % objetivo
global centYRed;
global centXGreen; % robot
global centYGreen;
global centXBlue; % robot
global centYBlue;
global centXWhite; %Obstáculos
global centYWhite;
global mapa;
global ruta;
global ResC_x;
global ResC_y;
global Optimal_path;
global lim_x;
global lim_y;
global Target;
global Rob_back;
global Rob_front;

%% Tamaño de los obstáculos y cuadricula del mapa

disp(length(centXWhite'))
eje_y = 9;
eje_x=floor (((640/480) *eje_y));
lim_y=ResC_y/eje_y;
lim_x=ResC_x/eje_x;
mapa=2*ones (eje_y, eje_x);

for i=1: length(centXWhite')
    disp(i)
    NewRes_y=ceil(double(centYWhite(i)) /lim_y);
    NewRes_x=ceil(double(centXWhite(i)) /lim_x);
    mapa (NewRes_y, NewRes_x) =-1; %Resolución/10

    if NewRes_y<eje_y
        mapa (NewRes_y+1, NewRes_x) =-1;
```

```
    end

    if NewRes_y>1
        mapa (NewRes_y-1, NewRes_x) =-1;
    end

    if NewRes_x<eje_x
        mapa (NewRes_y, NewRes_x+1) =-1;
    end

    if NewRes_x>1
        mapa (NewRes_y, NewRes_x-1) =-1;
    end

    if NewRes_y<eje_y & NewRes_x<eje_x
        mapa (NewRes_y+1, NewRes_x+1) =-1;
    end

    if NewRes_y<eje_y & NewRes_x>1
        mapa (NewRes_y+1, NewRes_x-1) =-1;
    end

    if NewRes_y>1 & NewRes_x<eje_x
        mapa (NewRes_y-1, NewRes_x+1) =-1;
    end

    if NewRes_y>1 & NewRes_x>1
        mapa (NewRes_y-1, NewRes_x-1) =-1;
    end

end

mapa(ceil(double(centYRed)/lim_y), ceil(double(centXRed)/lim_x)) =0;
mapa(ceil(double(centYBlue)/lim_y), ceil(double(centXBlue)/lim_x)) =1;
mapa(ceil(double(centYGreen)/lim_y), ceil(double(centXGreen)/lim_x)) =3;
map=mapa;

%%Se ejecuta función Astar

[xTarget, yTarget]=find(map==0);
Target= [xTarget, yTarget];
[xvalb, yvalb]=find(map==3);
Rob_back= [xvalb, yvalb];
[xval, yval]=find(map==1);
Rob_front= [xval, yval];

[ Optimal_path, bandera] = astar (map, eje_x, eje_y);

%% Cálculo de trayectoria
ruta = Optimal_path;
if(bandera)
```

```
    ruta2=ruta; %Para el dibujo no es necesario dibujar el último paso por eso se hace necesario borrar ese último cuadro
    ruta2(1, :)= []; %
    for i=1: size(ruta2)
        map(ruta2(i,1), ruta2(i,2)) =10; % coloca 10 en las casillas correspondientes a la ruta

    end
end
Optimal_path=ruta2;

%% Graficar ruta
r=map;
g=map;
b=map;
e=map;

r(r==0|r==-1) =255;
r(r~=255) =0;
g(g==-1|g==10) =255;
g(g~=255) =0;
b(b==1|b==-1|b==10) =255;
b(b~=255) =0;
e(e==10|e==3) =255;
e(e~=255) =0;

col (:1) =r;
col (:2) =g;
col (:3) =b;
col (:2) =e;

imshow(col,'InitialMagnification','Fit');
```

---------- SE EJECUTA PRESIONANDO EL BOTÓN EJECUTAR ----------

```
function Ejecutar_Callback (hObject, eventdata, handles)

greenThresh = 0.05; % Umbral para detección color verde
blueThresh = 0.15; % Umbral para la detección color azul

global ResC_x;
global ResC_y;
ResC_x=640;
ResC_y=480;
global izquierda;
global derecha;
global centXRed;
global centYRed;
global centXGreen;
global centYGreen;
global centXBlue;
```

```
global centYBlue;
global Target;
global Rob_back;
global Rob_front;
global umbralAng;
global umbralDis;
global Optimal_path;
global lim_x;
global lim_y;
global Fconv;

izquierda=NXTMotor('C');
derecha=NXTMotor('B');
umbralAng=15;
umbralDis=20;

Rob_back_pixels= [Rob_back (2) *lim_x, Rob_back (1) *lim_y];
Rob_front_pixels= [Rob_front (2) *lim_x, Rob_front (1) *lim_y];
Target_pixels= [Target (2) *lim_x, Target (1) *lim_y];
Path_pixels= flipa ([Optimal_path (:2). * lim_x, Optimal_path (:1). *lim_y]);

[ distancia, angulo, Mov_RM] = f_Polar_Path (Rob_back_pixels, Rob_front_pixels,
Target_pixels, Path_pixels, Fconv);
Num_paths=length(angulo); % Cantidad de trayectorias que va a realizar el robot
conv_angmot_mm=2.083; % Conversión °rot motor/mm
conv_angmot_ang=6.0; % Conversión °rot motor/°sexagesimales
ang_rot_motor=round (distancia. *conv_angmot_mm); % En grados de rotaciones de
motor
QuarterTurnTicks=Mov_RM. *round (angulo. *conv_angmot_ang); % En grados de motor

ctr_ang=-ones (Num_paths, 1);

for i=1: Num_paths
    if mod(i,2) ==0
        ctr_ang(i)=1;
    end
end

%% Comunicación de Trayectoria Calculada al Robot
COM_CloseNXT all
Ports = [MOTOR_B; MOTOR_C]; % Puertos del motor para rueda izquierda y derecha
DrivingSpeed     = 50; %Velocidad del Robot

% Abre conexión Bluetooth
h = COM_OpenNXT('bluetooth.ini');
COM_SetDefaultNXT(h);

% Inicia las características del motor
for i=1: Num_paths
    if i==1
        %Velocidad de giro del Robot
```

```
    if QuarterTurnTicks(i)==0
       TurningSpeed     = 40;
    else
       TurningSpeed     = 40*(QuarterTurnTicks(i)/abs(QuarterTurnTicks(i)));
    end
    mStraight                = NXTMotor(Ports);
    mStraight.SpeedRegulation   = false; % not for sync mode
    mStraight.Power             = DrivingSpeed;
    mStraight.TachoLimit        = ang_rot_motor(i);
    mStraight.ActionAtTachoLimit = 'Brake';
    mTurn1                   = NXTMotor (Ports (2)); % ports swapped because it's nicer
    mTurn1.SpeedRegulation      = false; % we could use it if we wanted
    mTurn1.Power             = -abs(TurningSpeed);
    mTurn1.TachoLimit          = abs(QuarterTurnTicks(i));
    mStraight.ActionAtTachoLimit = 'Brake';
    mTurn2         = mTurn1;
    mTurn2.Port     = Ports (1); % ports swapped again...
    mTurn2.Power    = mTurn1.Power;
    % Inicializa los motores
   % Se envía este comando en caso de que aún esten girando los motores
    mStraight.Stop('off');

    if QuarterTurnTicks(i)~=0
       if TurningSpeed<0
          mTurn1.SendToNXT();
          mTurn1.WaitFor();

       elseif TurningSpeed>0
          mTurn2.SendToNXT();
          mTurn2.WaitFor();
       end
    end
    % Ejecuta movimiento
    mStraight.SendToNXT();
    % Espera que finalice el movimiento
    mStraight.WaitFor();
    % Ahora espera para realizar el giro
else

    %Velocidad de giro del Robot
    if QuarterTurnTicks(i)==0
       TurningSpeed     = 40;
    else
       TurningSpeed     = 40*(QuarterTurnTicks(i)/abs(QuarterTurnTicks(i)));
    end
    mStraight                = NXTMotor(Ports);
    mStraight.SpeedRegulation   = false; % not for sync mode
    mStraight.Power             = DrivingSpeed;
    mStraight.TachoLimit        = ang_rot_motor(i);
    mStraight.ActionAtTachoLimit = 'Brake';
```

```
      mTurn1                          = NXTMotor (Ports (2)); % ports swapped because
it's nicer
      mTurn1.SpeedRegulation      = false; % we could use it if we wanted
      mTurn1.Power                = ctr_ang(i)*abs(TurningSpeed);
      mTurn1.TachoLimit           = abs(QuarterTurnTicks(i));
      mStraight.ActionAtTachoLimit = 'Brake';
      mTurn2          = mTurn1;
      mTurn2.Port     = Ports (1); % ports swapped again...
      mTurn2.Power    = mTurn1.Power;
      % Inicializa los motores
     % Se envía este comando en caso de que aún esten girando los motores
      mStraight.Stop('off');
      if ctr_ang(i)==1
         if QuarterTurnTicks(i)~=0
            if TurningSpeed>0
               mTurn1.SendToNXT();
               mTurn1.WaitFor();

            elseif TurningSpeed<0
               mTurn2.SendToNXT();
               mTurn2.WaitFor();
            end
         end
      else
         if QuarterTurnTicks(i)~=0
            if TurningSpeed<0
               mTurn1.SendToNXT();
               mTurn1.WaitFor();

            elseif TurningSpeed>0
               mTurn2.SendToNXT();
               mTurn2.WaitFor();
            end
         end

      end

      % Ejecuta movimiento
      mStraight.SendToNXT();
      % Espera que finalice el movimiento
      mStraight.WaitFor();
      % Ahora espera para realizar el giro
   end
end

% Inicializa los motores
% Se envía este comando en caso de que aún esten girando los motores
mStraight.Stop('off');
% Se apagan los motores
mStraight.Stop('off');
% Se cierra la conexión Bluetooth
```

```
COM_CloseNXT(h);

---------SE EJECUTA PRESIONANDO EL BOTÓN DETENER CAPTURA----------

function Detener_Captura_Callback (hObject, eventdata, handles)
global apagar;
global Inicio;
apagar=0;
Inicio=0;

---------------SE EJECUTA PRESIONANDO EL BOTÓN CERRAR ---------------

function Cerrar_Callback (hObject, eventdata, handles)
clear all
clc
close all
```