UNIVERSIDAD POLITÉCNICA SALESIANA

SEDE QUITO

CARRERA DE INGENIERÍA ELECTRÓNICA

ANÁLISIS COMPARATIVO ENTRE EL ALGORITMO A* Y EL ALGORITMO TIPO INSECTO PARA LA PLANEACIÓN DE RUTAS APLICADOS SOBRE UN ROBOT MÓVIL CON RUEDAS

Trabajo de titulación previo a la obtención del
Título de Ingeniero Electrónico

AUTOR: José Vicente Arias Ganchala
Alex Javier García Patiño
TUTOR: Carmen Johanna Celi Sánchez

Quito-Ecuador
2022

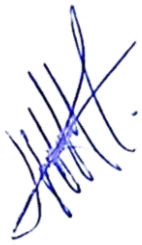
CERTIFICADO DE RESPONSABILIDAD Y AUTORÍA DEL TRABAJO DE TITULACIÓN

Nosotros, José Vicente Arias Ganchala con documento de identificación N° 1721140117 y Alex Javier García Patiño con documento de identificación N° 0803033562; manifestamos que:

Somos los autores y responsables del presente trabajo; y, autorizamos a que sin fines de lucro la Universidad Politécnica Salesiana pueda usar, difundir, reproducir o publicar de manera total o parcial el presente trabajo de titulación.

Quito, 04 de agosto del año 2022

Atentamente,



Arias Ganchala José Vicente
1721140117

García Patiño Alex Javier
0803033562

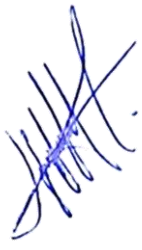
CERTIFICADO DE CESIÓN DE DERECHOS DE AUTOR DEL TRABAJO DE TITULACIÓN A LA UNIVERSIDAD POLITÉCNICA SALESIANA

Nosotros, José Vicente Arias Ganchala con documento de identificación N° 1721140117 y Alex Javier García Patiño con documento de identificación N° 0803033562, expresamos nuestra voluntad y por medio del presente documento cedemos a la Universidad Politécnica Salesiana la titularidad sobre los derechos patrimoniales en virtud de que somos autores del Proyecto Técnico: "Análisis comparativo entre el algoritmo A* y el algoritmo tipo insecto para la planeación de rutas aplicados sobre un robot móvil con ruedas", mismo que ha sido desarrollado para optar por el título de Ingeniero Electrónico, en la Universidad Politécnica Salesiana, quedando la Universidad facultada para ejercer plenamente los derechos cedidos anteriormente.

En concordancia con lo manifestado, suscribimos este documento en el momento que hacemos la entrega del trabajo final en formato digital a la Biblioteca de la Universidad Politécnica Salesiana

Quito, 04 de agosto del 2022

Atentamente,



Arias Ganchala José Vicente
1721140117

García Patiño Alex Javier
0803033562

CERTIFICADO DE DIRECCIÓN DEL TRABAJO DE TITULACIÓN

Yo, Carmen Johanna Celi Sánchez con documento de identificación N°1717437808, docente de la Universidad Politécnica Salesiana, declaro que bajo mi tutoría fue desarrollado el trabajo de titulación: ANÁLISIS COMPARATIVO ENTRE EL ALGORITMO A* Y EL ALGORITMO TIPO INSECTO PARA LA PLANEACIÓN DE RUTAS APLICADOS SOBRE UN ROBOT MÓVIL CON RUEDAS, realizado por José Vicente Arias Ganchala con documento de identificación N° 1721140117 y Alex Javier García Patiño con documento de identificación N° 0803033562, obteniendo como resultado final el trabajo de titulación bajo la opción Proyecto Técnico que cumple con todos los requisitos determinados por la Universidad Politécnica Salesiana.

Quito, 04 de agosto del 2022

Ing. Carmen Johanna Celi Sánchez, MSc.

1717437808

DEDICATORIA

Pasó el tiempo y cambiaron muchas veces los caminos, pero la meta era la misma hacerles sentir orgullosos a mis padres Vicente y Ruth, a mis hermanos Juan, Marcelo y Sebastián, y a mi hijo Matías, este esfuerzo diario es para y por ustedes.

Este logro no era posible sin el apoyo que han brindado en toda mi vida por eso quiero expresar mi cariño a ustedes papá Luchito y mamá Bachita y para mis tíos Luis, Emma y Mayra, gracias por enseñarme los valores del esfuerzo y la dedicación.

Siempre quise que este logro lo celebremos todos juntos papá Luchito y papá Vicente, pero la vida decidió que ustedes lo celebren desde el cielo, siempre sentí su apoyo y guía por lo cual estaré siempre agradecido.

José Vicente Arias Ganchala

Antes de nada, primero gracias a Dios por la salud y bendiciones que me ha brindado para poder conseguir los objetivos que me he propuesto.

Este trabajo de titulación se lo dedico con mucho amor a mis padres, hermanos y a mi hijo Carlitos por su ayuda y apoyo incondicional durante todo este tiempo en mi etapa universitaria.

Alex Javier García Patiño

AGRADECIMIENTO

Quiero agradecer en primer lugar a Dios por todas las bendiciones que me ha brindado, agradezco a mi familia cercana que ha estado en los buenos momentos y en los momentos más difíciles, a mis compadres, amigos y a El Nego.

Agradezco a la Universidad Politécnica Salesiana y a sus excelentes docentes en especial a la Mgtr. Johanna Celi que nos brindó la confianza y conocimiento para desarrollar nuestro proyecto.

José Vicente Arias Ganchala

Quiero agradecer a Dios, a la Universidad Politécnica Salesiana por los años de conocimiento y por las gratas experiencias que me brindaron. Agradezco a mi tutora la Mgtr. Johanna Celi que nos brindó su confianza, conocimiento y paciencia para poder culminar este proyecto de fin de carrera.

Agradezco a mi familia y amigos que estuvieron en las buenas y malas para darme una palabra de aliento y continuar, en especial al Mgtr. Junior Figueroa por guiarnos con conocimientos y sugerencias en ente proyecto técnico de titulación

Alex Javier García Patiño

CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

El presente proyecto de titulación tiene como objetivo el desarrollo y comparación de dos algoritmos de planificación de rutas conocidos A* (A- Star) y algoritmo tipo insecto, los cuales tienen por finalidad determinar la mejor trayectoria en un entorno conocido y reaccionar ante los cambios de luz ambiental; las tareas antes mencionadas son simples para los humanos, pero no tan sencillas para un robot móvil autónomo.

Los algoritmos antes mencionados permiten al robot móvil establecer las acciones de movimiento adecuadas que lo conducirán a la ubicación final deseada dentro de un laberinto cuyo entorno y distribución física es previamente conocida. Estos algoritmos de decisión y planificación de rutas ameritan el conocimiento de las restricciones cinemáticas del robot móvil el cual ha sido determinado mediante un modelamiento matemático.

A través de la implementación y ejecución de los dos algoritmos programados sobre el robot móvil, se pudo evidenciar y comparar los resultados en cuanto al tiempo y distancia recorrida por el robot, verificando que la solución generada por el algoritmo tipo Insecto conlleva mayor costo (tiempo y distancia) en comparación con la solución de ruta dada por el algoritmo A-Star.

Palabras claves: algoritmos A*, algoritmo tipo insecto, laberinto, LabVIEW, robot móvil**.**

ABSTRACT

The objective of this degree project is the development and comparison of two known route planning algorithms A* (A-Star) and an insect-type algorithm, which aim to determine the best trajectory in a known environment and react to changes in ambient light; The aforementioned tasks are simple for humans, but not so simple for an autonomous mobile robot.

The algorithms above will make it easier for the mobile robot to establish the appropriate movement actions that will lead it to the desired final location within a maze whose environment and physical distribution is previously known. These route planning and decision algorithms merit knowledge of the kinematic constraints of the mobile robot, which has been determined through mathematical modeling.

Through the implementation and execution of the two algorithms programmed on the mobile robot, it was possible to demonstrate and compare the results in terms of time and distance traveled by the robot, verifying that the solution generated by the Insect-type algorithm entails a higher cost (time and distance) compared to the route solution given by the A-Star algorithm.

Keywords: A* algorithms, insect-like algorithm, maze, LabVIEW, mobile robot.

INTRODUCCIÓN

Con la necesidad que se crea en el avance tecnológico y la búsqueda de la resolución de problemas para robots móviles se ve la necesidad de la implementar nuevos algoritmos que aporten con una solución que permitan el control en el camino de un punto a otro. Un robot utiliza sensores para percibir el entorno en que se encuentra hasta cierto grado de incertidumbre para poder construir o renovar su mapa, con esto poder buscar rutas adecuadas para la solución del problema.

En el Capítulo 1 se detalla el planteamiento del problema, así como la justificación y los objetivos propuestos en este proyecto.

El en Capítulo 2 se presentan temas teóricos en cuanto a hardware y software del robot móvil, en donde la temática principal corresponde a conceptos generales de los algoritmos implementados, posición y orientación del robot móvil, modelos cinemáticos y planeación de rutas.

El Capítulo 3 corresponde a la redacción del diseño e implementación del proyecto, el esquema general de conexión del proyecto, implementación de los algoritmos en el software de LabVIEW para la posición y direccionamiento del robot móvil y la planeación de rutas en el laberinto.

En el Capítulo 4 se exponen las conclusiones y recomendaciones generadas una vez culminado el proyecto, a través de la metodología experimental se realizó diferentes pruebas y análisis del funcionamiento de los dos algoritmos propuestos con lo cual se logró cumplir con los objetivos planteados en este proyecto de titulación.

CAPÍTULO 1
ANTECEDENTES

1.1. Planteamiento del problema

La navegación de un robot móvil es un proceso complejo porque depende de la información que se obtenga del entorno y de los diferentes factores que inciden en ella, las cuales presentan un riesgo en el desempeño del robot móvil.

En el contexto real casi todos los entornos son dinámicos, por dicha razón un robot móvil debe estar en la capacidad de reaccionar ante las variaciones del medio en el que se desenvuelve, con mucha más exactitud. Bajo esta premisa surge la necesidad del desarrollo de algoritmos de planeación que permitan a cualquier robot móvil desplazarse dentro de estos entornos dinámicos. En la actualidad el interés por este tipo de algoritmos ha ido incrementándose, dando lugar a la aparición de numerosos métodos de evasión de obstáculos y planificación local de trayectorias. (Robots Móviles Utilizando ROS, 2017).

La complejidad de la navegación autónoma de un robot móvil depende en gran parte de la información obtenida de su entorno. Así cuando se cuenta con la información necesaria y similar a la realidad, la creación de una trayectoria adecuada y su seguimiento son tareas que no presentan un alto grado de incertidumbre ya que en el mundo real la mayoría de los entornos no son estructurados. Por lo cual se necesita la capacidad del robot móvil para actuar ante las variaciones del medio que los rodea.

La planificación global tiene en cuenta la información proporcionada en el instante inicial del movimiento, es decir, el mapa del entorno y lo que vean los sensores en dicho instante de tiempo, con el fin de desarrollar una trayectoria que conduzca desde la posición inicial hasta la posición final. Por otro lado, la planificación local realiza modificaciones a la global para evitar obstáculos no previstos en el camino. El conjunto de dichas planificaciones da lugar a la trayectoria final. (Millán et al., 2016).

En virtud de lo señalado anteriormente, se busca implementar un algoritmo de planeación de rutas para la resolución de laberintos mediante robots móviles con ruedas, es decir, el uso de un laberinto físico para examinar el rendimiento y la eficiencia del algoritmo integrado. Se debe conseguir que el robot móvil navegue con cierto grado de inteligencia en un entorno conocido y que previamente se realicen simulaciones que permitan verificar de forma gradual y completa la funcionalidad del programa informático desarrollado.

Luego, el robot usará su algoritmo integrado para navegar a través del laberinto físico cuya trayectoria debe coincidir con el laberinto virtual donde se realizarán las simulaciones.

Por estas razones se realizará la comparación de dos algoritmos A* y algoritmo tipo insecto para verificar cuál de estos es el óptimo en las mismas condiciones expuestas para la navegación del robot móvil.

1.2. Justificación del proyecto

La planificación de rutas es utilizada para solucionar problemas en diferentes áreas, desde rutas espaciales sencillas hasta la elección de una secuencia de acciones adecuadas que son requeridas para cumplir un objetivo determinado. Debido a que el entorno donde se moviliza el robot no siempre es conocido, este tipo de planificación a menudo se restringe a los entornos previamente diseñados y los que se pueden describir con suficiente precisión. La planificación de rutas puede ser utilizadas en entornos completamente conocidos o de forma parcial, así como totalmente desconocidos en cuyo caso la información detectada define el movimiento requerido del robot.

Una de las aplicaciones más populares para realizar planeación de rutas con robots móviles con ruedas es el empleo de laberintos que consisten en determinar la mejor trayectoria para alcanzar la posición de salida. Según la información recopilada se han implementado hasta la actualidad diferentes algoritmos de control para la planeación de rutas, desde aquellos que emplean controladores básicos hasta los que utilizan principios de inteligencia artificial; la gran mayoría de ellos son verificados a nivel de simulación y unos pocos puestos en práctica sobre los robots físicos. Existen varios programas que incorporan librerías con sus respectivas funciones que permiten implementar varios tipos de algoritmos de planeación de rutas, pero en la mayoría de las situaciones el programador desconoce los fundamentos matemáticos que rigen el funcionamiento de dichos comandos.

Este proyecto de titulación busca analizar de forma detallada todos los fundamentos matemáticos que determinan el funcionamiento de un algoritmo de planeación de rutas en específico, para posteriormente traducirlo a lenguaje de programación de alto nivel, sin necesidad de emplear librerías proporcionadas por el software; más bien se trata de conocer a profundidad las variables de control y parámetros de configuración que se deben utilizar en el algoritmo de programación. Con esta visión se practican y combinan

los conocimientos adquiridos en las asignaturas de Teoría de Control, Sistemas Microprocesados, Programación, Instrumentación, Sensores, Robótica e Inteligencia Artificial desde un punto de vista matemático, analítico y práctico, demostrando los conocimientos teóricos aprendidos a lo largo de nuestra formación académica dentro de la carrera Electrónica.

1.3. Objetivos

1.3.1. Objetivo General

Comparar entre el algoritmo A* y el algoritmo tipo insecto en un robot móvil con ruedas para la planeación de rutas en un laberinto previamente conocido y determinar mediante programación la efectividad de ambos.

1.3.2. Objetivos específicos

- Investigar el estado del arte a través de la revisión bibliográfica para mejorar la utilidad de los algoritmos aplicados en robots móviles.
- Analizar los fundamentos matemáticos de los algoritmos A* y tipo insecto en la planeación de rutas de robots móviles para implementarlos mediante programación utilizando el software especializado.
- Simular el funcionamiento de los algoritmos en un software especializado para comprobar su funcionamiento.
- Implementar los algoritmos en el controlador del robot mediante una tarjeta de desarrollo para realizar pruebas experimentales en ruta.
- Verificar el funcionamiento de los algoritmos en cuestión para la comparación de su desempeño en la trayectoria de ruta mediante un laberinto de competencia.

CAPÍTULO 2
MARCO TEÓRICO

2. Introducción

En el presente capítulo se explicará el principio de funcionamiento de un robot móvil con ruedas en configuración diferencial el cual será empleado para la ejecución y verificación del algoritmo de planeación de rutas A*. Inicialmente se detallan los elementos (sensores, actuadores y controlador) con los que cuenta el kit de robótica Lego Mindstorms EV, los cuales formaran parte de la estructura mecánica del robot móvil. Posteriormente, se explicarán temas relacionados con el movimiento de un robot móvil con ruedas sobre el plano, por ejemplo: representación de la posición y orientación, cinemática directa e inversa, restricciones cinemáticas de las ruedas y el modelo cinemático obtenido para el robot diferencial que se va a emplear en este proyecto. Finalmente se abarcarán temas relacionados a la planeación de rutas de los robots móviles con ruedas y se analizará el principio de funcionamiento del algoritmo A* y tipo Insecto.

2.1 Kit de Robótica Lego Mindstorms EV3

Lego Mindstorms EV3 es un kit educativo de robótica diseñado y fabricado por la compañía Lego (ver Figura 2.1), que contiene componentes específicos de robótica, piezas estructurales y un entorno de programación amigable. Aunque a veces este kit es catalogado como un juego, en realidad sus prestaciones dentro del entorno educativo son muy elevadas y permiten creaciones muy prácticas.

El kit físico está compuesto por:

- El microprocesador (conocido también como ladrillo EV3).
- Los sensores (rotación, luz, color, ultrasonido, sonido, tacto).
- Los actuadores (servomotores).
- Elementos estructurales.

Así como el cerebro de cualquier ser vivo, recibe la información a través de los sentidos y gracias a ellos planifican los movimientos dentro del espacio donde se encuentren, una aplicación basada en esta plataforma recibe y efectúa acciones mediante los sensores y actuadores que interactúan con el ladrillo EV3 (cerebro). Con Lego Mindstorms EV3 se pueden hacer todo tipo de aplicaciones automatizadas (principalmente robóticas) que interactúen con el entorno.

Figura 2.1 Kit de robótica Lego Mindstorms EV3

Elementos que componen el kit educativo, Fuente (Lego.Com, 2019)

El kit básico Lego Mindstorms Education EV3 incluye el ladrillo programable, dos motores grandes, un motor mediano, cinco sensores (ultrasónico, táctiles, giroscópico, y de color), una batería recargable, cables para conectar los diferentes sensores y actuadores al ladrillo, un cable USB y 541 piezas de la serie Technic.

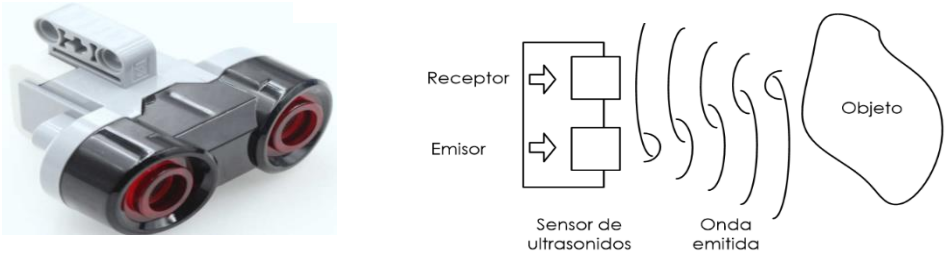
2.1.1. Sensores

De forma similar a los seres vivos, los sensores permiten obtener la información necesaria para que los robots interactúen con el entorno en donde se movilizan, estos elementos electrónicos les permiten saber dónde y cómo es el lugar en el que están.
Un sensor captura una magnitud física y debe ser capaz de transformarla en un cambio eléctrico que se pueda utilizar directamente, o sino a través de una etapa previa que lo condicione (amplificando, filtrando, etc.), para ser utilizada en el control del robot.

Sensor Ultrasónico

Este sensor es capaz de suministrar visión al robot, permitiéndole detectar objetos que se encuentran en el entono por el cual se moviliza. El principio de funcionamiento del sensor (Figura 2.2) consiste en emitir un sonido (onda ultrasónica) y medir el tiempo que la señal demora en retornar para luego determinar la distancia.

Figura 2.2 Sensor ultrasónico EV3

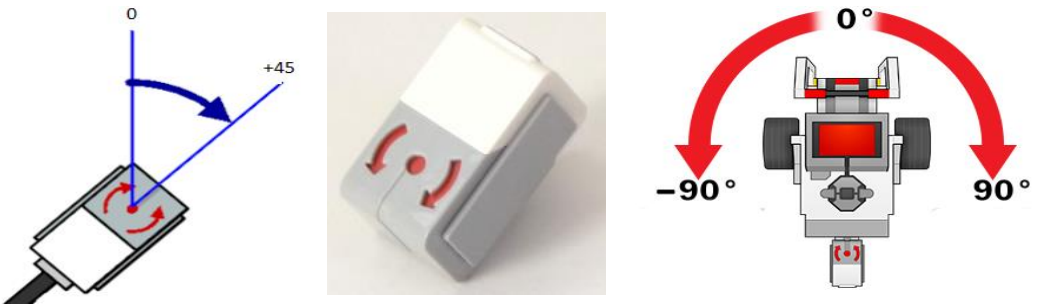


Vista física del sensor ultrasónico EV3, Fuente (Lego.Com, 2019)

Sensor Giroscópico

El sensor giroscópico o también llamado gyro sensor como se observa en la Figura 2.3, este sensor determina el movimiento de rotación en un solo eje. Si el sensor rota en la dirección de las flechas señaladas sobre la carcasa del mismo, este puede determinar la velocidad de rotación en °/s, con una velocidad máxima de giro de 440 °/s. Esta información de velocidad de rotación permite verificar cuando el robot está girando o cuando se ha volcado.

Figura 2.3 Sensor giroscópico EV3



Vista física del sensor giroscópico EV3, Fuente (*Girosensor EV3 - EsMindstorms*, 2019)

2.1.2. Actuadores

Los actuadores son los componentes que se acoplan a la estructura del robot y que permiten generar los movimientos del mismo. Gracias a estos elementos el robot será capaz de desplazarse, cerrar y abrir unas garras, girar, sujetar y transportar objetos, etc. El kit de Lego Mindstorms EV3 contiene tres motores DC que pueden ser acoplados al mecanismo.

Motores

El servomotor grande como se observa en la Figura 2.4 (a), es un motor DC que incorpora un tacómetro para un control preciso de la posición y la velocidad angular dentro de 1° de exactitud. El motor grande está optimizado para ser la base de conducción de los robots.

Figura 2.4 Motores EV3

Vista física del motor grande EV3, Fuente(Vizcaíno Juan, 2017)

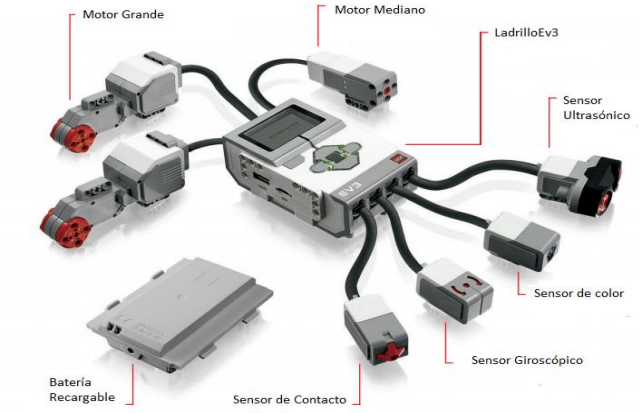
El sensor de rotación incorporado en el motor se lo puede emplear para alinear el movimiento del robot, realizar giros exactos y medir velocidades angulares de los ejes de giros.

El servomotor mediano como se observa en la Figura 2.4 (b), es especialmente utilizado para reducir la carga (torque) y en aplicaciones donde se requiera gran velocidad y cuando se precisan tiempos de respuesta rápidos y un tamaño más compacto en el diseño del robot. Este servomotor emplea la realimentación del tacómetro para un control de 1° de precisión, y tiene incorporado un sensor de rotación.

2.1.3. Bloque Ev3

Es el controlador del sistema, denominado también ladrillo inteligente EV3 (ver Figura 2.5). Este componente permite almacenar y gestionar los programas almacenados en su memoria, se podría decir que es el cerebro del robot, cuya CPU gestiona todos los procesos y se encarga de relacionar las entradas y salidas.

Figura 2.5 Bloque inteligente EV3



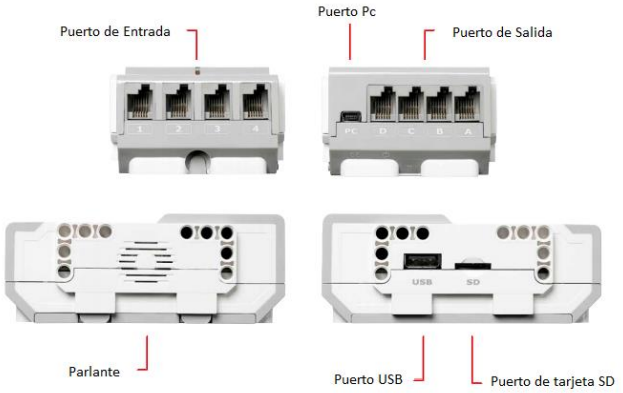
Conexión de los sensores y actuadores con el bloque EV3, Fuente (Vizcaíno Juan, 2017)

Componentes y Funcionamiento

La fuente de alimentación del bloque EV3 consiste en un conjunto de seis pilas AA (de 1,5 V cada una) o una batería recargable de 7.9 V, que alimenta a todos los componentes electicos y electrónicos del robot. La batería recargable es de corriente continua y tiene una capacidad de 2050 mAh.

Dispone de 4 entradas (0, 1, 2 y 3) donde se conectan los sensores y 4 salidas (A, B, C y D) donde van conectados los motores como se observa en la Figura 2.6. El programa diseñado en algún entorno de programación especifico instalado en el computador puede ser transmitido al bloque EV3 por medio de un cable USB (incluido en el kit) o mediante tecnología inalámbrica Bluetooth o WiFi.

Figura 2.6 Vista frontal y posterior del bloque EV3

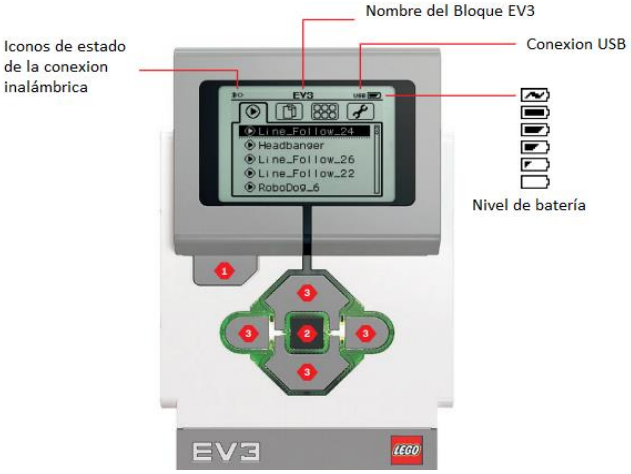


Ubicación física de entradas, salidas, parlantes y puertos de conexión del bloque EV3, Fuente (Vizcaíno Juan, 2017)

El bloque EV3 puede ser programado desde el propio ladrillo, o bien desde el computador. El bloque EV3 cuenta con una pantalla y varios botones que le permiten al usuario configurarlo y programarlo. La pantalla muestra lo que está sucediendo dentro del ladrillo EV3 y le permite utilizar la interfaz de bloque. También gracias a ella se puede agregar texto y datos numéricos o gráficos en su programación.

Los botones permiten navegar dentro de la interfaz de bloque EV3 y también se pueden utilizar como activadores programables. (ver Figura 2.7).

Figura 2.7 Vista superior del bloque EV3



Disposición de la pantalla y el bloque EV3, Fuente (Vizcaíno Juan, 2017)

El bloque EV3 se ilumina con diferentes colores por diversos motivos; las que rodean a los diferentes botones indican el estado actual del bloque EV3. Pueden ser de color verde, naranja o rojo y además intermitentes.

Programación

El software propio de Lego Mindstorms, es el EV3 de la compañía Lego, ofrece una excelente introducción a la programación, también se puede programar acciones más complejas que requieren un software diferente, por ejemplo, una opción es utilizar un lenguaje de programación basado en texto, como RobotC, que refleja un mejor estilo de programación y que es dominante en la industria informática. A continuación, se presenta una lista de algunas alternativas de software de terceros (tanto libres como de pago) más populares.

MakeCode: es una plataforma de programación en línea que puede controlar el bloque EV3 y otros dispositivos, como Cue y BBC micro bit. MakeCode usa bloques (como Scratch) o programación de JavaScript (texto).

Swift Playground: es un entorno exclusivo para iPad diseñado para ayudar a los usuarios a aprender a programar utilizando el lenguaje de programación Swift Playground, los usuarios pueden programar robots, recopilar información a través de sensores y realizar acciones a través de los motores.

EV3Python: ayuda a los usuarios familiarizados con la programación de Python a utilizar este lenguaje para controlar un robot EV3. La programación se hace con la ayuda de Microsoft Visual Studio Code.

RobotC: es un lenguaje de programación basado en C con un depurador de software totalmente integrado que admite una variedad de plataformas de hardware diferentes. RobotC incluye verificación de sintaxis en tiempo real, compilación y ayuda contextual y autocompletado de funciones y variables.

Scratch: es un entorno de programación visual muy popular que permite programar aplicaciones mediante bloques sin depender del entorno de programación de Lego. El proceso de integración de Lego Mindstorms EV3 con Scratch es relativamente sencillo y únicamente se necesita instalar Scratch Link, seleccionar la extensión adecuada y contar con una conexión bluetooth en el equipo.

EV3dev: no es un lenguaje de programación, sino un sistema operativo basado en Debian Linux que puede ejecutar casi todos los lenguajes de cualquier otra distribución de Linux, incluidos C ++, Node.js y Python. Es una versión modificada del sistema operativo Linux Debian Jessie.

OpenRoberta: es una plataforma basada en la nube, gratuita, que consiste en arrastrar y soltar elementos, para programar los robots basados en bloques EV3.

EV3 Basic: es un lenguaje de programación textual basado en el lenguaje de programación "Small Basic", el cual fue desarrollado por Microsoft como un lenguaje de codificación introductorio. EV3 Basic es compatible solo con PC con Windows.

LabVIEW: LabVIEW para Lego Mindstorms (LVLM) y LabVIEW para Educación (LV4E) son entornos de programación visual. El software EV3 fue construido en base a código LabVIEW, por lo que LVLM proporciona un gran paso para los usuarios que están

familiarizados con este lenguaje de programación y que están listos para desarrollar aplicaciones más complejas y versátiles.

Matlab: esta herramienta permite crear scripts en el entorno Matlab y ejecutarlos sobre el robot, así como crear modelos de diseño en el entorno de Simulink. Para establecer la comunicación con el bloque EV3 previamente debe descargarse el paquete de apoyo de Lego Mindstorms EV3 para Matlab.

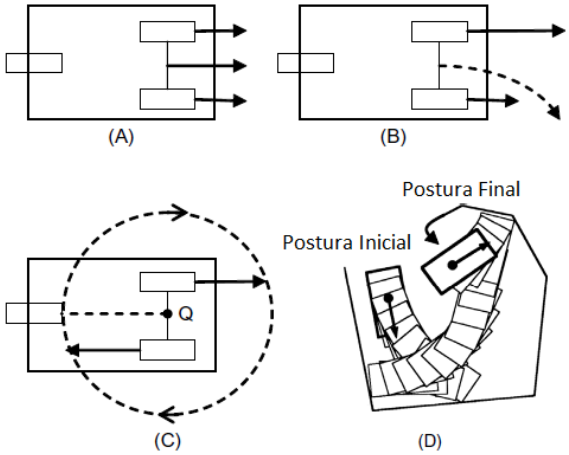
2.2. Cinemática del Robot Móvil Diferencial EV3

En robótica móvil la cinemática estudia la evolución de la postura, velocidad y aceleración, sin considerar las fuerzas o los torques. Claramente, las ecuaciones cinemáticas dependen de la geometría fija del robot respecto a un sistema de referencia global.

2.2.1. Robot con Accionamiento Diferencial

Un robot con accionamiento diferencial posee una cinemática sencilla en una estructura compuesta por dos ruedas fijas activas, accionadas de forma independiente cada una por un motor y una rueda pasiva tipo castor o esférica, con lo cual evita que la plataforma bascule mantiendo el equilibrio y la estabilidad. Con este tipo de sistema motriz el control de la trayectoria del robot se consigue ajustando la velocidad de giro de ambas ruedas. La diferencia de velocidad entre dos ruedas impulsa al robot en cualquier trayectoria y dirección requeridas, de ahí el nombre de accionamiento diferencial. Así, cuando las ruedas giran a la misma velocidad, el robot debe moverse hacia adelante (giran en el mismo sentido) o hacia atrás (giran en sentido opuesto) describiendo una línea recta. Si una rueda gira más rápido que la otra, el robot sigue una trayectoria curvilínea. Si las dos ruedas giran a la misma velocidad en direcciones opuestas, el robot gira alrededor del punto medio de las dos ruedas motrices, es decir, gira sobre sí mismo, así pues, se trata de una trayectoria con un radio de curvatura nulo. Esto permite al robot realizar maniobras sencillas e incluso el seguimimiento de trayectorias complejas. Los modos de locomoción mencionados se ilustran en la Figura 2.8.

Figura 2.8 Posibilidades de locomoción del accionamiento diferencial



(A) camino recto, (B) camino curvo, (C) camino circular, (D) maniobras sin obstáculos para pasar de una postura inicial a una final, (Tzafestas, 2014). Adaptado por: José Arias y Alex García.

En la Figura 2.9 se muestra el robot diferencial armado a partir de los elementos del kit de robótica de Lego Mindstorms EV3 y que será empleado para el desarrollo de este proyecto.

Figura 2.9 Robot móvil diferencial EV3

(A) (B)

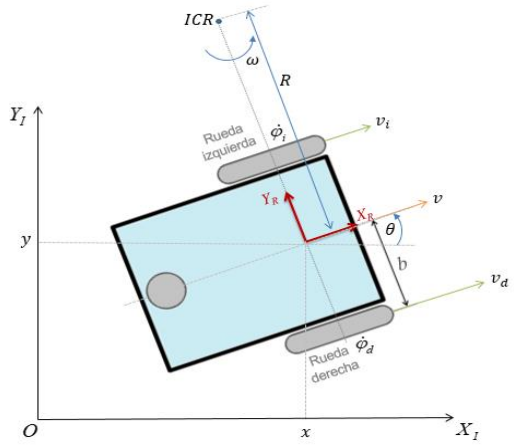
(A)Vista superior física del robot móvil con ruedas, (B) Vista frontal. Elaborado por: José Arias y Alex García.

2.2.2. Modelo Cinemático del Robot Diferencial EV3

El movimiento de un robot móvil con ruedas puede ser representado como una rotación alrededor de un punto (Figura 2.10), el cual se denomina Centro Instantáneo de Rotación (ICR) o a veces llamado Centro Instantáneo de Curvatura (ICC).

Este punto ICR corresponde al punto en donde se cruzan los ejes de todas las ruedas y sobre el cual el robot gira en un instante determinado y cuya posición con respecto al chasis del robot puede variar en el tiempo. Las restricciones cinemáticas y por ende el modelo cinemático del robot móvil diferencial se pueden obtener geométricamente usando el concepto de Centro Instantáneo de Rotación.

Figura 2.10 Geometría de un robot con accionamiento direccional



Análisis cinemático usando el concepto de centro instantáneo de rotación., Elaborado por: José Arias y Alex García.

De acuerdo con la Figura 2.10, las variables de entrada (control) son la velocidad de la rueda derecha v_d y la velocidad de la rueda izquierda v_i en donde para el análisis se asume que $v_d > v_i$. R es el radio instantáneo de la trayectoria de conducción del robot y que corresponde a la distancia entre el centro del chasis (punto medio entre las ruedas activas) y el punto ICR. En cada instancia de tiempo, ambas ruedas tienen la misma velocidad angular ω alrededor del ICR. Las velocidades lineales de cada rueda se obtienen a partir de:

$$v_d = \omega(R + b) \qquad v_i = \omega(R - b) \qquad \text{Ec (2.1)}$$

Resolviendo las dos ecuaciones anteriores es posible determinar ω y R en función de las velocidades lineales de las ruedas:

$$\omega = \frac{v_d - v_i}{2b} \qquad R = \frac{b(v_d + v_i)}{v_d - v_i} \qquad \text{Ec (2.2)}$$

La velocidad tangencial del robot se calcula entonces como:

$$v = \omega R = \frac{v_d + v_i}{2} \qquad \text{Ec (2.3)}$$

Las velocidades tangenciales de las ruedas son $v_i = r\dot{\varphi}_i$ y $v_d = r\dot{\varphi}_d$ donde $\dot{\varphi}_i$ y $\dot{\varphi}_d$ son las velocidades angulares izquierda y derecha de las ruedas alrededor de sus ejes, respectivamente. Reemplazando estos valores en las expresiones v y ω de las ecuaciones (2.2) y (2.3) se obtiene:

$$v = \frac{r\dot{\varphi}_d + r\dot{\varphi}_i}{2} = \frac{r}{2}\dot{\varphi}_d + \frac{r}{2}\dot{\varphi}_i \qquad \text{Ec (2.4)}$$

$$\omega = \frac{r\dot{\varphi}_d - r\dot{\varphi}_i}{2b} = \frac{r}{2b}\dot{\varphi}_d - \frac{r}{2b}\dot{\varphi}_i \qquad \text{Ec (2.5)}$$

Teniendo en cuenta las relaciones anteriores, la cinemática directa del robot (en coordenadas locales) se puede expresar en forma matricial como:

$$\begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix} = \begin{bmatrix} \frac{r}{2} & \frac{r}{2} \\ 0 & 0 \\ \frac{r}{2b} & -\frac{r}{2b} \end{bmatrix} \begin{bmatrix} \dot{\varphi}_d \\ \dot{\varphi}_i \end{bmatrix} \qquad \begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} \frac{r}{2} & \frac{r}{2} \\ \frac{r}{2b} & -\frac{r}{2b} \end{bmatrix} \begin{bmatrix} \dot{\varphi}_d \\ \dot{\varphi}_i \end{bmatrix} \qquad \text{Ec (2.6)}$$

Por lo tanto, la expresión para la velocidad lineal y angular del robot en base a la velocidad de las ruedas con respecto a su sistema de referencia local quedaría:

$$\dot{x} = v = \frac{r}{2}(\dot{\varphi}_d + \dot{\varphi}_i) \qquad \dot{\theta} = \omega = \frac{r}{2b}(\dot{\varphi}_d - \dot{\varphi}_i) \qquad \text{Ec (2.7)}$$

La cinemática directa del robot respecto al sistema inercial según la ecuación quedaría:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ 0 \\ \omega \end{bmatrix} = \begin{bmatrix} vcos\theta \\ vsin\theta \\ \omega \end{bmatrix} = \begin{bmatrix} cos\theta & 0 \\ sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}$$

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} cos\theta & 0 \\ sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{r}{2} & \frac{r}{2} \\ \frac{r}{2b} & -\frac{r}{2b} \end{bmatrix} \begin{bmatrix} \dot{\varphi}_d \\ \dot{\varphi}_i \end{bmatrix} = \frac{r}{2} \begin{bmatrix} cos\theta & cos\theta \\ sin\theta & sin\theta \\ \frac{1}{b} & -\frac{1}{b} \end{bmatrix} \begin{bmatrix} \dot{\varphi}_d \\ \dot{\varphi}_i \end{bmatrix} \qquad \text{Ec (2.8)}$$

Las velocidades respecto al sistema inercial y en función de las velocidades lineales y angulares locales del robot quedan:

$$\dot{x} = v\,cos\theta \qquad \dot{y} = v\,sin\theta \qquad \dot{\theta} = \omega \qquad \text{Ec (2.9)}$$

$$\dot{x} = \frac{r}{2}(\dot{\varphi}_d cos\theta + \dot{\varphi}_i cos\theta) \quad \dot{y} = \frac{r}{2}(\dot{\varphi}_d sin\theta + \dot{\varphi}_i sin\theta) \quad \dot{\theta} = \frac{r}{2b}(\dot{\varphi}_d - \dot{\varphi}_i) \quad \text{Ec (2.10)}$$

Según las dimensiones físicas del robot móvil diferencial EV3 implementado en este proyecto, $r = 2.3\ cm = 0.023\ m$ y $b = 6\ cm = 0.06\ m$ .Remplazando estos parámetros constantes en los términos de la ecuación 2.14, se obtiene las ecuaciones cinemáticas finales dadas por:

$$\dot{x} = 0.0115(\dot{\varphi}_d cos\theta + \dot{\varphi}_i cos\theta) \qquad \dot{y} = 0.0115(\dot{\varphi}_d sin\theta + \dot{\varphi}_i sin\theta)$$
$$\dot{\theta} = 0.1916(\dot{\varphi}_d - \dot{\varphi}_i) \qquad \text{Ec (2.11)}$$

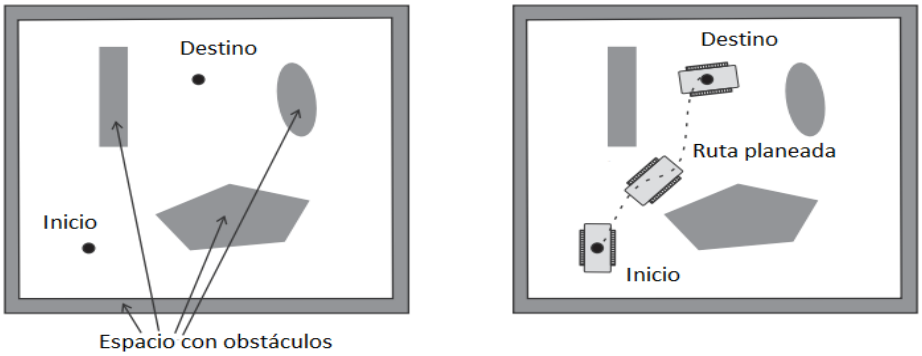
2.3. Planeación de Rutas de Robot Móviles

La planeación de rutas es una capacidad general incorporada en todo tipo de robots, se ocupa de la determinación de cómo un robot se moverá y maniobrará en un espacio de trabajo o en el entorno para lograr sus objetivos. El problema de la planeación de rutas implica calcular una ruta sin colisiones entre una posición inicial y una posición final, además de evitar obstáculos, el robot también debe satisfacer algunos otros requisitos u optimizar ciertos criterios de rendimiento.

2.3.1. Entorno del Robot

El entorno en el que se desplaza el robot podría ser un espacio libre y/o un espacio ocupado por los obstáculos como se muestra en la Figura 2.11, en donde la postura de inicio y final están ubicadas en el espacio libre del entorno por el cual se movilizará el robot. La planeación de rutas se distingue de acuerdo con el conocimiento disponible sobre el entorno, es decir, entornos completamente conocidos (estructurados), entornos parcialmente conocidos y entornos completamente desconocidos (no estructurados). En la mayoría de los casos prácticos, el entorno sólo se conoce parcialmente, donde el robot antes de la planeación de ruta y la navegación ya tiene conocimiento de algunas áreas dentro del espacio de trabajo, es decir, áreas locales que pueden presentar problemas mínimos.

Figura 2.11 Entorno de un robot con obstáculos



Izquierda: entorno con obstáculos y configuraciones de inicio y meta. Derecha: uno de los muchos caminos posibles desde el principio hasta la configuración del objetivo. (Gregor Klancar Andrej Zdesar Saso Blazic Igor Skrjanc, 2017) Adaptado por: José Arias y Alex García.

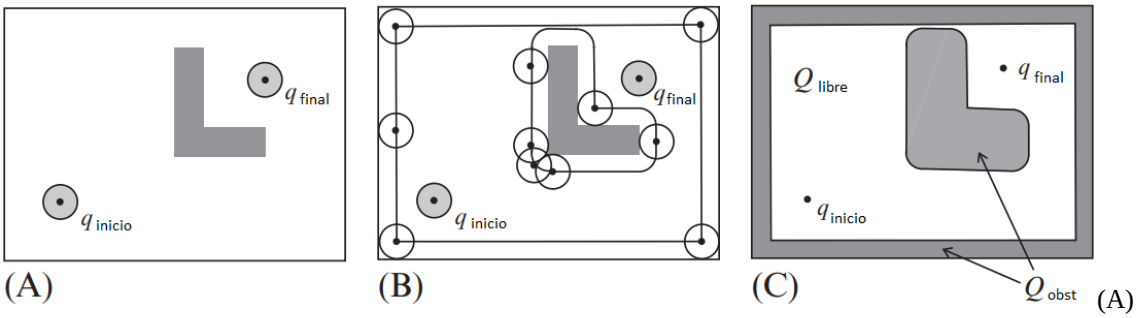
2.3.2. Configuración y Espacio de Configuración

La configuración es el estado del sistema en el espacio del entorno abarcado en n dimensiones. El vector de estado se describe mediante $\boldsymbol{q} = [q_1, \dots, q_n]^T$, en donde el estado $\boldsymbol{q}$ es un punto en el espacio n-dimensional llamado espacio de configuración Q, que presenta todas las configuraciones posibles del sistema móvil según su modelo cinemático. Parte del espacio de configuración que ocupan los obstáculos O_i es denotado por $Q_{obst} = \cup_i O_i$, entonces la parte libre del entorno sin obstáculos se define como:

$$Q_{libre} = Q - Q_{obst} \qquad \text{Ec (2.12)}$$

Q_{libre} por lo tanto, contiene el espacio donde el sistema móvil puede planificar su movimiento. Suponga que hay un robot circular que solo puede ejecutar movimientos traslacionales en el plano (es decir, tiene dos grados de libertad $\boldsymbol{q} = [x \ y]^T$), luego su configuración se puede definir mediante un punto (x, y) que representa el centro del robot. En este caso el espacio de configuración Q se determina moviendo el robot alrededor de los obstáculos manteniéndolo en contacto con el mismo, como se muestra en la Figura 2.12. Al hacerlo, el punto central del robot describe el límite entre Q_{libre} y Q_{obst} , en otras palabras, los obstáculos se agrandan por la dimensión conocida del robot (radio) para ver al robot como si se tratase de un punto en el problema de planificación de ruta.

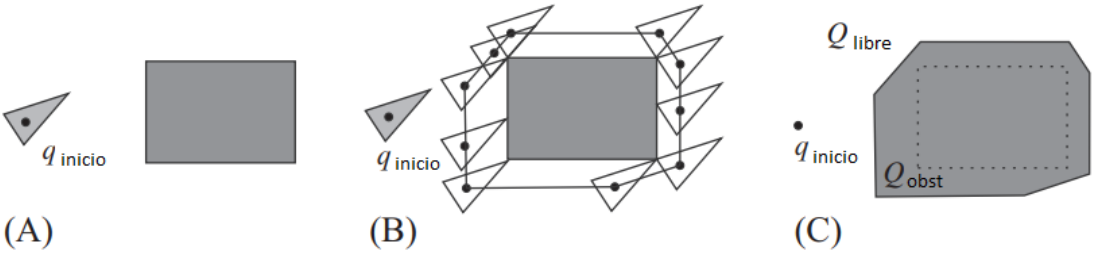
Figura 2.12 Robot circular en el entorno que contiene un obstáculo



Configuraciones de inicio y meta marcadas, (B) definición del espacio de configuración, (C) entorno transformado de (A) a un espacio de configuración. (Gregor Klancar Andrej Zdesar Saso Blazic Igor Skrjanc, 2017) y Adaptado por: José Arias y Alex García.

Un ejemplo adicional de espacio de configuración para un robot triangular y un obstáculo rectangular se muestra en la Figura 2.13; en donde el robot solo puede moverse en las direcciones x e y ($\boldsymbol{q} = [x \ \ y]^T$).

Figura 2.13 Robot triangular en el espacio de configuración



(A) Obstáculo rectangular y robot triangular con un punto que define su configuración q, (B) determinación del espacio de configuración, (C) configuración libre Q_{libre} y espacio ocupado por el obstáculo Q_{obst} . (Gregor Klancar Andrej Zdesar Saso Blazic Igor Skrjanc, 2017) Adaptado por: José Arias y Alex García.

Si el robot de la Figura 2.13 también pudiera girar, su configuración tendría tres dimensiones $\boldsymbol{q} = [x \ y \ \varphi]^T$ y el espacio de configuración sería más complejo. Por simplificación se podría determinar el espacio de configuración usando un círculo que delinea la forma del robot y tiene su centro en el punto central del robot. La configuración obtenida Q_{libre} es entonces más pequeña que la verdadera configuración libre porque el área del círculo delineada es más grande que el área del robot, sin embargo, esto simplifica significativamente el problema de planificación de rutas.

2.3.3. Presentación del Entorno mediante Descomposición de Celdas

Antes de la planificación de la ruta, el entorno debe presentarse de una forma matemática que sea adecuada para el procesamiento en los algoritmos de búsqueda de rutas. Los mapas de navegación del robot que se utilizan para representar el entorno pueden ser una descripción geométrica continua, un mapa geométrico basado en la descomposición o un mapa topológico. Estas alternativas se pueden realizar mediante cuatro metodologías generales:

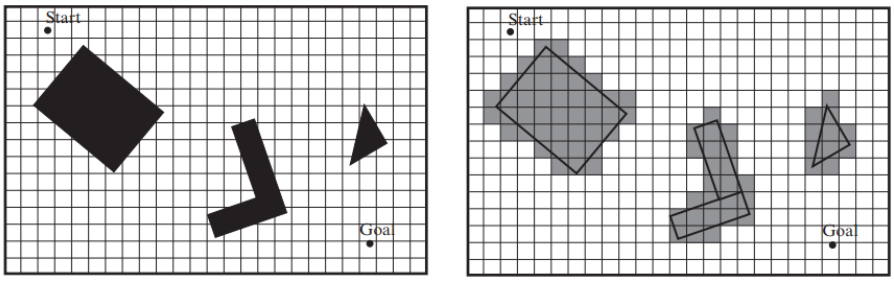
- Representación gráfica de transición de estados
- Mapas viales
- Descomposición de celdas
- Campos potenciales
- Histogramas de campo vectorial

En este apartado solo se hará referencia a la representación del entorno mediante el de descomposición de celdas puesto que es el que se utilizará para la implementación del algoritmo A*.

Descomposición Aproximada de Celdas

La descomposición del entorno en celdas es aproximada cuando es posible que ciertas celdas contengan el espacio de configuración libre. Todas las celdas que contengan al menos parte del obstáculo son por lo tanto normalmente marcadas como ocupadas, mientras que las celdas restantes son marcadas como libres. Si la descomposición de las celdas es del mismo tamaño, entonces se obtiene una cuadrícula de ocupación (Figura 2.14).

Figura 2.14 Descomposición aproximada de celdas



Izquierda: colocando una cuadrícula de celdas iguales en el entorno. Derecha: marcando las celdas como libres u ocupadas (celdas sombreadas). (Gregor Klancar Andrej Zdesar Saso Blazic Igor Skrjanc, 2017) y Adaptado por: José Arias y Alex García.

El centro de cada celda (en la Figura 2.14, la celda libre está marcada con blanco) se presenta como un nodo en un gráfico. Las transiciones entre las celdas están en 4 u 8 direcciones si se permiten transiciones diagonales. El principal inconveniente de este enfoque es el uso de memoria, que aumenta al incrementar el tamaño del entorno o al reducir el tamaño de la celda.

2.4. Algoritmos de Planeación de Rutas

Estos algoritmos buscan en un gráfico un camino comenzando en un vértice y explorando los nodos adyacentes hasta que se alcanza el nodo de destino, generalmente con la intención de encontrar la ruta más corta y barata.

En este proyecto de titulación se analizará el principio de funcionamiento de dos algoritmos de planeación de rutas, uno considerado un algoritmo versus el otro, para posteriormente comparar los resultados obtenidos en la solución de la búsqueda del camino óptimo dentro de un laberinto. Estos algoritmos son los de tipo Insecto y el A*, los cuales serán explicados a continuación.

2.4.1. Algoritmo Tipo Insecto

Los algoritmos tipo Insecto son los algoritmos de planeación de rutas más simples que asumen solo el conocimiento local y no necesitan un mapa del entorno. Por lo tanto, son apropiados en situaciones en las que se desconoce un mapa del entorno o está cambiando rápidamente y también cuando el controlador del robot móvil tiene una potencia computacional muy limitada. Estos algoritmos utilizan la información local suministrada por sus sensores y la información global del punto objetivo. Su funcionamiento consta de

dos comportamientos simples: movimiento en línea recta hacia la meta y seguimiento del límite de obstáculos.

Los robots móviles que utilizan estos algoritmos pueden evitar obstáculos y avanzar hacia la meta. Estos algoritmos requieren poco uso de memoria. Existen 3 versiones básicas del algoritmo tipo Insecto, a continuación, se describe el tipo cero debido a que es éste el implementado en el proyecto.

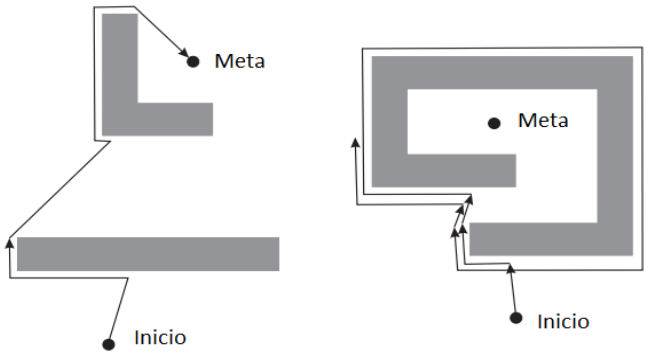
Tipo Insecto 0

Un algoritmo Insecto 0 tiene dos comportamientos básicos:

1. Moverse hacia el objetivo hasta que se detecte un obstáculo o se alcance la meta.
2. Si se detecta un obstáculo, gire a la izquierda (o a la derecha, pero siempre en la misma dirección) y siga el contorno del obstáculo hasta que vuelva a ser posible el movimiento en línea recta hacia la meta.

En la figura 2.15 se ofrece un ejemplo del rendimiento del algoritmo Insecto 0.

Figura 2.15 Ejemplo de búsqueda de ruta para el algoritmo Insecto 0



El algoritmo tipo Insecto 0 encuentra con éxito un camino hacia la meta en el entorno de la izquierda, mientras que no tiene éxito en el entorno de la derecha. (Gregor Klancar Andrej Zdesar Saso Blazic Igor Skrjanc, 2017) y Adaptado por: José Arias y Alex García.

2.4.2. Algoritmo A*

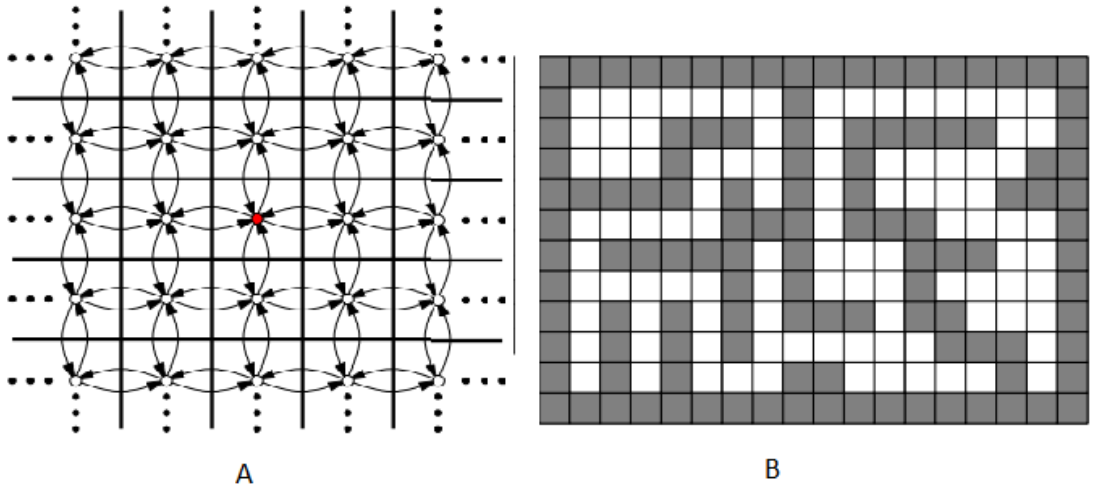
A* (pronunciado “A Star”) es un algoritmo de búsqueda informada porque incluye información adicional o heurística, es decir, que utiliza el conocimiento específico del problema, la heurística es la estimación del costo de la ruta desde el nodo actual hasta el nodo meta que corresponde a la parte del grafo de árbol que aún no se ha explorado.

Para cada nodo, el algoritmo calcula la función heurística que es la estimación del costo de la ruta desde este nodo hasta el nodo meta, y que es denominado “costo a meta”. Esta

función heurística puede ser la distancia Euclidiana o la distancia de Manhattan (suma de movimientos verticales y horizontales) desde el nodo actual hasta el nodo meta. La heurística también puede calcularse mediante alguna otra función apropiada.

El algoritmo de planificación de ruta A* se utiliza en muchas aplicaciones, desde la implementación de videojuegos hasta el desarrollo de software de conducción y control de vehículos autónomos. Es mejor comprender el algoritmo conceptualmente para que se pueda utilizar de forma más eficaz. En este apartado, se explicarán los conceptos detrás del algoritmo A*, iniciando con el análisis de un ejemplo simple y posteriormente se resumirán los pasos que se aplicarán para el desarrollo de este.

Figura 2.16 Ejemplo de representación gráfica del entorno

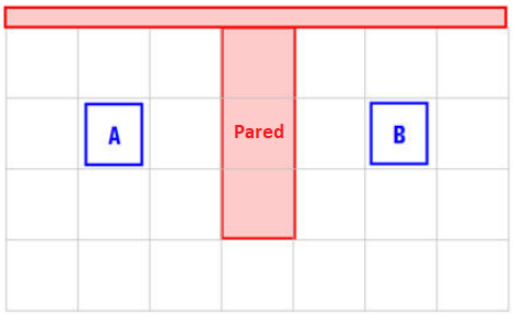


Es quema de transición de estados para un ejemplo que involucra movilizarse sobre un suelo infinito de baldosas, B) Problema de planificación de rutas basado en la exploración de un laberinto. Fuente: (An Introduction to A* Path Planning (Using LabVIEW) - NI Community, 2017) y adaptado por: José Arias y Alex García.

Ejemplo de Aplicación del Algoritmo A*

Se inicia la explicación del algoritmo A* con un ejemplo simple para indicar los por menores del mismo. Se asumirá que el robot debe movilizarse desde el punto A al punto B de la manera más eficientemente posible, pero es notorio que una pared los separa, tal y como se observa en la Figura 2.17.

Figura 2.17 Situación inicial del entorno



División del área de búsqueda en una cuadricula 2D. Fuente: (An Introduction to A* Path Planning (Using LabVIEW) - NI Community, 2017) y adaptado por: José Arias y Alex García.

Se ha dividido la zona de búsqueda en una cuadrícula para simplificar el análisis del entorno, este método en particular disminuye el área de búsqueda a una sencilla matriz bidimensional, en donde cada elemento de dicha matriz constituye uno de los recuadros de la cuadrícula y su estado se registra como transitable o inaccesible. La ruta se determina computando qué recuadros se deben considerar para movilizarse del punto A al punto B, trasladándose desde el punto medio de un recuadro hasta el punto medio del otro. A demás, es viable fraccionar el área de búsqueda en otras formas geométricas.

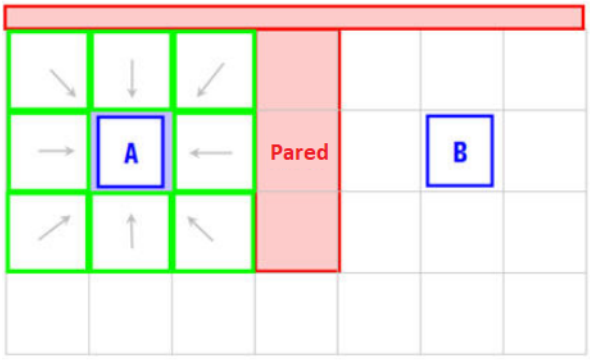
Inicio de la Búsqueda

Luego de haber simplificado el área de búsqueda en un número aceptables de nodos, el paso siguiente es realizar una búsqueda para determinar la trayectoria más corta. Esta acción se la debe realizar iniciando en el nodo A, verificando los nodos contiguos, valorando el mejor paso siguiente y continuando hasta hallar el nodo meta B.

El proceso del algoritmo A* lleva a cabo el inicio de la búsqueda con la creación de dos listas. La "*lista abierta*" que contiene los nodos considerados como candidatos potenciales para el próximo paso, y que esencialmente es una lista de los recuadros que precisan ser verificados. La *"lista cerrada"* la cual contiene los nodos que ya han sido evaluados y que no necesitan considerarse nuevamente. Para iniciar el algoritmo, se añade el nodo inicial a la lista cerrada y luego se debe observar todos los nodos cercanos al nodo de partida A y agregar los nodos transitables o alcanzables a la lista abierta mientras se ignora los nodos inaccesibles (cuadrados que representan las paredes). En este ejemplo particular, todos los nodos cercanos son transitables, por lo cual se va a agregar ocho nuevos nodos a la lista abierta. Todos estos nodos deben ser referenciados a partir de un nodo principal conocido como el *"nodo padre"* que en este caso corresponde al nodo A.

En la Figura 2.18, se puede observar que los nodos incluidos en la lista abierta están delineados con recuadros de color verde y el nodo padre A es identificado mediante las flechas grises.

Figura 2.18 Evaluación del entorno primera parte



Asignación de nodos a la lista abierta y asignación de un nodo padre. Fuente: (An Introduction to A* Path Planning (Using LabVIEW) - NI Community, 2017) y adaptado por: José Arias y Alex García.

Puntuación y Determinación de Ruta

El algoritmo A* emplea la información del gráfico para determinar el costo de la ruta que va desde actual nodo padre al siguiente nodo n (a este costo se le denominará G) y el costo estimado de la mejor ruta que va desde el nodo n al nodo meta B (a este costo se le denominará H). La suma de estos dos costos permite encontrar el costo estimado (F) de la solución más viable que pasa por el nodo n, considerando un criterio seleccionado. Para este ejemplo, el criterio podría ser la distancia entre nodos, puesto que es el único dato proporcionado por ahora. La clave para determinar qué nodos usar al calcular la ruta es la siguiente ecuación:

$$F = G + H \qquad \text{Ec (2.13)}$$

en donde:

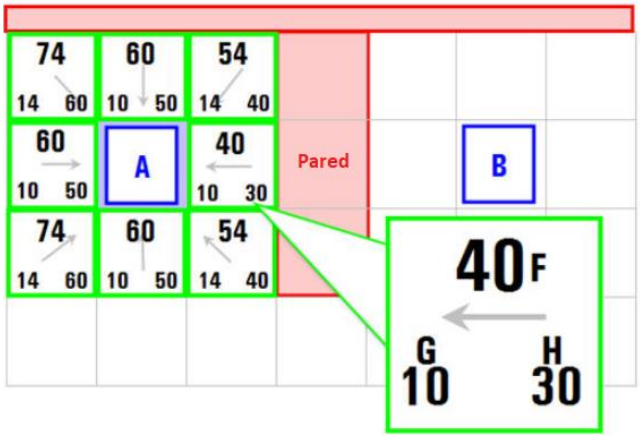
- G es el costo de movimiento para pasar del nodo inicial A ha un cierto nodo n dado en la cuadrícula.
- H es el costo de movimiento estimado para moverse desde el nodo actual n dado en la cuadrícula hasta el nodo meta B. Esto a menudo se conoce como la heurística.

La ruta se va formando por una inspección continua de los nodos incluidos en la lista abierta y seleccionando el nodo con el valor F más bajo. Para este ejemplo, se asignará un costo de $G = 10$ a cada nodo (recuadro) que requiera ejecutar un movimiento

horizontal o vertical, y un costo de $G = 14$ para un movimiento en diagonal. Se ha considerado estos valores debido a que la distancia real para moverse en diagonal es la raíz cuadrada de 2 o aproximadamente 1.414 veces el costo de moverse horizontal o verticalmente, de manera que la proporción es correcta y los números enteros son más fáciles de emplear.

En la Figura 2.19, se muestran los resultados de la puntuación de la primera fase de nodos que se encuentran alrededor del nodo padre A. El nodo agrandado (ubicado a la derecha del recuadro A) posee un valor de $G = 10$, puesto que sólo requiere movilizarse un nodo desde el nodo inicial A en sentido horizontal. Los nodos inmediatamente por encima, por debajo y a la izquierda del nodo inicial A, poseen todos las mismas puntuaciones de $G = 10$, y los nodos diagonales poseen puntuaciones de $G = 14$.

Figura 2.19 Evaluación del entorno segunda parte



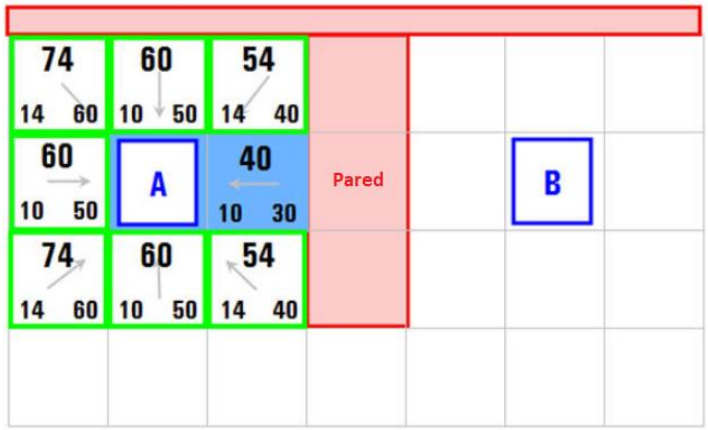
Asignación de los valores de los costos *G* y *H*. Fuente: ((An Introduction to A* Path Planning (Using LabVIEW) - NI Community, 2017)) y adaptado por: José Arias y Alex García.

El valor del costo *H* se puede estimar de diversas formas. El método que se ha utilizado en este ejemplo se conoce como el método de Manhattan, el cual determina el número total de nodos desplazados vertical y horizontalmente para llegar al nodo meta B desde el nodo actual n analizado, omitiendo el empleo de movimientos diagonales e ignorando cualquier obstáculo que pueda localizarse en la trayectoria. Luego se multiplica este valor por 10, que corresponde al costo por mover un nodo de forma horizontal o verticalmente. Por ejemplo, haciendo uso de la estimación de distancias de Manhattan, el nodo ubicado a la derecha del recuadro A necesita recorrer tres espacios de forma horizontal para alcanzar al nodo meta B, luego dicho valor se multiplica por 10 y se consigue un valor de

30 en *H*, como se muestra en la Figura 2.20. Los nodos por debajo, por encima y a la izquierda del actual nodo padre A, requieren recorrer 5 nodos de distancia para alcanzar al nodo meta B, por lo tanto, se les asigna una puntuación de $H = 50$. Utilizando el mismo procedimiento se determinan los valores de costo *H* para los restantes nodos.

Posteriormente, se selecciona el nodo de puntuación *F* más bajo de los localizados en la lista abierta, a continuación, se quita este nodo de la lista abierta y se lo agrega a la lista cerrada, luego este nodo se convierte en el nuevo nodo padre. Todos los nodos transitables y cercanos a este nuevo nodo padre, deben agregarse a la lista abierta. Aquellos nodos que ya se ubicaban en la lista cerrada no se tomaran en cuenta en el proceso de agregar nodos a la lista abierta, y todos los nodos que ahora constituyen la lista abierta deben volver a evaluarse para explorar una ruta nueva, como se muestra en la Figura 2.21.

Figura 2.20 Evaluación del entorno tercera parte



Evaluación del nodo con menor costo F a partir del nodo padre A. Fuente: (An Introduction to A* Path Planning (Using LabVIEW) - NI Community, 2017) y adaptado por: José Arias y Alex García.

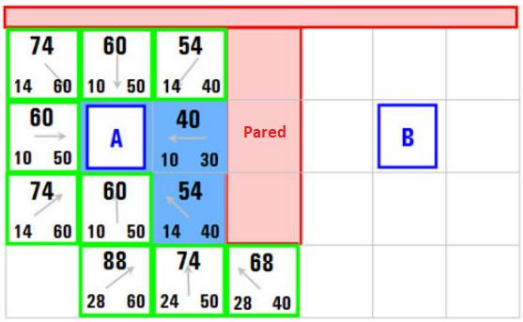
En la Figura 2.21, se puede apreciar que, de los nueve nodos iniciales ahora solo quedan ocho en la lista abierta (delineados en verde), una vez que el nodo A haya sido incluido en la lista cerrada. De todos ellos, el nodo con el costo *F* más bajo es el que está seguidamente a la derecha del nodo inicial A, con un valor $F = 40$, por lo cual este nodo será designado como el nuevo nodo padre. A continuación, se quita este nodo de la lista abierta y se lo agrega a la lista cerrada (resaltado en azul). Posteriormente se verifican todos sus nodos cercanos; según se observa en la Figura 2.21 los nodos que están inmediatamente a la derecha de este cuadrado resaltado forman parte de la pared así que son ignorados y el nodo que está a la izquierda inmediata es el nodo inicial A, pero debido a que este nodo ya se encuentra localizado en la lista cerrada también debe ser ignorado.

Los últimos cuatro nodos ya se ubican en la lista abierta, por lo cual se debe verificar si cualquiera de los caminos a esos nodos (comenzando desde el nodo inicial A) es más apropiado que la ruta que se puede establecer con el actual nodo padre seleccionado, por lo cual se utiliza las puntuaciones *G* como punto de referencia. Observe el nodo que está justo encima del actual nodo padre, su puntaje *G* actual es 14; si se pretende llegar hasta allí desde el nodo A pasando por el actual nodo padre, el costo *G* sería igual a 10, que corresponde al puntaje *G* para alcanzar al actual nodo padre, más 10 de un movimiento vertical justo encima del actual nodo padre. Una puntuación de $G = 20$ es superior a una de 14, por lo que dicha situación no corresponde a un mejor camino; todo lo mencionado debería tener sentido si se observa el gráfico de la Figura 2.21. Como es notorio es más corto el camino a ese nodo desde el nodo inicial A simplemente moviendo un nodo en diagonal, en vez de mover horizontalmente un nodo y luego verticalmente otro nodo.

Si se replica este método para los otros tres nodos cercanos que ya están en la lista abierta, se pone en manifiesto que ninguna de las rutas a mejorado al pasar por el actual nodo padre, debido a aquello no se modifica nada y se culmina la valoración de este nodo padre. Entonces, como ya se han verificado todos los nodos cercanos respecto al actual nodo padre, se requiere saltar al siguiente mejor nodo en la lista abierta, que corresponde al nodo localizado debajo del actual nodo padre con un costo $F = 54$. Este nodo seleccionado debe ser retirado de la lista abierta y movido a la lista cerrada, también este nodo se convertirá en el nuevo nodo padre. Este procedimiento se repetirá para todos los nuevos nodos de la lista abierta y se obtendrán los resultados como los mostrados en la Figura 2.22.

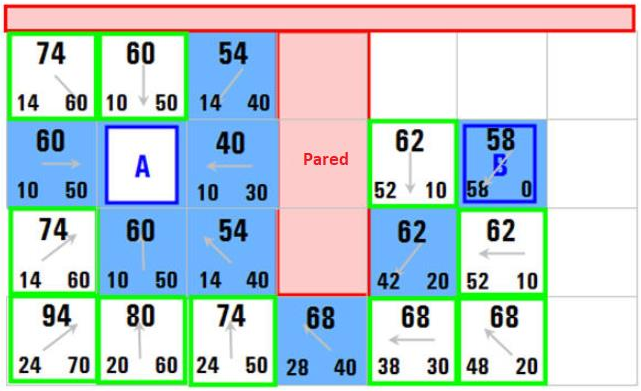
Para obtener el camino completo, se debe continuar repitiendo este procedimiento hasta que el nodo meta B haya sido añadido a la lista cerrada, posterior a ello el mapa de evaluación de la ruta tendría una apariencia como la mostrada en la Figura 2.22.

Figura 2.21 Evaluación del entorno cuarta parte



Siguiente nodo padre para el proceso de evaluación. Fuente: (An Introduction to A* Path Planning (Using LabVIEW) - NI Community, 2017) y adaptado por: José Arias y Alex García.

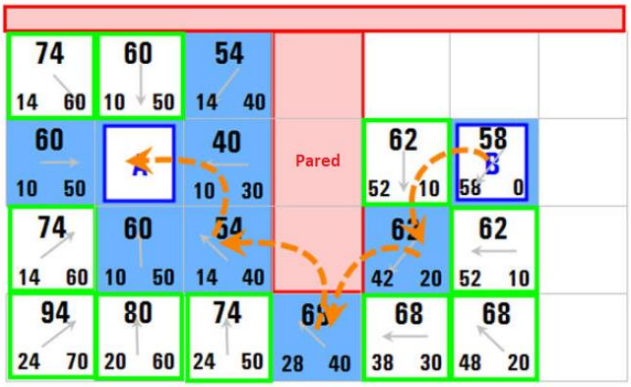
Figura 2.22 Evaluación del entorno quinta parte



Proceso completo de evaluación de nodos. Fuente: (An Introduction to A* Path Planning (Using LabVIEW) - NI Community, 2017) y adaptado por: José Arias y Alex García.

Ahora, para determinar la mejor ruta, simplemente se parte del nodo meta B y se continua hacia atrás desplazándose desde un nodo a su nodo padre; en la gráfica de la Figura 2.22, se observa este procedimiento visualizando el sentido de las flechas. Ello eventualmente conllevará a regresar al nodo inicial A, como se aprecia en la Figura 2.23. Puede verificarse entonces que este camino corresponde a la ruta de menor costo obtenida desde el nodo inicial A hasta el nodo meta B.

Figura 2.23 Evaluación del entorno parte final



Mejor ruta determinada desde el nodo A al nodo B. Fuente:(An Introduction to A* Path Planning (Using LabVIEW) - NI Community, 2017) y adaptado por: José Arias y Alex García.

Resumen de la Secuencia de Ejecución del Algoritmo A*

Como se ha analizado en el ejemplo anterior, durante la ejecución del algoritmo A* para cada nodo, el costo de la ruta completa *F* se calcula a partir del costo *G* y el costo *H*. Durante la búsqueda de la ruta, también se utilizan la lista de nodos abiertos y la lista de nodos cerrados.

La operación del algoritmo es la siguiente. Al principio, la lista abierta contiene solo el nodo inicial, que tiene un costo $G = 0$ y no tiene conexión con un nodo anterior. Luego se repiten los siguientes pasos para cada nodo padre seleccionado:

1. De la lista abierta se toma este primer nodo y se lo agrega a la lista cerrada, este nodo es conocido como actual nodo padre. Posteriormente se agregan a la lista abierta todos los nodos adyacentes transitables desde el actual nodo padre excluyendo aquellos que forman parte de las paredes. La lista abierta se ordena de acuerdo con el costo total *F* de la ruta en orden creciente, donde el primer nodo es el que tiene el menor costo *F*.
2. Para todos los nodos adyacentes n a los que se puede acceder desde el actual nodo padre, se calcula lo siguiente:

- El costo *G* para pasar desde el nodo inicial a un cierto nodo n dado en la cuadrícula.
- El costo *H* para moverse desde el nodo actual n hasta el nodo meta.
- El costo total *F* como la suma del costo *G* y el costo *H*.

3. Para cada uno de esos nodos adyacentes que aún no tienen almacenado el costo *G*, se almacenan los valores de la conexión (costo *H* y *F*) desde el actual nodo padre.

4. Si en el paso anterior algún nodo adyacente ya tenía asignado los valores de costos de alguna iteración anterior, entonces se comparan ambos costos *G* (el actual y el anterior) y se almacena el de menor valor, junto con la conexión correspondiente y el costo total *F*.
5. Los nodos cuyos valores de costo se han calculado por primera vez se agregan a la lista abierta. Los nodos que ya han estado en la lista abierta y se actualizaron se mantienen en la lista abierta. Los nodos que se actualizaron y estaban en la lista cerrada se mueven a la lista abierta. La lista abierta se ordena según el costo total *F* en orden creciente. El actual nodo padre se transfiere a la lista cerrada.

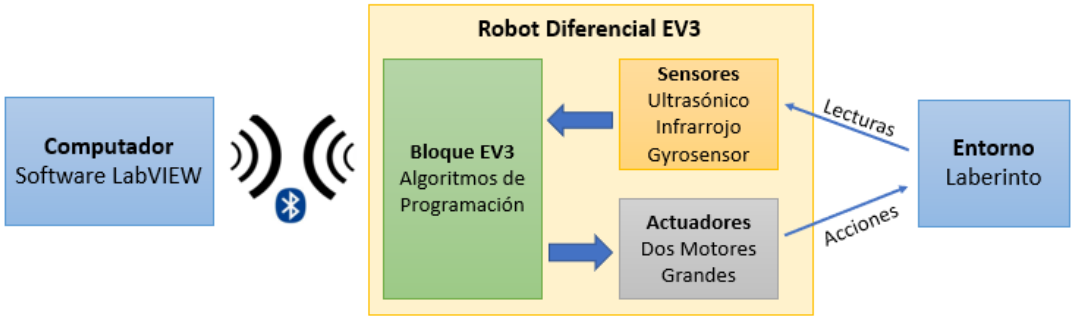
CAPÍTULO 3
DISEÑO E IMPLEMENTACIÓN

En este capítulo se redacta el desarrollo del presente proyecto tomando en consideración los diferentes parámetros de hardware y software. Se iniciará indicando los elementos que participan en la implementación de los dos algoritmos de planeación de rutas sobre el robot móvil diferencial EV3. Seguidamente se indicará como establecer la comunicación entre el controlador EV3 del robot móvil y el computador para el proceso de descarga y ejecución de los algoritmos de programación desarrollados en el software LabView. Luego se explicará la implementación de los algoritmos de planeación de ruta tipo insecto y A* mediante la programación en el software LabView.

3.1. Esquema General del Proyecto

En este proyecto intervienen diferentes elementos tanto de software como de hardware que interactúan entre sí para determinar la ruta de salida del robot móvil dentro de un laberinto conocido. En la Figura 3.1 se pueden observar los elementos que participan en el desarrollo del proyecto.

Figura 3.1 Esquema general del proyecto



Elaborado por: José Arias y Alex García

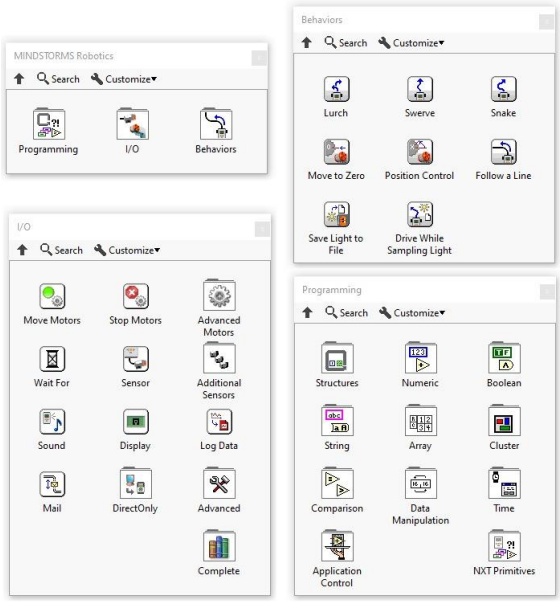
Según se observa en la Figura 3.1 el proyecto se compone de tres elementos principales: el robot móvil diferencial, el computador y el entorno donde se movilizará el robot que corresponde a un laberinto.

El robot móvil diferencial (**ver Anexo 1**) está compuesto por varios de los elementos mecánicos y electrónicos que vienen en el kit de robótica Mindstorms EV3. En cuanto a los sensores la estructura del robot incorpora un sensor ultrasónico ubicado en la parte

frontal del robot que le permitirá detectar las paredes del laberinto que se ubiquen frente al robot, también cuenta con un sensor de proximidad infrarrojo que detecta las paredes del laberinto que se localizan de forma lateral al robot e incorpora además un sensor giroscópico (gyrosensor) que permite al robot realizar giros exactos (usualmente de 90 y 180 grados). El robot este compuesto por dos motores grandes que le permiten movilizarse dentro del laberinto para generar la trayectoria de salida. La lógica de funcionamiento de robot móvil es controlada mediante el bloque EV3, al cual se le ha descargado previamente el código de programación.

Como parte de los equipos utilizados en este proyecto está el computador al cual se le ha instalado el software LabView versión 2016 puesto que solo se han generado controladores para el bloque EV3 hasta esta versión. Para poder programar al robot móvil diferencial EV3 mediante el software LabView se debe instalar previamente dos módulos adicionales, el uno conocido como LabView Robotics 2016 y el otro etiquetado como Lego Mindstorms 2016 (ver Figura 3.2), los cuales contienen todos los bloques necesarios para adquirir datos desde los sensores y controlar el accionamiento de los dos motores.

Figura 3.2 Bloques principales del módulo Lego Mindstorms 2016



Elaborado por: José Arias y Alex García

El intercambio de datos entre el computador y el robot móvil se puede realizar mediante comunicación inalámbrica bluetooth o USB. Por último, se ha construido un laberinto de madera como el mostrado en la Figura 3.3 que contienen una sola entrada y salida que pueden ser intercambiables.

Figura 3.3 Construccion de laberinto de madera.

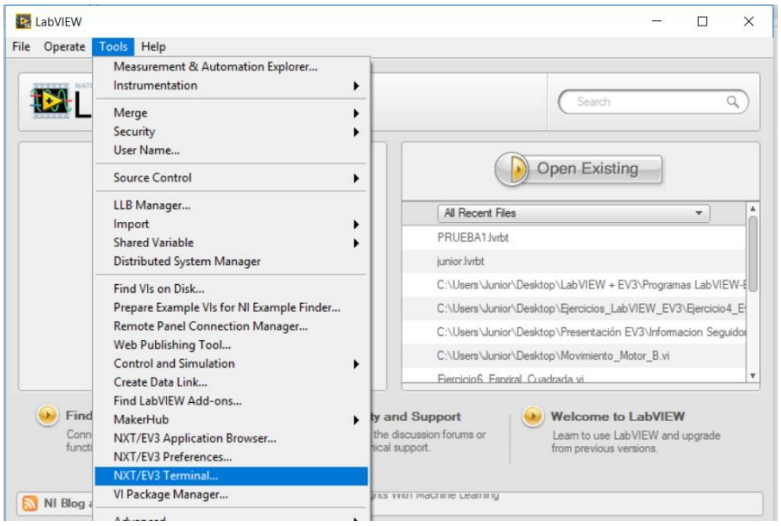
Elaborado por: José Arias y Alex García

3.2. Conexión Entre el Robot Móvil y el Computador

La conexión entre el robot móvil y el computador puede llevarse a cabo antes o después de elaborar el programa en el entorno de programación de LabView. Existen dos caminos para establecer la comunicación, mediante un cable USB o vía inalámbrica Bluetooth y en ambos casos es posible que se instalen automáticamente los controladores necesarios; adicionalmente, si la conexión es Bluetooth, se deberá llevar a cabo el proceso de apareamiento entre el bloque inteligente EV3 y el dispositivo Bluetooth del computador. De los dos métodos antes mencionados es necesario abrir la ventana del Terminal NXT/EV3 en LabView. Esta terminal sirve para establecer la conectividad entre el robot móvil y LabView. Para abrir la Terminal NXT/EV3 se debe proceder con los siguientes pasos.

Paso 1: Primero se debe abrir el entorno LabView haciendo doble clic en el ícono que está localizado en el escritorio del computador. Si todo se ejecutó correctamente aparecerá una ventana como se muestra en la Figura 3.4.

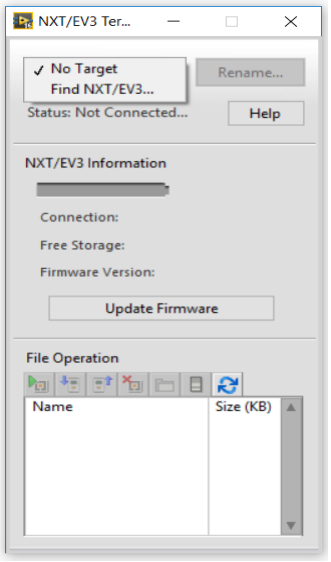
Figura 3..4 Ventana inicial del software LabView



Elaborado por: José Arias y Alex García

Paso 2: Para abrir la ventana de configuración se debe seguir la ruta *Tools >> NXT/EV3 Terminal* tras lo cual se abrirá una nueva ventada como indica la Figura 3.5.

Figura 3.5 Ventana inicial para busqueda del bloque EV3

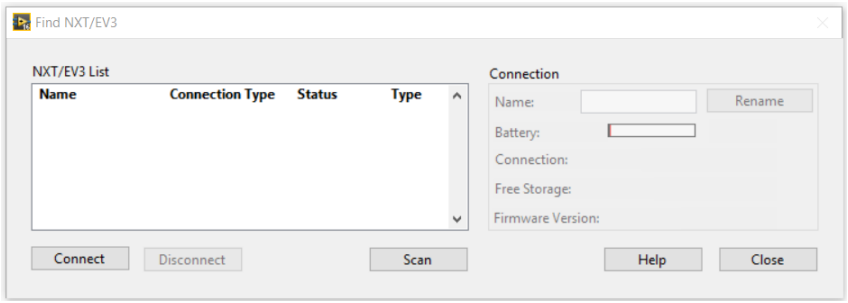


Elaborado por: José Arias y Alex García

Paso 3: La Terminal NXT/EV3 cuenta con un selector (*Target*) que permite buscar, añadir y establecer la comunicación entre el computador y uno o varios bloques inteligentes EV3. Para llevar a cabo este propósito se debe dar clic izquierdo sobre la opción "*Find NXT/EV3…"* tras la cual se despliega una nueva ventana de búsqueda como se indica en la Figura 3.6. Si se va a establecer una comunicación USB asegúrate de que

el robot está encendido y conecta el cable USB. Si la comunicación se va a establecer vía inalámbrica Bluetooth, asegúrate de que el módulo Bluetooth tanto del computador como del robot está activado.

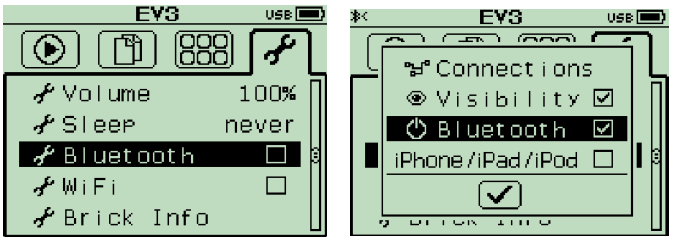
Figura 3.6 Ventana para escanear el bloque EV3



Elaborado por: José Arias y Alex García.

Paso 4: Asumiendo que el módulo Bluetooth del computador ya está activado solo resta activar el módulo Bluetooth del bloque inteligente EV3, para ello se utilizan los botones físicos del bloque. En primer lugar, hay que pulsar tres veces el botón "*Derecha*" del bloque inteligente EV3 con lo cual se accede a la pestaña de *Ajustes*  . Después, usando los botones "*Arriba*" y "*Abajo*" hay que seleccionar la opción: "*Bluetooth*" y pulsar el botón "*Aceptar*" que está en el centro. Tras estas acciones se abre una nueva ventana con varias opciones de menú como se indica en la Figura 3.7.

Figura 3.7 Ventana del bloque EV3 para la activacion Bluetooh

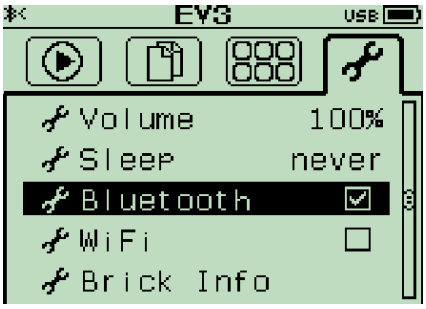


Elaborado por: José Arias y Alex García

Paso 5: Luego se debe activar los menús "*Visibility*" y "*Bluetooth*", como se puede ver en la captura de pantalla anterior. Al controlar el robot móvil mediante comunicación Bluetooth desde el computador, se tiene que deshabilitar o desmarcar la opción "*iPhone/iPad/iPod*" que se encuentra marcada por defecto en el bloque inteligente EV3. Después de marcar las casillas correspondientes, se debe pulsar el botón "*Aceptar*" que

está en el centro y de esta forma se ha habilitado la comunicación Bluetooth sobre el robot móvil como se indica en la Figura 3.8.

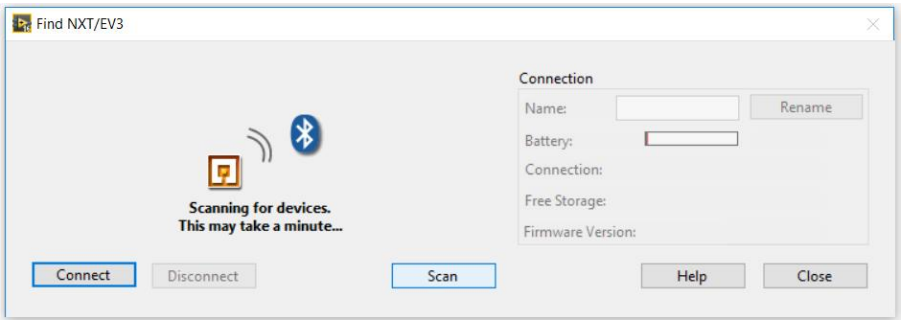
Figura 3.8 Módulo Bluetooth del bloque EV3 activado



Elaborado por: José Arias y Alex García

Paso 6: Una vez culminados los pasos anteriores tanto para la comunicación mediante cable USB o vía inalámbrica Bluetooth se debe regresar a la ventana de la Figura 3.6 localizada en el computador y dar clic derecho en el botón "*Scan*"; después de esta acción LabView empieza a buscar todos los bloques inteligentes conectados o cercanos al computador como se indica en la Figura 3.9.

Figura 3.9 Escaneo de lso bloques EV3 vinculados al computador

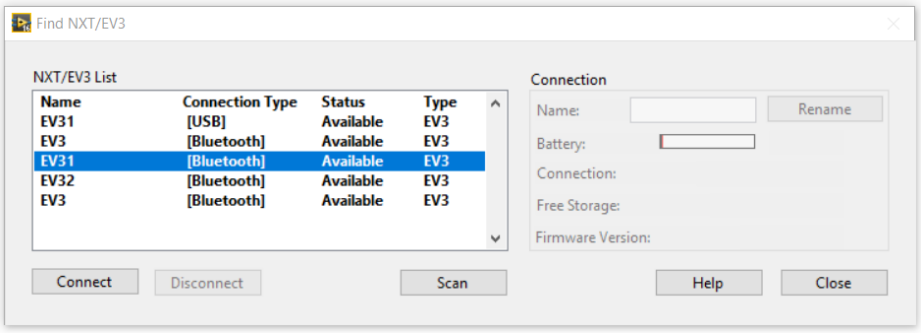


Elaborado por: José Arias y Alex García

Paso 7: Una vez culminado el proceso de escaneo se visualizan en la misma ventana todos los bloques inteligentes al alcance del computador (ver Figura 3.10) indicando también el tipo de conexión establecida. Además de los bloques inteligentes encontrados en ese momento, también se muestran los bloques que ya se hayan conectados al computador en sesiones anteriores. Por defecto el nombre asignado para cualquier bloque inteligente aparece como "*EV3*". Para poder diferenciar un bloque de otro conectado al

computador se debe renombrar dicho bloque una vez que se haya establecido la conexión definitiva.

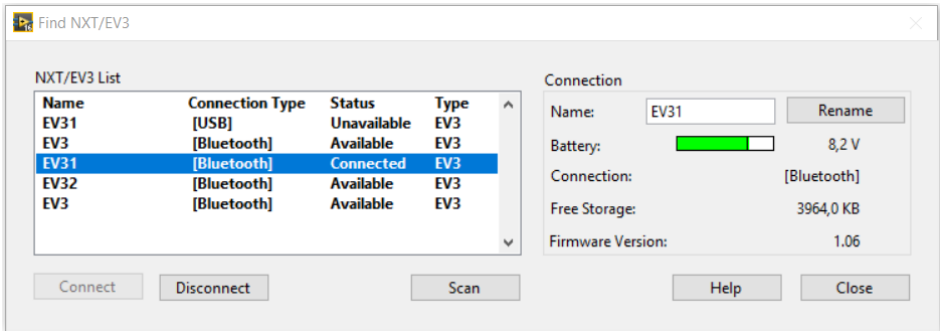
Figura 3.10 Bloques EV3 detectados



Elaborado por: José Arias y Alex García

Paso 8: El último paso es seleccionar el bloque inteligente EV3 del robot móvil y dar clic sobre el botón "*Connect*", se debe esperar unos segundos y el estado de conexión del dispositivo cambiará de "*Available*" a "*Connected*" como se indica en la Figura 3.11.

Figura 3.11 Conexión establecida entre el bloque EV3 y el computador



Elaborado por: José Arias y Alex García

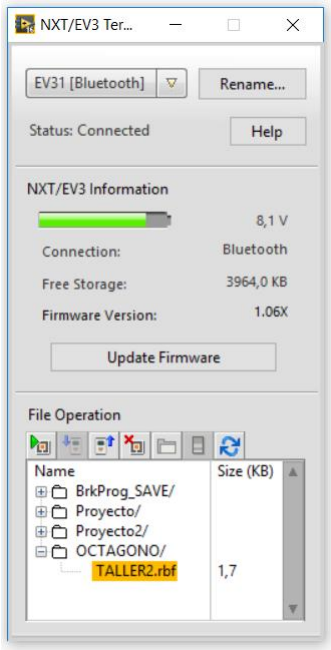
Después de estas acciones ya se tiene establecida por completo la conexión entre el computador y el robot, esta puede ser por vía inalámbrica Bluetooth o cable USB. Desde este momento es posible transferir los programas desarrollados en el entorno LabView hacia el bloque inteligente EV3 localizado en el robot. Lo último que resta es dar clic izquierdo sobre el botón "*Close*" para finalizar este proceso.

En cuanto se establece la conexión USB o Bluetooth, el nombre del bloque inteligente EV3 aparecerá en el selector (*Target*), en este caso con el nombre de *"EV31[Bluetooth]"* y el estado de conexión cambiará a "*Connected*" tal y como se muestra en la Figura 3.12.

Como ya se mencionó con anterioridad también es posible modificar el nombre del robot, de ser necesario, dando clic izquierdo sobre el botón "*Rename…*".

En la parte intermedia de la ventana de la Terminal NXT/EV3 etiquetada como "*NXT/EV3 Information*" se muestra información básica del bloque inteligente EV3 como el estado de la batería, tipo de conexión (*Connection*), memoria libre (*Free Storage*) y versión del software (*Firmware Version*). En caso necesario, se puede buscar una actualización del firmware y actualizar el bloque inteligente EV3 con la ayuda del botón "*Update Firmware*".

Figura 3.12 Ventana final de la conexión entre el computador y el bloque EV3



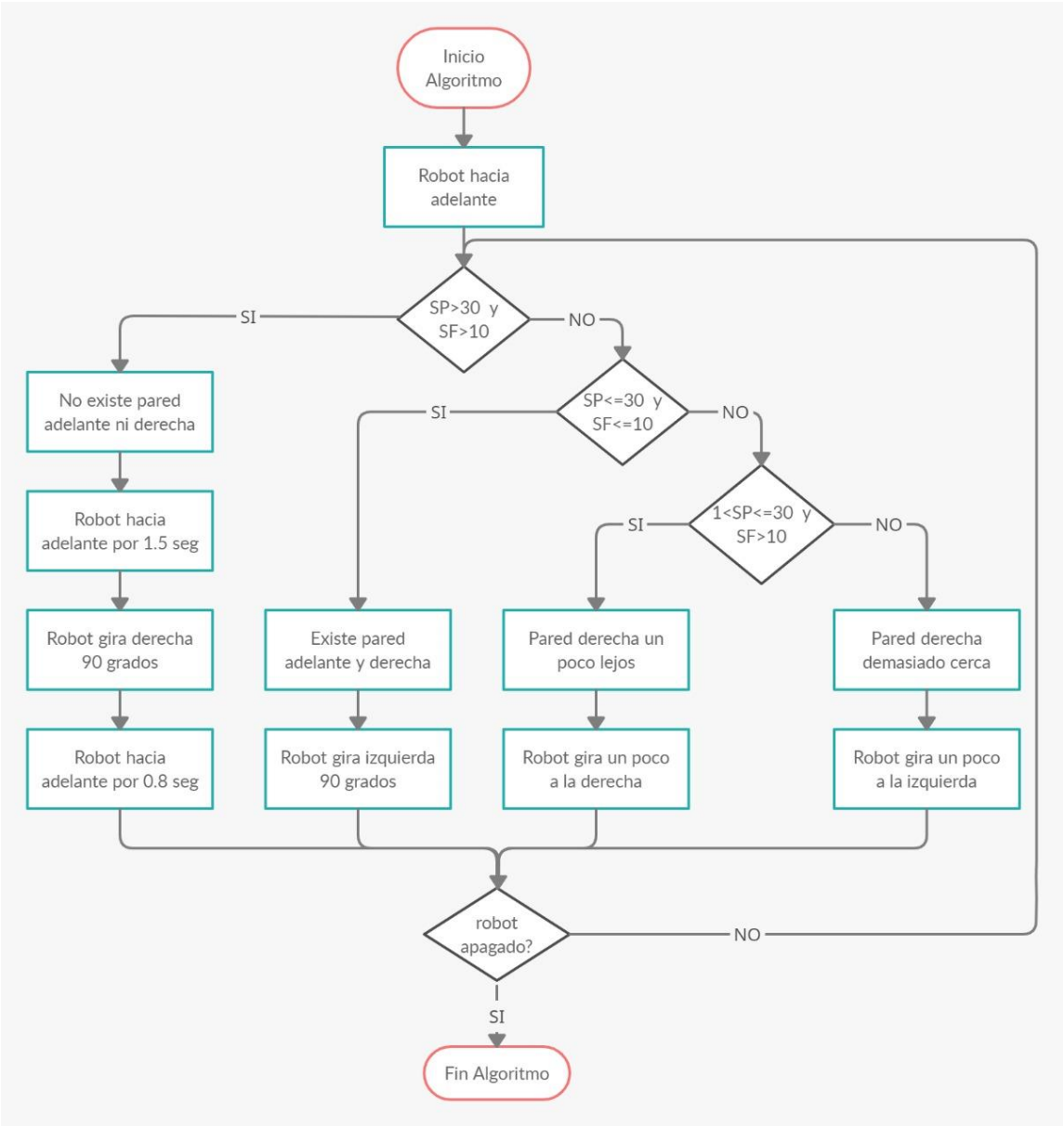
Elaborado por: José Arias y Alex García

3.3. Implementación del Algoritmo Tipo Insecto

En este apartado se va a explicar el código de programación del algoritmo Tipo Insecto que se implementó sobre el robot móvil diferencia EV3 para que este pueda determinar la ruta de salida sobre el laberinto físico mostrado en la Figura 3.3.

Como se indica en la Figura 3.13 el diagrama de flujo de la lógica de programación que posteriormente se traduce a código LabView. En este diagrama de flujo se hace referencia al sensor infrarrojo con las iniciales SP el cual determina la distancia entre el robot y la pared, también se hace referencia al sensor ultrasónico localizado al frente del robot con las iniciales SF el cual determina la distancia frontal del robot a una pared.

Figura 3.13 Diagrama de flujo del algoritmo Insecto

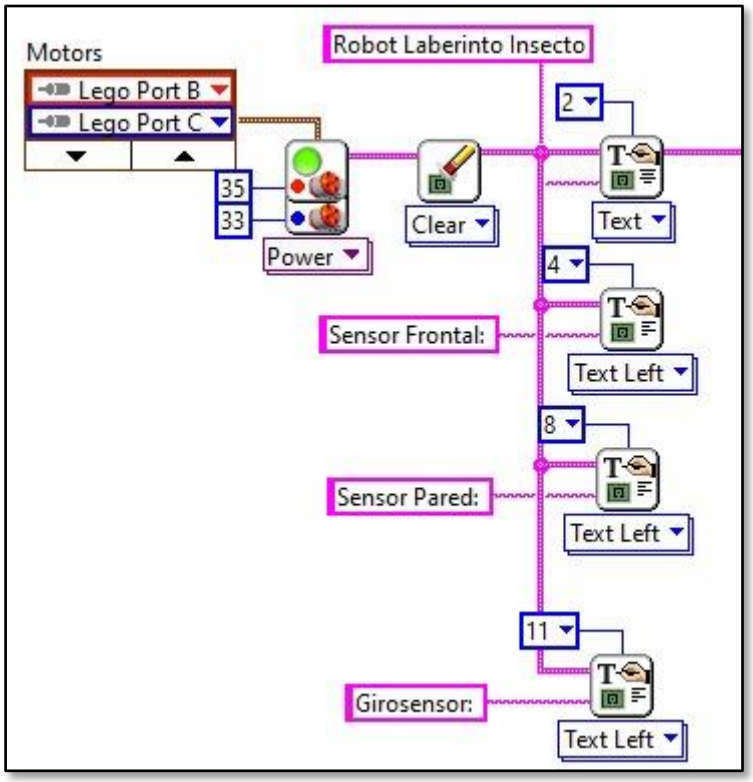


Elaborado por: José Arias y Alex García

Las distancias mínimas y máximas que se han tomado como referencia para los dos sensores incorporados en el robot (30 unidades para el sensor SP y 10 para el sensor SF) pueden ser modificadas acorde a la textura-color de las paredes del laberinto y las condiciones de luz ambiental en las que se encuentre en ese momento el robot ejecutando el algoritmo.

En la Figura 3.14 se muestran las configuraciones iniciales que se deben programar sobre el robot móvil diferencial, específicamente la velocidad inicial de los motores y los parámetros numéricos que se muestran sobre el bloque inteligente EV3.

Figura 3.14 Configuraciones iniciales para la pantalla del bloque EV3



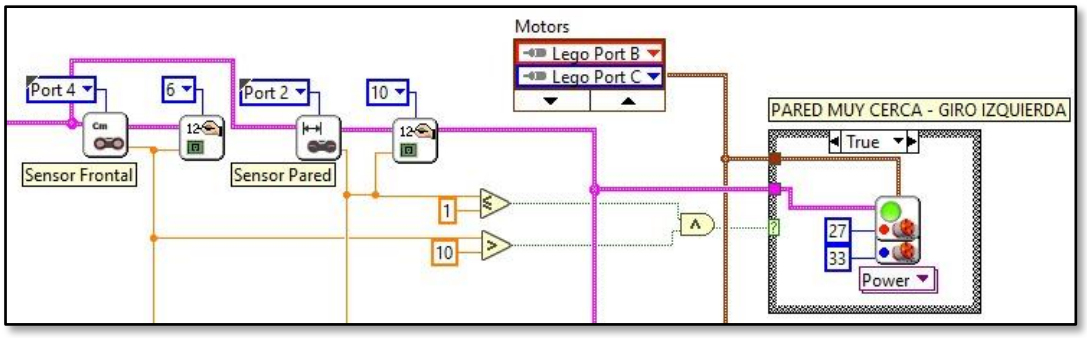
Elaborado por: José Arias y Alex García

El robot móvil debe iniciar su recorrido hacia adelante en línea recta, pero para cumplir con este objetivo no se puede colocar la misma potencia a los motores debido al peso no equilibrado del robot y a las características propias de diseño de los dos motores grandes que a pesar de ser de la misma serie no generan la misma velocidad si se les aplica potencias iguales puesto que siempre tienen una pequeña diferencia. Una vez realizadas varias pruebas se consideró colocar una potencia de 35 al motor derecho conectado al Puerto C y una potencia de 33 al motor izquierdo conectado al puerto B. También en esta primera configuración se han colocado varios textos en la pantalla del bloque EV3 junto a los cuales posteriormente se mostrará de forma numérica el valor del "Sensor Frontal", "Sensor Pared" y el "Girosensor".

En la Figura 3.15 se muestran los bloques utilizados para adquirir y visualizar de forma numérica los valores obtenidos por el sensor frontal (SF) y el sensor pared (SP); este fragmento de código también será utilizado por los siguientes subprocesos que serán explicados más adelante. El sensor frontal se encuentra conectado al Puerto 4 del bloque EV3 y el sensor pared al Puerto 2. Si la distancia del SP es menor o igual a 1 y la distancia

del SF es mayor que 10 entonces significa que el robot no tiene obstáculo al frente y está muy pegado a la pared por lo cual se debe realizar un pequeño giro hacia la izquierda. En el caso *False* de la estructura *Case* no se ha programado nada.

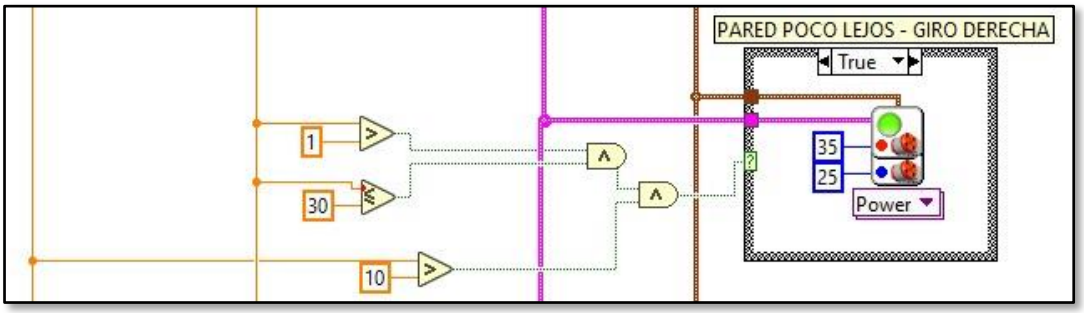
Figura 3.15 Lógica de programación cuando el robot está muy cerca a la pared.



Elaborado por: José Arias y Alex García

Como se indica en la Figura 3.16 la lógica de programación cuando el robot móvil se encuentra un poco alejado de la pared. Si la distancia del SP es mayor a 1 y menor o igual a 30 y la distancia del SF es mayor que 10 entonces significa que el robot no tiene obstáculo al frente y está un poco alejado de la pared por lo cual se debe realizar un pequeño giro hacia la derecha. En el caso *False* de la estructura *Case* no se ha programado nada.

Figura 3.16 Lógica de programación cuando el robot está un poco lejos de la pared.

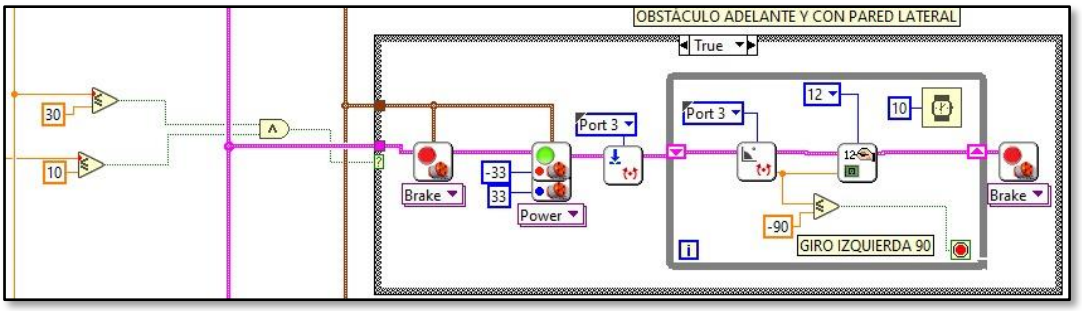


Elaborado por: José Arias y Alex García

Como se indica en la Figura 3.17 se muestra la lógica de programación cuando el robot móvil se encuentra con pared al frente y pared a la derecha, esto ocurre si la distancia del SP es menor o igual a 30 y la distancia del SF es menor o igual a 10, entonces el robot

móvil debe detenerse y luego realizar un giro de 90 grados hacia la izquierda. En el caso *False* de la estructura *Case* no se ha programado nada.

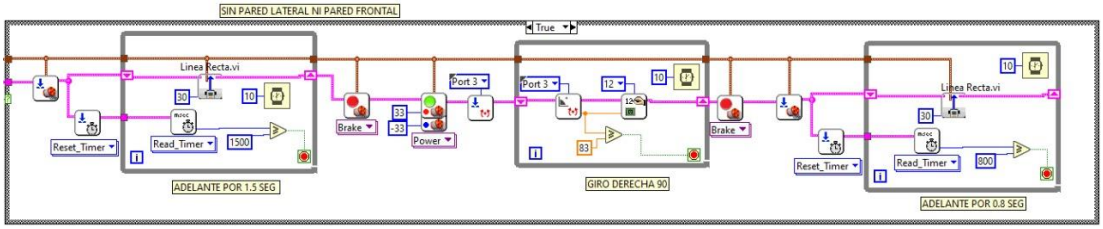
Figura 3.17 Lógica de programación cuando el robot detecta pared al frente y pared a la derecha.



Elaborado por: José Arias y Alex García

Como se indica en la Figura 3.18 la lógica de programación cuando el robot móvil no detecta ninguna pared al frente ni a la derecha, esto ocurre si la distancia del SP es mayor a 30 y la distancia del SF es mayor a 10, entonces el robot móvil debe seguir en línea recta hacia adelante por un tiempo de 1.5 segundos, luego debe detenerse, seguidamente debe realizar un giro de 90 grados hacia la derecha y finalmente seguir en línea recta hacia adelante por un tiempo de 0.8 segundos, todas estas acciones se realizan con la finalidad de que el robot vuelva a detectar una pared ubicada a la derecha dela estructura. En el caso *False* de la estructura *Case* no se ha programado nada.

Figura 3.18 Lógica de programación cuando el robot no detecta pared al frente ni pared a la derecha



Elaborado por: José Arias y Alex García

Los fragmentos de códigos individuales explicados anteriormente están dentro de una estructura *While*, la cual debe repetirse indefinidamente hasta que se presione el pulsador que se encuentra conectado al Puerto 1 del bloque EV3, que permite detener por completo

la ejecución del algoritmo tipo insecto en cualquier instante de tiempo. El código completo del algoritmo Tipo Insecto se muestra en el **Anexo 2**.

3.4. Implementación del Algoritmo A-Star

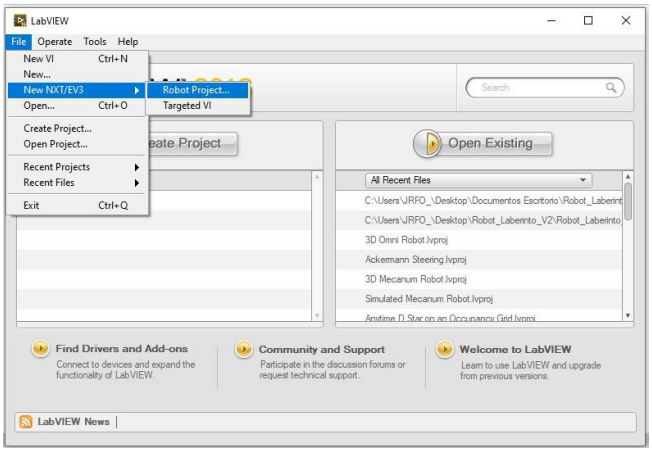
En este apartado se explica la implementación del código de programación para la solución del laberinto mediante el algoritmo A*. Debido a que la lógica de programación es demasiada extensa se optó por crear un proyecto que contenga varios SubVIs que posteriormente serán utilizados en el programa definitivo. La idea detrás de la implementación de esta lógica de programación es la de diseñar un algoritmo matemático que proporcione al robot móvil diferencial EV3 la capacidad de navegar por un entorno conocido de forma autónoma, como por ejemplo un laberinto y determinar la mejor ruta de salida. Esto conlleva a generar un arreglo bidimensional que represente la configuración del laberinto físico y que permitirá especificar las posiciones de salida y llegada del robot móvil.

Antes de explicar los diagramas de flujos y la programación detrás de los diferentes SubVIs implementados, es necesario conocer la funcionalidad de varios bloques del módulo LabView Robotics utilizados en la planeación de rutas, cuya explicación pueden ser consultada en el **Anexo 3**.

Para el desarrollo del algoritmo debe considerarse que el diseño del laberinto puede especificarse como una cuadrícula bidimensional, es decir, que el robot se movilizará en un medio donde cada posición de la cuadrícula tiene coordenadas enteras de la forma (*i*, *j*) y por lo tanto ejecutará pasos discretos en cualquiera de las cuatro direcciones (derecha, izquierda, abajo, arriba), en donde cada movimiento realizado incrementa o decrementa una coordenada. Bajo esta premisa se requiere establecer las dimensiones de la cuadrícula y el área de cada elemento del laberinto. Se diseñó un laberinto con una distribución de 14x14 divisiones, considerando las paredes y los espacios vacíos por donde se desplazará el robot; las dimensiones de las paredes se consideran insignificantes (pero su empleo en la lógica de programación es de suma importancia), mientras que el área de los espacios vacíos está formada por recuadros de 25 cm de longitud. Tomando en consideración estos datos y en base a los parámetros de diseño de los actuadores del robot móvil se establece el tiempo que le llevará al vehículo desplazarse de un punto a otro o ejecutar un giro especifico.

La implementación del algoritmo A* inicia con la creación de un proyecto etiquetado como *Robot_Laberinto_AStar*, siguiendo la ruta mostrada en la Figura 3.19 de la ventana inicial de LabView.

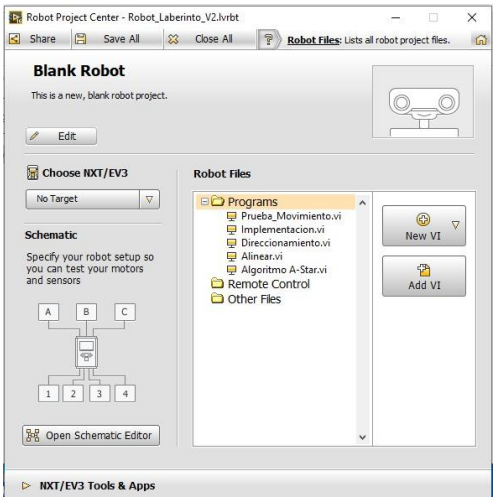
Figura 3.19 Ventana inicial de LabView 2016



Elaborado por: José Arias y Alex García

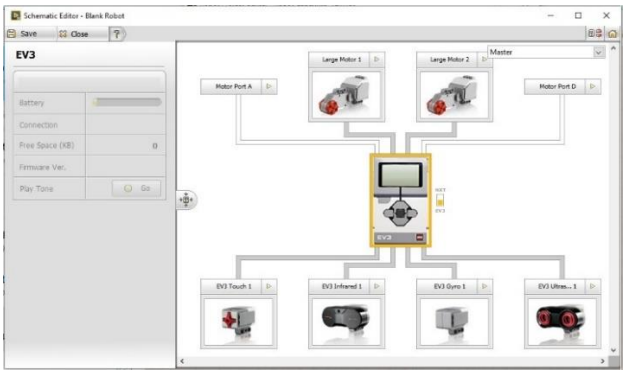
La vista final de la ventana de configuración es la mostrada en la Figura 3.19, en donde se han creado y añadido varios VIs que se utilizarán en conjunto para generar el algoritmo de programación completo. El *VI Implementación* es el que contiene la codificación del programa principal de ejecución, y que posteriormente emplea los VIs *Algoritmo A-Star* y *Direccionamiento* para determinar la mejor ruta dentro del laberinto y controlar el direccionamiento del robot móvil, respectivamente. El código de programación de estos sub-algoritmos serán explicados en los siguientes apartados de forma individual. Una vez creado el proyecto y añadidos los diferentes SubVIs se debe configurar la localización de los diferentes motores y sensores que incorpora el robot móvil diferencial EV3, para lo cual se debe activar el botón *Open Schematic Editor* que permitirá desplegar la ventana de configuración visualizada en la Figura 3.20. Haciendo uso de esta nueva ventana se deben elegir los puertos de entradas y salidas del bloque EV3 donde están conectados de forma física los dos motores y los cuatro sensores utilizados en toda la programación del algoritmo de planeación de rutas, tal y como se presenta en la Figura 3.21

Figura 3.20 Ventana para la configuración de proyectos EV3



Elaborado por: José Arias y Alex García

Figura 3.21 Ventana Schematic Editor



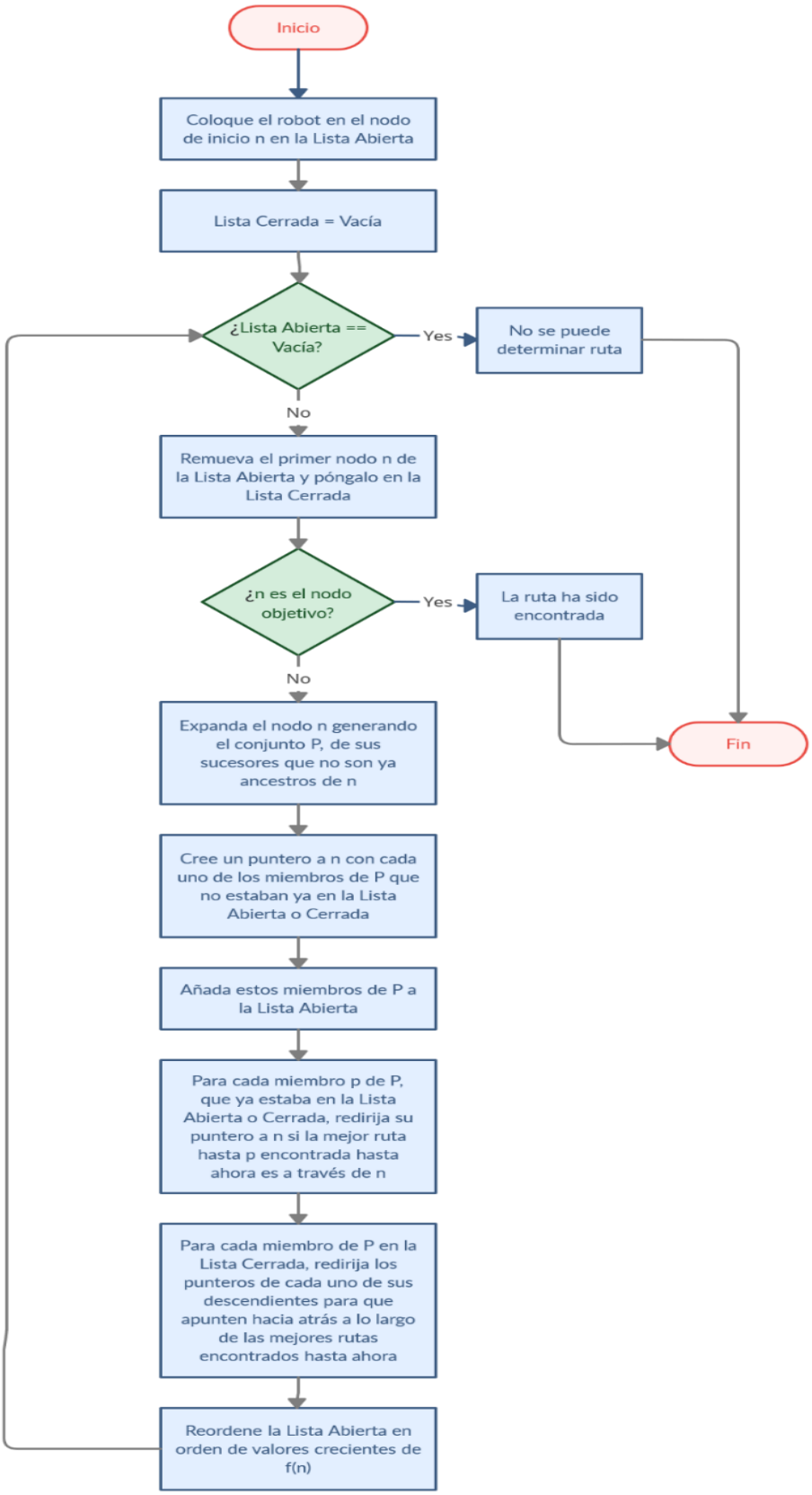
Elaborado por: José Arias y Alex García

Algoritmo de Programación del VI A-Star

En esta sección se va a explicar el código de programación del algoritmo A* que se implementó sobre el robot móvil diferencia EV3 para que este pueda determinar la mejor ruta de salida sobre el laberinto físico.

Como se indica en la Figura 3.22 el diagrama de flujo de la lógica de programación del algoritmo A* explicado en el apartado 2.4.2 que posteriormente se traduce a código LabView.

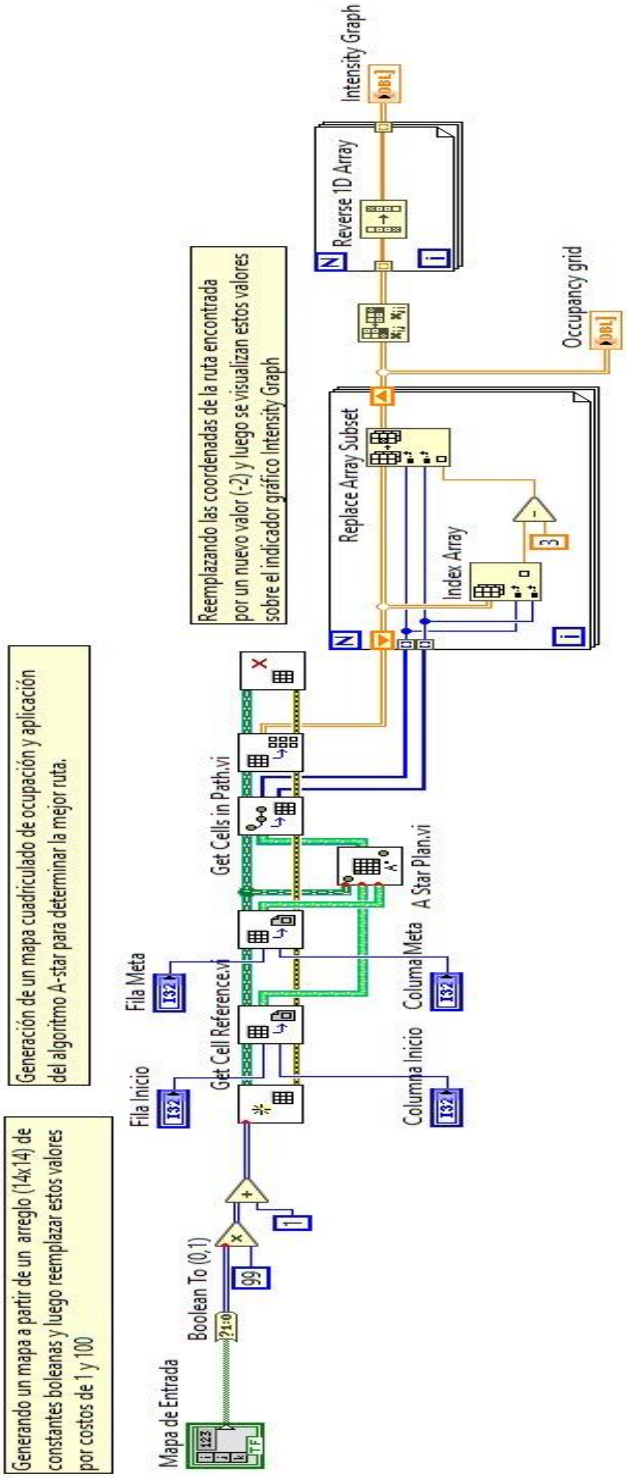
Figura 3.22 Diagrama de flujo del algoritmo A-Star



Elaborado por: José Arias y Alex García

A partir del diagrama de flujo de la Figura 3.22 se ha desarrollado el código de programación en LabView mostrado en la Figura 3.23, en donde se han utilizado varias de los comandos incorporados en el módulo LabView Robotics diseñados para el proceso de planeación de rutas.

Figura 3.23 Código de programación en LabView del algoritmo A-Star



Elaborado por: José Arias y Alex García

El algoritmo inicia con la definición de un arreglo bidimensional que representa al laberinto físico, por esta razón se ha generado en el panel frontal una matriz compuesta por bloques de controles booleanos del tipo *Push Button*. Los estados True constituyen los espacios ocupados y los estados False corresponden a los espacios desocupados.

La señal de salida de la matriz booleana es dirigida a la función *Boolean To (0,1)* (*Programming* > *Boolean*), que permite transformar un estado booleano True o False en un valor numérico 1 o 0, respectivamente. Luego se realiza un proceso de transformación en donde los elementos del arreglo bidimensional que tengan el estado 1 son sustituidos por 100 y los elementos que tengan el estado 0 son sustituidos por 1. Esta conversión se debe realizar obligatoriamente puesto que los VIs *Occupancy Grid Map* solo pueden descifrar como espacios ocupados el valor de 100 y como espacios vacíos el valor de 1. A continuación, se indica la finalidad de todos los VIs que conforman el algoritmo de programación y que permiten determinar la mejor ruta que el robot debe seguir para salir del laberinto.

- El VI *Create Occupancy Grid Map* permite crear un mapa visto como una cuadrícula la cual representa el medio (laberinto) por el cual se movilizará el robot.
- Al VI *Get Cell Reference* se le proporcionan las coordenadas de la posición de inicio del robot y en el siguiente VI *Get Cell Reference* se le ingresan las coordenadas de la posición de meta del robot.
- Las señales de salidas (*cell reference)* de los anteriores bloques son direccionadas a las entradas *start reference* y *goal reference* del código *A Star Plan* el cual generará como resultado el camino de menor costo que seguirá el vehcíulo dentro del laberinto.
- La salida *path reference* del VI *A Star Plan* es direccionada a la entrada del bloque *Get Cells in Path*, que generará todas las coordenadas (*x,y*) para la ruta encontrada. Las coordenadas *x* y *y* son guardadas por separado como parte de los elementos de un arreglo unidimencional.
- Posteriormente, utilizando el bloque *Get All Occupancy Grid Cells* se accede al arreglo bidimencional que corresponde al laberinto fisico y finalmente se concluye la referencia *occupancy grid map*.

Utilizando la estructura de repetición *For* y considerando las señales derivadas de los bloques *Get Cells in Path* y *Get All Occupancy Grid Cells* se desarrolla la codificación de programación encargada de sustituir sobre el arreglo 2D las coordenadas (*x,y*) que representan a la ruta determinada por el algoritmo A*. Todos los componentes son

sustituidos por el valor numérico -2, lo cual permitirá posteriormente codificar la lógica de programación del VI *Implementación*.

Por último, en un indicador gráfico de tipo *Intensity Graph* se visualizará el camino que seguirá el vehículo móvil dentro del laberinto. Las señales de entrada del VI corresponden a las salidas de los controles etiquetados como *Mapa de Entrada, Fila Inicio, Columna Inicio, Fila Meta* y *Columna Meta*. La salida de este bloque (*Occupancy grid*) es un arreglo bidimensional que ejemplifica al laberinto y en donde los componentes con valores de -2 constituyen el camino que debe ejecutar el vehículo móvil.

3.4.1. Algoritmo de Programación del VI Direccionamiento

La finalidad de este subVI es la de interpretar y procesar la información contenida en la matriz bidimensional generada por el subVI *Algoritmo A-Star* como se muestra en la Figura 3.25. Además, en este algoritmo se programa la lógica de funcionamiento para el control de los movimientos del vehículo móvil. Considerando que el código de programación es muy extenso, se ha optado por realizar la explicación de este dividiéndolo en varias secciones. Para determinar la posición y orientación (postura) que el vehículo seguirá sobre el laberinto, se deben comparar las filas y columnas en los alrededores de la posición actual en la que se encuentra localizado, de la manera que se esquematiza en la Figura 3.24.

Las variables *Col* y *Fila* mostradas en la Figura 3.24 corresponden a las coordenadas actuales del robot móvil. En este punto se debe aclarar que en la implementación del código se consideró que el vehículo puede movilizarse de una posición a otra únicamente empleando movimientos hacia delante y con giros múltiplos de 90 grados. Por ejemplo, cuando el vehículo debe movilizarse a la posición *Atrás Diagonal Izquierda* y se considera que su postura actual es la mostrada en la Figura 3.24, el robot primero deberá realizar un giro antihorario de 90 grados, luego trasladarse hacia adelante un tiempo y/o distancia determinada, después debe realizar otro giro antihorario de 90 grados y por último trasladarse hacia adelante, considerando tiempos y distancias específicas.

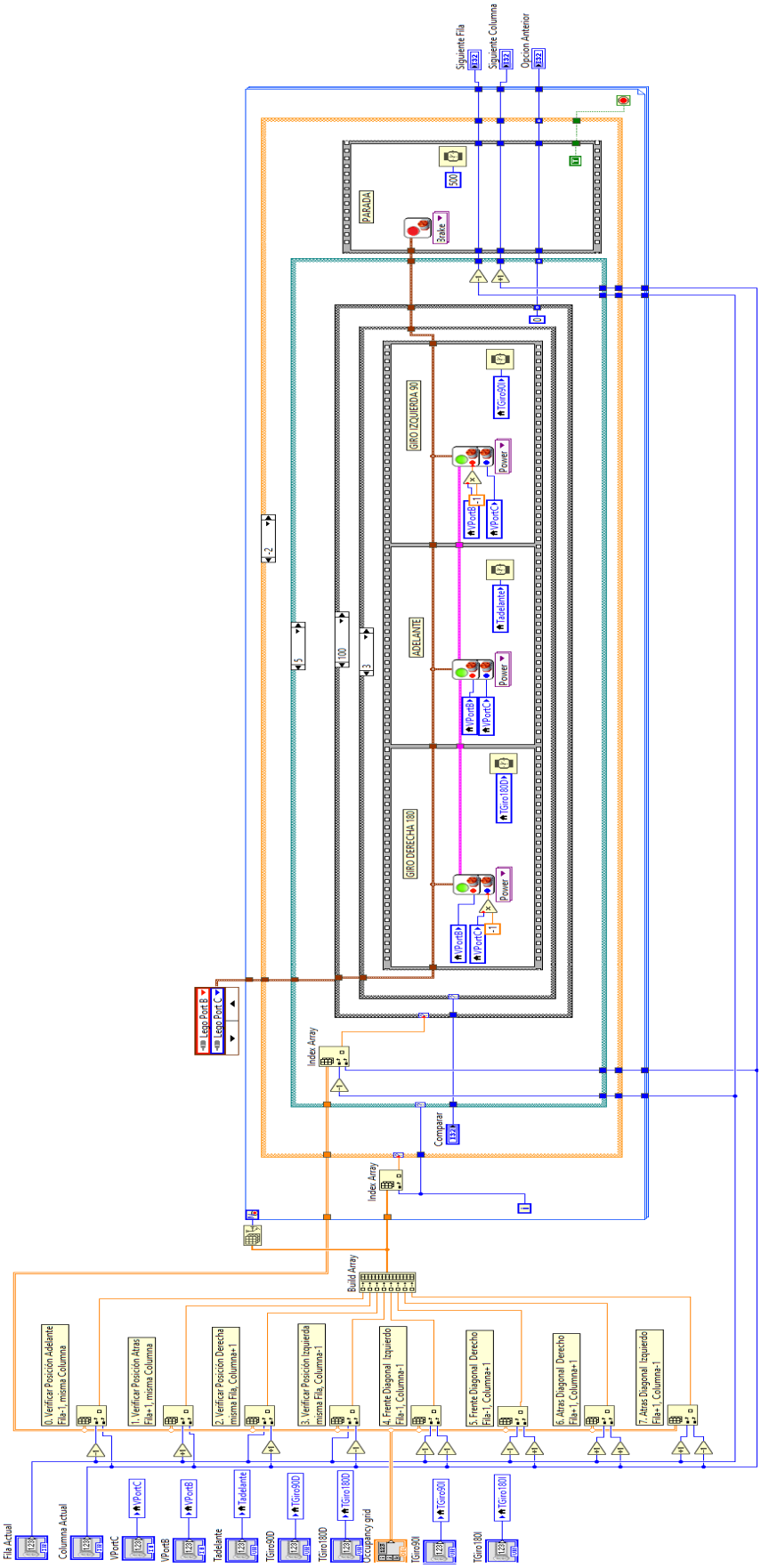
Figura 3.24 Lógica de desplazamiento del robot EV3

Frente Diagonal Izquierda Col-1 Fila-1	**Frente** **Misma Col** **Fila-1**	Frente Diagonal Derecha Col+1 Fila-1
Posición Lateral Izquierda **Col-1** **Misma Fila**		**Posición Lateral Derecha** **Col+1** **Misma Fila**
Atrás Diagonal Izquierda Col-1 Fila+1	**Atrás** **Misma Col** **Fila+1**	Atrás Diagonal Derecha Col+1 Fila+1

Elaborado por: José Arias y Alex García

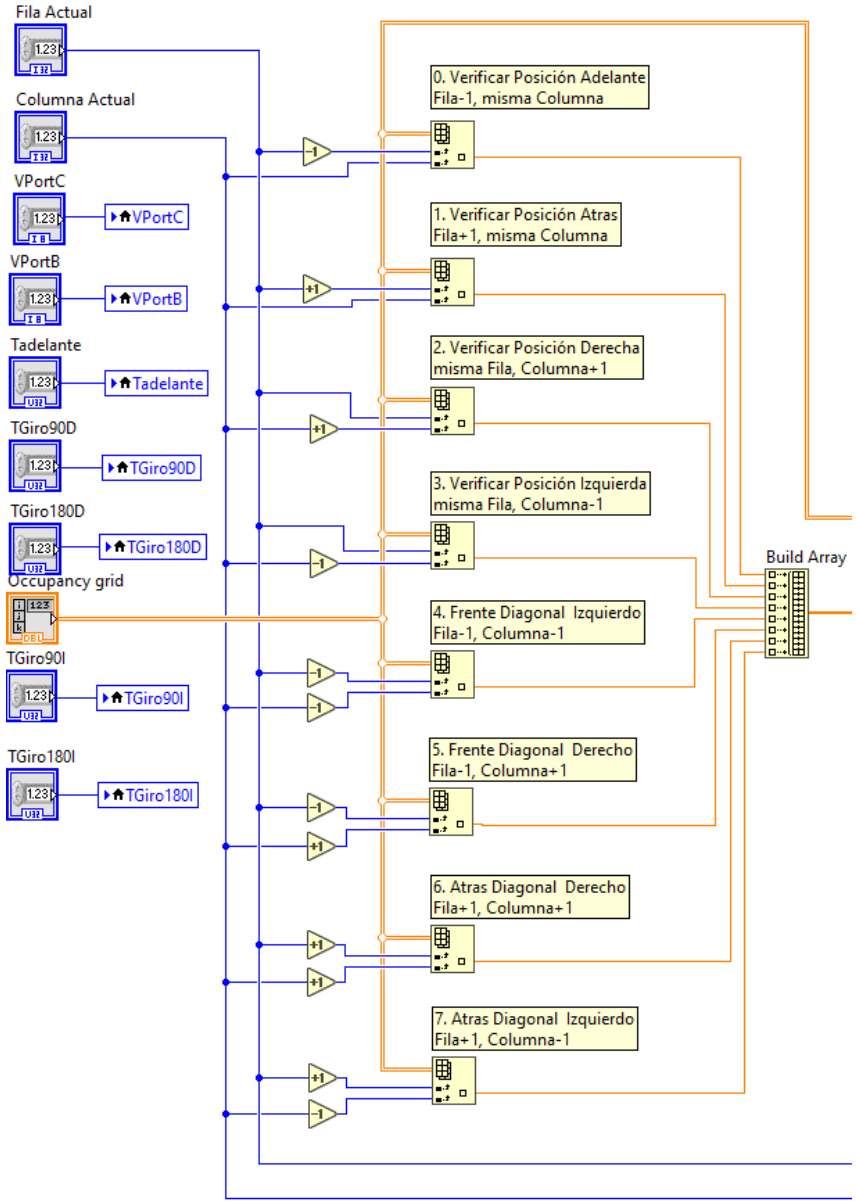
Tomando en consideración la lógica mostrada y analizada en la Figura 3.24 se procede a generar el código de programación para que el robot pueda posicionarse en cada una de estas posiciones a partir de su posición actual (ver Figura 3.26). El código muestra un control numérico marcado como *Ocuppancy grid* el cual representa una matriz bidimensional previamente creada en el panel frontal. Además, se cuenta con dos controles numéricos marcados como *Fila Actual* y *Columna Actual* que hacen referencia a las coordenadas de la posición actual del vehículo. Tomando como coordenada inicial esta posición se irá aumentando o disminuyendo en una unidad el valor de la fila o la columna para encontrar las otras posiciones en la cercanía del vehículo, con estas acciones se puede conseguir el valor guardado en cada una de estas coordenadas dentro de la matriz que previamente fue generada por el subVI *Algoritmo A-Star.* Para efectuar el paso explicado se emplea la función *Index Array*. Luego de haber obtenido todos los valores alrededor de la posición actual, estos son transferidos a un arreglo unidimensional empleando el comando *Build Array*. Los valores guardados en el nuevo arreglo son posteriormente analizados, tomando en consideración el orden en el que fueron ingresados y el índice asignado, tal y como se visualiza en la Figura 3.26.

Figura 3.25 Diagrama de bloques del subVI Direccionamiento



Elaborado por: José Arias y Alex García

Figura 3.26 Verificación de las posiciones adyacentes al robot móvil



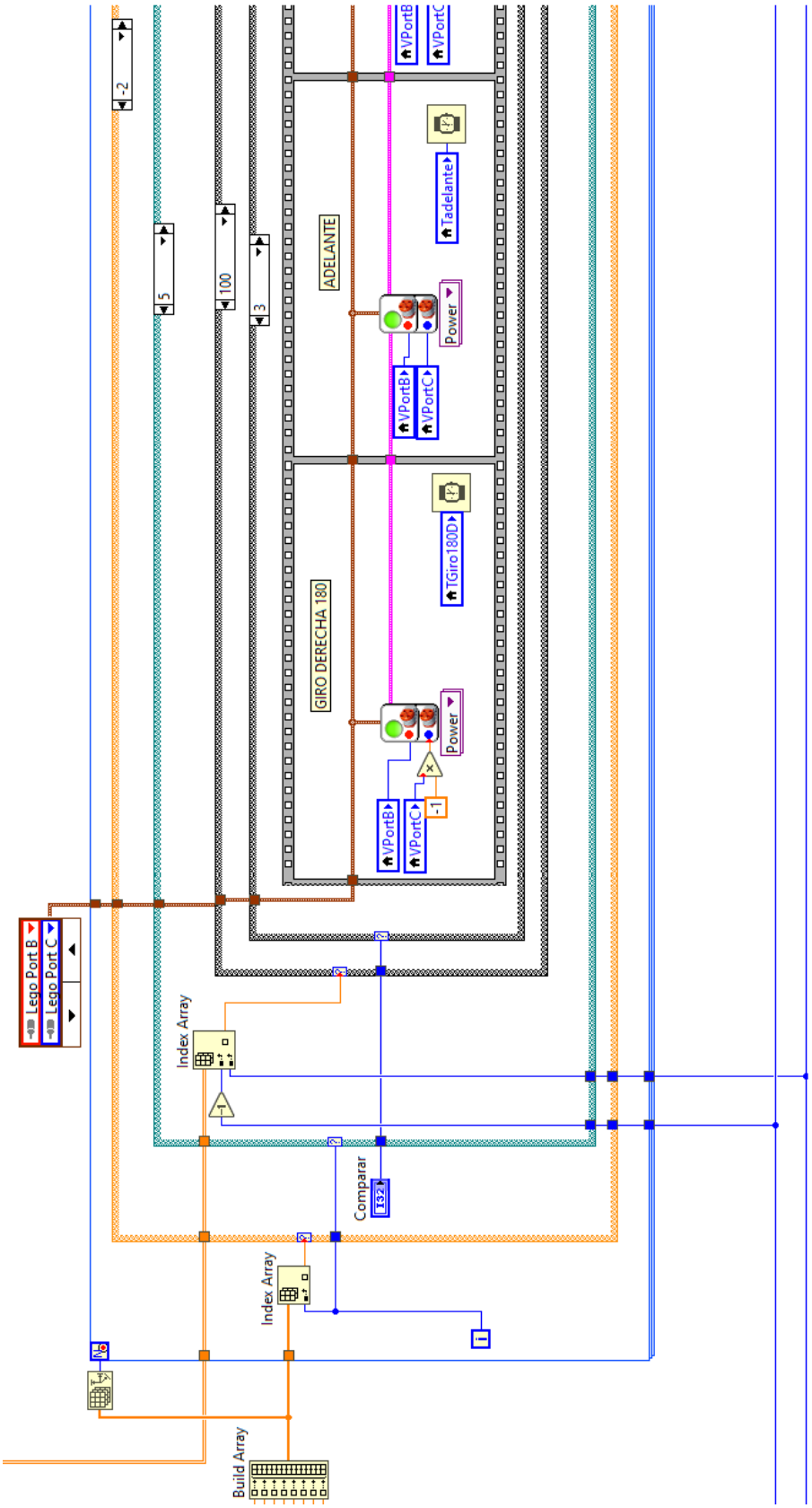
Elaborado por: José Arias y Alex García

Continuando con el análisis del programa principal (Figura 3.25), la matriz unidimensional anterior es transferida como un dato de entrada a una estructura *For*, en donde el número de iteraciones depende de la cantidad de elementos del arreglo y también a este bucle se le ha añadido un terminal condicional para realizar una parada independiente del número de interacciones. Al interior del bucle *For* se localiza una estructura *Case* con su selector conectado al valor del elemento actual del arreglo dado por el índice de iteración del bucle *For* y que es accedido mediante la función *Index Array (*ver Figura 3.27). Si un valor numérico de -2 es leído en la posición del índice actual del

arreglo, significa que el robot debe moverse a dicha posición y se deberá ejecutar el código correspondiente para realizar los movimientos necesarios y cumplir con dicho objetivo. Si el valor leído es diferente a -2, por lo que la estructura *Case* aguarda la siguiente iteración del *For* para leer y comparar el valor del siguiente índice del arreglo; dicho procedimiento se repetirá hasta localizar el valor de -2 en el arreglo. Tener presente que alrededor de la posición actual solo se encuentra un valor numérico de -2 al cual se debe de trasladar el robot móvil.

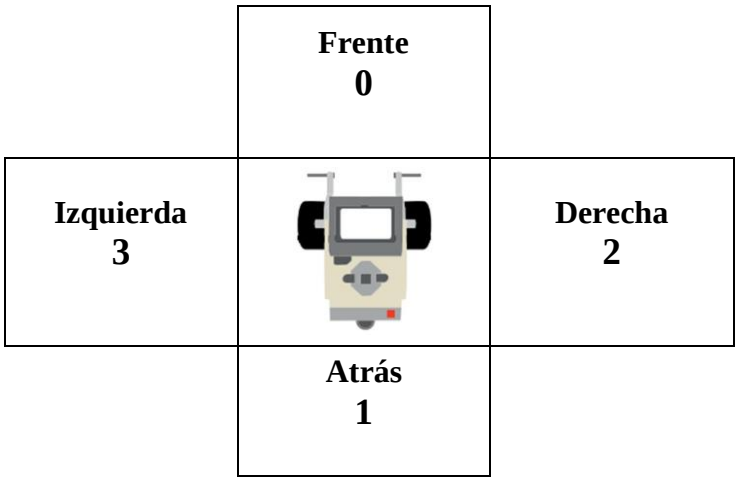
Luego de haber localizado el valor de -2, es indispensable encontrar el índice del arreglo de su ubicación; dicho índice representa el valor del indicador de iteraciones del bucle *For*, el cual es transferido a la entrada de una nueva estructura *Case* en donde se programará la lógica para la generación de los movimientos del vehículo con una velocidad constante. Las acciones que se deben generar para realizar cambios de posiciones en base al índice del arreglo se han enumerados desde el número 0 hasta el 7, como se observa en la Figura 3.27. Para la implementación del código de programación hay que considerar la dirección en la que apunta la parte frontal del vehículo en su movimiento anterior, dichas direcciones se han enumerado desde el número 0 hasta el 3 y su interpretación se indica en la Figura 3.28.

Figura 3.27 Lógica para generar los movimientos que debe realizar el vehiculo para ejecutar el camino dado por el algoritmo A*



Elaborado por: José Arias y Alex García

Figura 3.28 Identificación de las posibles direcciones en las que puede quedar ubicada la parte frontal del vehículo.



Elaborado por: José Arias y Alex García.

Como se observa en la Figura 3.27, luego de haber encontrado la siguiente posición del vehículo, hay que determinar la dirección actual de la parte frontal del mismo y según el conocimiento de estos datos se genera la lógica de programación para controlar las velocidades y sentidos de giro de los motores. Para realizar estas acciones se emplea la estructura *Case* cuyos casos representan las diferentes direcciones donde apuntaría la parte frontal del vehículo y cuyo selector está conectado a un control numérico etiquetado como *Comparar* el cual contiene la información de la dirección inicial del robot. Al interior de los diferentes casos de dicha estructura se emplean los bloques *Move DC Motors* que permiten establecer sobre el controlador EV3 las diferentes velocidades de los motores.

Tabla 3.1 Movimientos del robot móvil según la dirección actual de la parte delantera del chasis.

Posición Meta: Frente (0)		
Vista Actual	**Acción**	**Vista Siguiente**
0	Adelante	0
1	Giro180º-Adelante	
2	Giro Izquierda 90º-Adelante	
3	Giro Derecha 90º-Adelante	
Posición Meta: Atrás (1)		
0	Giro 180º-Adelante	1
1	Adelante	
2	Giro Derecha 90º-Adelante	
3	Giro Izquierda 90º-Adelante	
Posición Meta: Lateral Derecha (2)		
0	Giro Derecha 90º -Adelante	2
1	Giro Izquierda 90º-Adelante	
2	Adelante	
3	Giro 180º-Adelante	
Posición Meta: Lateral Izquierda (3)		
0	Giro Izquierda 90º-Adelante	3
1	Giro Derecha 90º-Adelante	
2	Giro 180º-Adelante	
3	Adelante	

Elaborado por: José Arias y Alex García

Tabla 3.2 Movimientos para trasladarse a una posición frente diagonal izquierda.

Posición Frente Diagonal Izquierda			
Ruta por donde seguir	**Vista Actual**	**Acción**	**Vista Siguiente**
Frente (si tiene un valor de 1 en la posición frente)	0	Adelante-Giro Izquierda 90º-Adelante	3
	1	Giro180º-Adelante-Giro Izquierda 90º-Adelante	
	2	Giro Izquierda 90º-Adelante-Giro Izquierda 90º-Adelante	
	3	Giro Derecha 90º - Adelante-Giro Izquierda 90º-Adelante	
Izquierda (si tiene un valor de 100 en la posición frente)	0	Giro Izquierda 90º- Adelante-Giro Derecha 90º-Adelante	0
	1	Giro Derecha 90º-Adelante-Giro Derecha 90º-Adelante	
	2	Giro 180º-Adelante-Giro Derecha 90º-Adelante	
	3	Adelante-Giro Derecha 90º-Adelante	

Elaborado por: José Arias y Alex García

Tabla 3.3 Movimientos para trasladarse a una posición frente diagonal derecha.

Posición Frente Diagonal Derecha			
Ruta por donde seguir	**Vista Actual**	**Acción**	**Vista Siguiente**
Frente (si tiene un valor de 1 en la posición frente)	0	Adelante-Giro Derecha 90º-Adelante	2
	1	Giro 180º-Adelante-Giro Derecha 90º-Adelante	
	2	Giro Izquierda 90º- Adelante-Giro Derecha 90º-Adelante	
	3	Giro Derecha 90º-Adelante-Giro Derecha 90º-Adelante	
Derecha (si tiene un valor de 100 en la posición frente)	0	Giro Derecha 90º-Adelante-Giro Izquierda 90º-Adelante	0
	1	Giro Izquierda 90º-Adelante-Giro Izquierda 90º-Adelante	
	2	Adelante-Giro Izquierda 90º-Adelante	
	3	Giro 180º-Adelante-Giro Izquierda 90º-Adelante	

Elaborado por: José Arias y Alex García

Tabla 3.4 Movimientos para trasladarse a una posición atrás diagonal derecha

Posición Atrás Diagonal Derecha			
Ruta por donde seguir	**Vista Actual**	**Acción**	**Vista Siguiente**
Atrás (si tiene un valor de 1 en la posición atrás)	0	Giro 180º-Adelante-Giro Izquierda 90º-Adelante	2
	1	Adelante-Giro Izquierda 90º-Adelante	
	2	Giro Derecha 90º-Adelante-Giro Izquierda 90º-Adelante	
	3	Giro Izquierda 90º-Adelante-Giro Izquierda 90º-Adelante	
Derecha (si tiene un valor de 100 en la posición atrás)	0	Giro Derecha 90º-Adelante-Giro Derecha 90º-Adelante	1
	1	Giro Izquierda 90º-Adelante-Giro Derecha 90º-Adelante	
	2	Adelante-Giro Derecha 90º-Adelante	
	3	Giro 180º-Adelante-Giro Derecha 90º-Adelante	

Elaborado por: José Arias y Alex García

Tabla 3.5 Movimientos para trasladarse a una posición atrás diagonal izquierda.

Posición Atrás Diagonal Izquierda			
Ruta por donde seguir	**Vista Actual**	**Acción**	**Vista Siguiente**
Atrás (si tiene un valor de 1 en la posición atrás)	0	Giro 180º-Adelante-Giro Derecha 90º-Adelante	3
	1	Adelante-Giro Derecha 90º-Adelante	
	2	Giro Derecha 90º-Adelante-Giro Derecha 90º-Adelante	
	3	Giro Izquierda 90º-Adelante-Giro Derecha 90º-Adelante	
Izquierda (si tiene un valor de 100 en la posición atrás)	0	Giro Izquierda 90º-Adelante-Giro Izquierda 90º-Adelante	1
	1	Giro Derecha 90º-Adelante-Giro Izquierda 90º-Adelante	
	2	Giro 180º-Adelante-Giro Izquierda 90º-Adelante	
	3	Adelante-Giro Izquierda 90º-Adelante	

Elaborado por: José Arias y Alex García

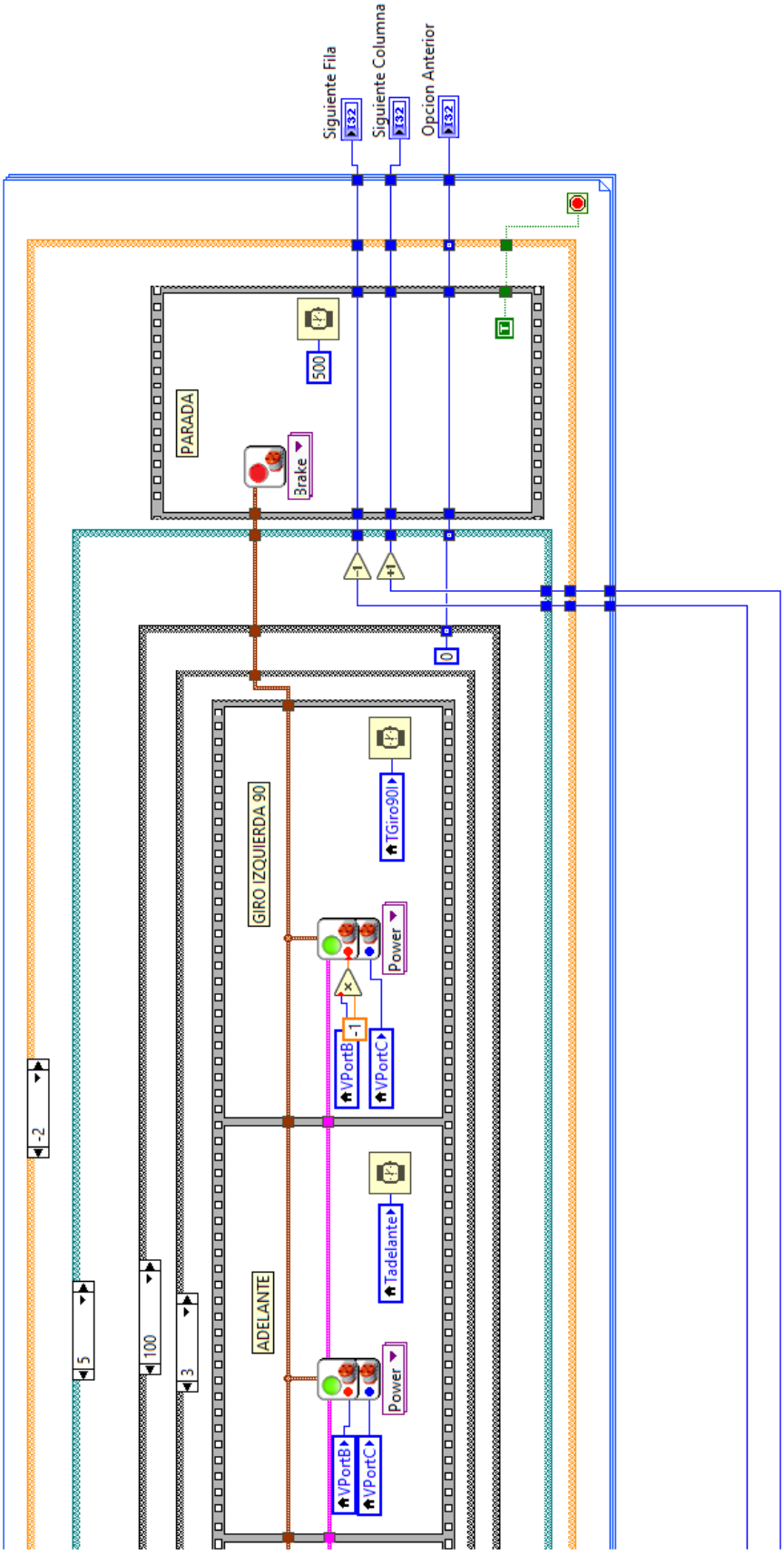
Tabla 3.6 Asignación de velocidades de los motores para diferentes movimientos.

Dirección del Robot	Velocidad Motor Izquierdo (%)	Velocidad Motor Derecho (%)
Adelante	+30	+27
Atrás	-30	-27
Derecha	+30	-27
Izquierda	-30	+27
Parado	0	0

Elaborado por: José Arias y Alex García

Una vez implementada la lógica para la asignación de los valores y signos de las velocidades, se actualizan los nuevos valores de la columna y la fila correspondientes a la nueva posición donde se trasladó el vehículo. Esta configuración es programada al final de la estructura y permite acceder el índice del arreglo que almacena las posibles posiciones de desplazamiento, como se indica en la Figura 3.29.

Figura 3.29 Código final del subVI Direccionamiento.

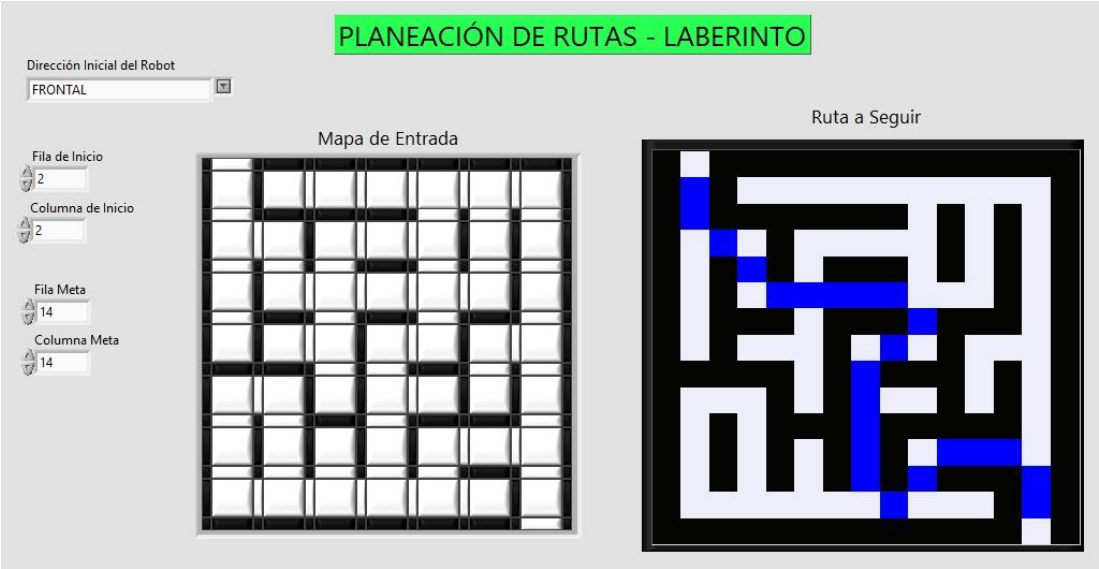


Elaborado por: José Arias y Alex García

3.4.2. Algoritmo de Programación del VI Implementación

En este VI se establece de forma virtual la configuración física del laberinto indicando mediante cuatro controles numéricos la coordenada de inicio y la coordenada de meta a la cual debe llegar el vehículo móvil. Además, en el panel frontal mostrado en la Figura 3.30, se observa el laberinto con la ruta más optima y de menor costo que debe ejecutar el vehículo

Figura 3.30 Panel frontal del VI implementación



Elaborado por: José Arias y Alex García

El código de programación del *VI Implementación* se presenta en la Figura 3.31. El *Mapa de Entrada* representado por el *cluster* que contiene estados booleanos y que representan la configuración del laberinto es transferido al interior del primer *frame* de la estructura *Flat Sequence,* donde es convertido a un arreglo unidimensional empleando el bloque *Cluster To Array*. Este arreglo unidimensional de 196 elementos debe ser transformado en un arreglo de 14x14, para ello se implementa una lógica de conversión utilizando dos bucles *For* y con una matriz previamente definida e inicializada con estados booleanos False se reemplazan los valores booleanos guardados en el arreglo unidimensional.

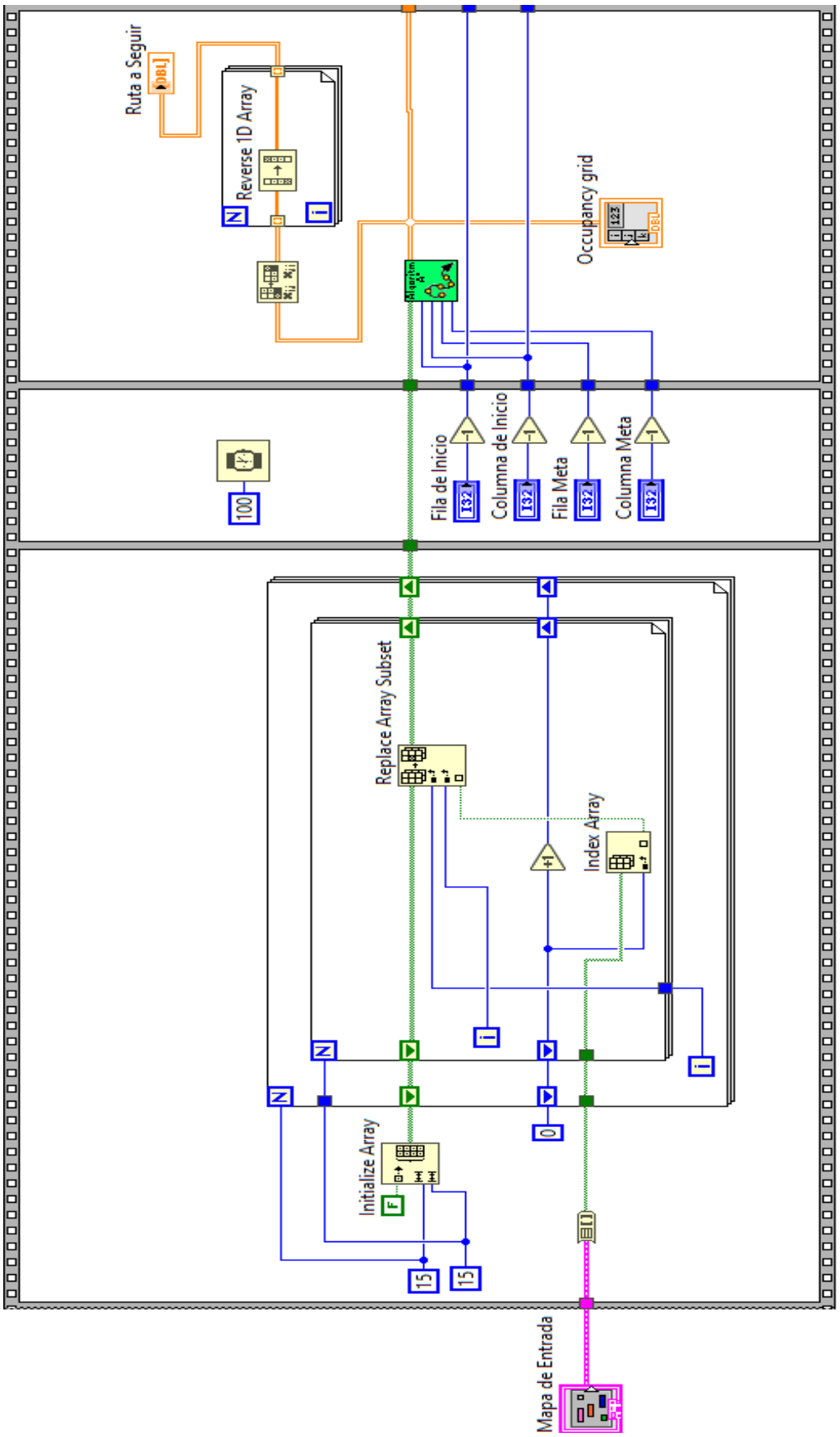
Dentro del segundo *frame* de la estructura *Flat Sequence* se programa un retardo de 100 ms y se especifican los valores de las coordenadas de inicio y llegada en el laberinto. Tal y como se muestra en la Figura 3.31 a cada control numérico ubicado dentro del *frame* se le ha restado una unidad, esta operación se la lleva a cabo debido a que el usuario establece en el panel frontal las coordenadas con números del 1 al 14, pero en LabView la ubicación

de los elementos dentro de un arreglo 2D (filas y columnas) se le especifica desde el índice 0.

Dentro del tercer *frame* de la estructura *Flat Sequence* se utiliza el *VI Algoritmo A-Star* que permite encontrar el mejor camino que el vehículo móvil ejecutara en el laberinto, dada las coordenadas inicial y finales ingresadas en el panel frontal del *VI Implementación*. Dentro del mismo *frame* se emplea un indicador gráfico *Intensity Graph* que permite representar el laberinto y el camino definido por al algoritmo A-Star. La salida del *VI Algoritmo A-Star* es una matriz bidimensional que almacena los valores que representan a los espacios ocupados (100), espacios vacíos (1) y los espacios seleccionados por los cuales debe trasladarse el vehículo (-2).

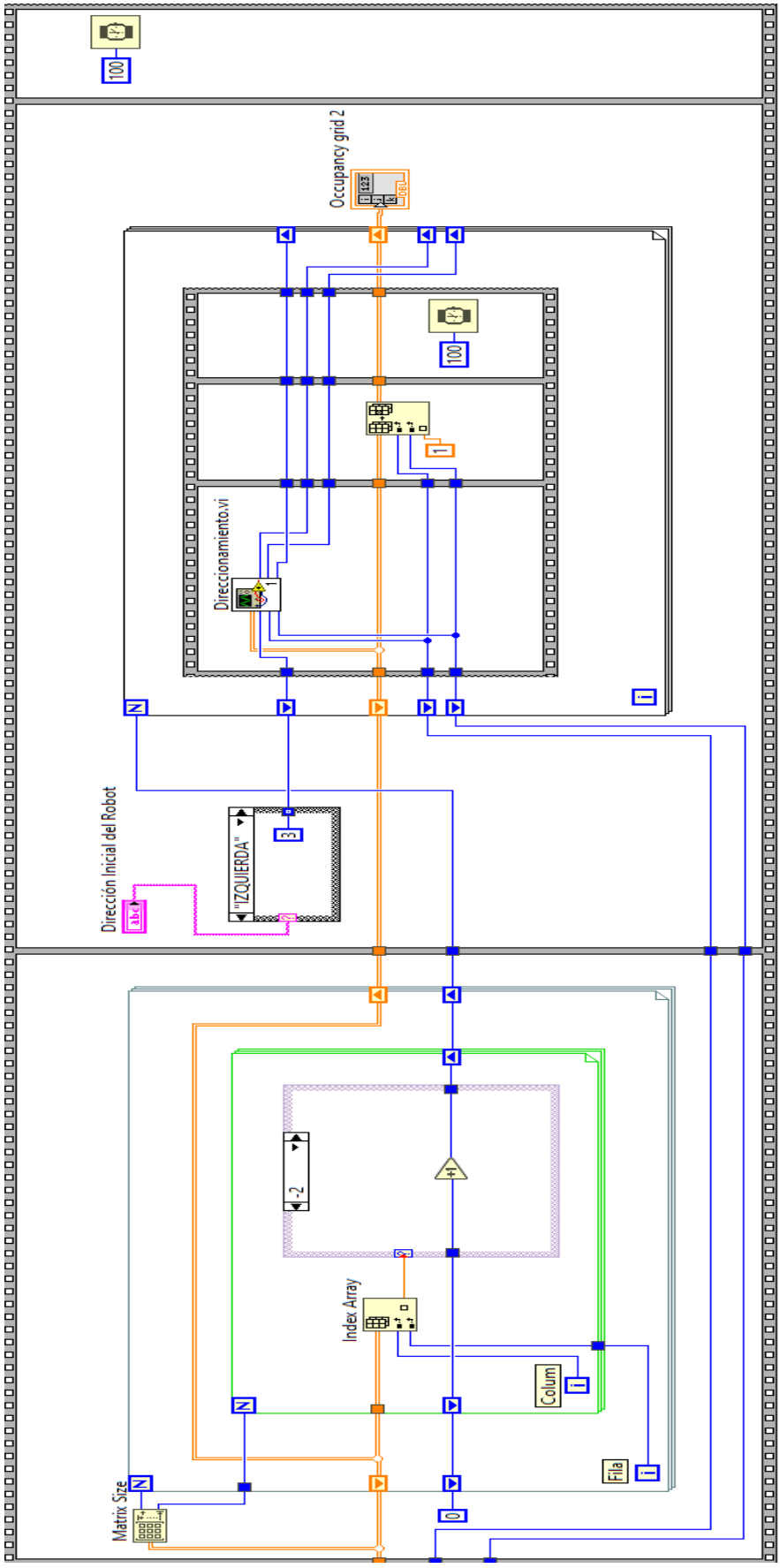
Al interior del cuarto *frame* (como se observa en la Figura 3.32) se realiza un barrido de los valores almacenados dentro de la matriz bidimensional obtenida anteriormente. Cada vez que se detecte un valor numérico de -2 se aumenta en una unidad el valor de un contador y una vez generado todo el barrido del arreglo bidimensional se paran los bucles *For* y se consigue como resultado el número total de posiciones que debe ejecutar el vehículo móvil.

Figura 3.31 Primer fragmento del diagrama de bloques del VI Implementación.



Elaborado por: José Arias y Alex García

Figura 3.32 Segundo fragmento del diagrama de bloques del VI Implementación.



Elaborado por: José Arias y Alex García

CAPÍTULO 4

PRUEBAS – RESULTADOS

En este capítulo se presentan los resultados obtenidos una vez que se realizaron diversas pruebas de funcionamiento de los algoritmos implementados sobre el robot móvil EV3 y el laberinto físico, tanto para el algoritmo tipo Insecto como el A-Star. Se indicarán los tiempos de ejecución obtenidos en la determinación de las rutas de salidas para cada algoritmo y se verificará que algoritmo resuelve el laberinto con un menor costo, tanto en tiempo como en distancia recorrida.

4.1. Prueba Sobre Laberinto Físico – Primera Ruta

Para la verificación de la efectividad de los algoritmos de planeación de rutas en la resolución del laberinto se construyó un escenario controlado con diferentes alternativas de rutas de salidas y con medidas estandarizadas utilizadas en diferentes concursos de robótica a nivel local e internacional, como se observan en la Figura 4.1. El laberinto está formado por un área de 2.1x2.1 m^2 de color blanco, sus paredes tienen un espesor de 6 mm y las divisiones internas tienen un ancho de 30 cm y un alto de 15 cm (con una tolerancia del $\pm$5% en todas las dimensiones dadas).

Figura 4.2 Laberinto Físico

Elaborado por: José Arias y Alex García

Las pruebas ejecutadas sobre el laberinto se realizaron considerando que las velocidades lineales de los dos motores del robot móvil EV3 se establecieron en un 30% de su potencia nominal, dicha elección se debe al tiempo de respuesta de los sensores incorporados en el robot como son los sensores de proximidad (longitud de onda típica de 820 a 880 nm) y particularmente el sensor giroscópico (rango de escala completa de 440°/s).

En la Tabla 4.1 se indica los resultados obtenidos de los parámetros de evaluación, obtenidos para las 10 pruebas realizadas sobre el laberinto físico ejecutando el algoritmo tipo Insecto sobre el robot móvil EV3.

Tabla 4.1 Resultados del algoritmo tipo Insecto – Primera ruta

Prueba #	**Tiempo (s)**	**Distancia(cm)**	**Porcentaje de trayectoria completada (%)**
1	133	886	100
2	143	886	100
3	146	886	100
4	139	886	100
5	137	886	100
6	137	886	100
7	137	886	100
8	151	886	100
9	142	886	100
10	147	886	100
Promedios	141.2	886	100

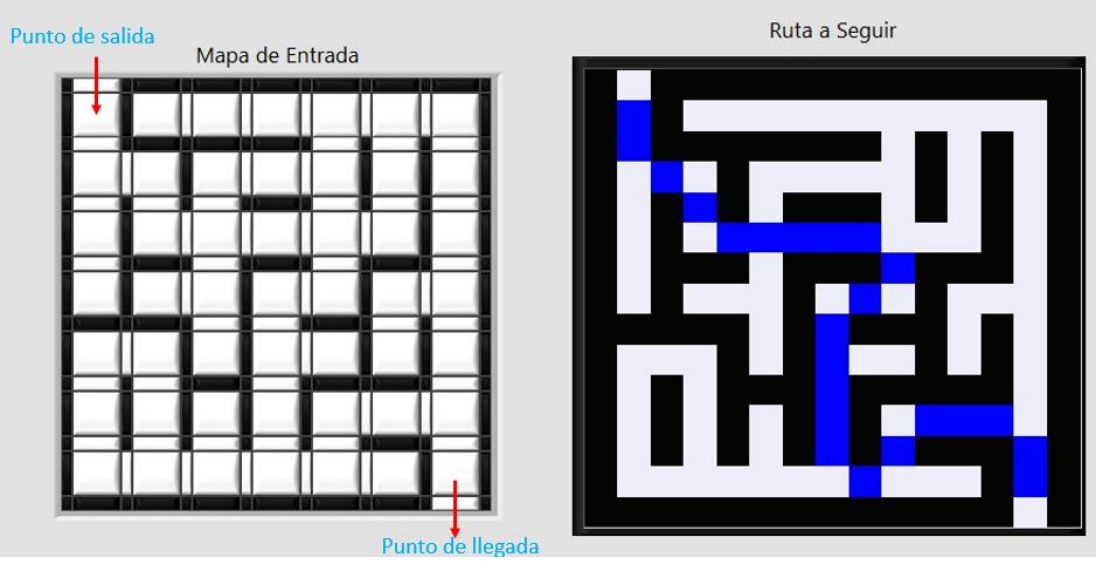
Elaborado por: José Arias y Alex García

De los valores obtenidos en Tabla 4.1, se puede evidenciar que el robot móvil EV3 recorre en todas las pruebas la misma ruta, con un tiempo promedio de 141.2 segundos y una tolerancia ± 9 segundos en cada prueba, debido a diferentes factores como el rozamiento de las llantas y el nivel de la batería. Al ejecutar la misma ruta para resolver el laberinto la distancia recorrida en todas las pruebas resultó ser la misma con un valor de 886 cm y con un tiempo promedio de llegada de 141.2 segundos.

Antes de ejecutar el algoritmo A-Star sobre el robot móvil EV3 localizado en el laberinto físico, se debe de ejecutar previamente el VI Implementación explicado en el apartado 3.4.3 para obtener la ruta de menor costo que debe recorrer el robot. Esta ruta es visualizada en la Figura 4.2, donde muestra los puntos de partida y llegada del robot, así como la ruta suministrada por el algoritmo A-Star.

En la Tabla 4.2 se indica los resultados obtenidos de los parámetros de evaluación, obtenidos para las 10 pruebas realizadas sobre el laberinto físico ejecutando el algoritmo tipo A-Star sobre el robot móvil EV3.

Figura 4.2 Ruta proporcionada por el algoritmo A-Star



Elaborado por: José Arias y Alex García

Tabla 4.2 Resultados del algoritmo tipo A-Star – Primera ruta

Prueba #	**Tiempo (s)**	**Distancia(cm)**	**Porcentaje de trayectoria completada (%)**
1	42	457	100
2	45	457	100
3	44	457	100
4	48	457	100
5	39	457	100
6	40	457	100
7	42	457	100
8	42	457	100
9	43	457	100
10	47	457	100
Promedios	43.2	457	100

Elaborado por: José Arias y Alex García

De los valores obtenidos en Tabla 4.2, se puede evidenciar que el robot móvil EV3 recorre en todas las pruebas la ruta indicada en la Figura 4.2, con un tiempo promedio de 43.2 segundos y una tolerancia ± 3.5 segundos en cada prueba, debido a diferentes factores como el rozamiento de las llantas y el nivel de la batería. Al ejecutar la misma ruta para resolver el laberinto la distancia recorrida en todas las pruebas resultó ser la misma con un valor de 457 cm y con un tiempo promedio de llegada de 43.2 segundos.

4.2. Prueba Sobre Laberinto Físico – Segunda Ruta

En la Tabla 4.3 se indican los resultados obtenidos de los parámetros de evaluación, obtenidos para las 10 pruebas realizadas sobre el laberinto físico ejecutando el algoritmo tipo Insecto sobre el robot móvil EV3.

Tabla 4.3 Resultados del algoritmo tipo Insecto – Segunda ruta

Prueba #	**Tiempo (s)**	**Distancia(cm)**	**Porcentaje de trayectoria completada (%)**
1	85	663	100
2	81	663	100
3	92	663	100
4	87	663	100
5	93	663	100
6	91	663	100
7	85	663	100
8	87	663	100
9	86	663	100
10	84	663	100
Promedios	87.1	663	100

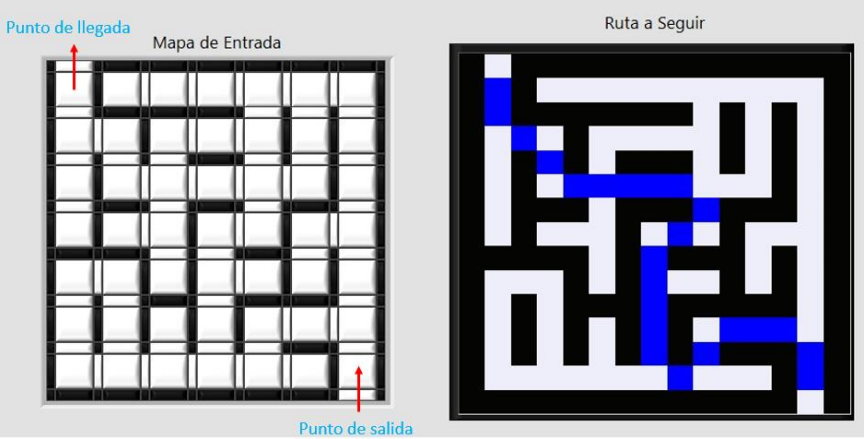
Elaborado por: José Arias y Alex García

De los valores obtenidos en Tabla 4.3, se puede evidenciar que el robot móvil EV3 recorre en todas las pruebas la misma ruta, con un tiempo promedio de 87.1 segundos y una tolerancia ± 6 segundos en cada prueba. Al ejecutar la misma ruta para resolver el laberinto la distancia recorrida en todas las pruebas resultó ser la misma con un valor de 663 cm y con un tiempo promedio de llegada de 87.1 segundos.

Antes de ejecutar el algoritmo A-Star sobre el robot móvil EV3 localizado en el laberinto físico, se debe de ejecutar previamente el VI Implementación explicado en el apartado 3.4.3 para obtener la ruta de menor costo que debe recorrer el robot. Esta ruta es visualizada en la Figura 4.3, en donde indica los puntos de partida y llegada del robot, así como la ruta suministrada por el algoritmo A-Star.

Como se muestra en la Tabla 4.4 los resultados obtenidos de los parámetros de evaluación, obtenidos para las 10 pruebas realizadas sobre el laberinto físico ejecutando el algoritmo tipo A-Star sobre el robot móvil EV3.

Figura 4.3 Ruta proporcionada por el algoritmo A-Star



Elaborado por: José Arias y Alex García

Tabla 4.4 Resultados del algoritmo tipo A-Star – Segunda ruta

Prueba #	**Tiempo (s)**	**Distancia(cm)**	**Porcentaje de trayectoria completada**
1	42	457	100
2	43	457	100
3	42	457	100
4	41	457	100
5	45	457	100
6	48	457	100
7	43	457	100
8	45	457	100
9	43	457	100
10	44	457	100
Promedios	43.6	457	

Elaborado por: José Arias y Alex García

De los valores obtenidos en Tabla 4.4, se puede evidenciar que el robot móvil EV3 recorre en todas las pruebas la ruta indicada en la Figura 4.3, con un tiempo promedio de 43.6 segundos y una tolerancia ± 3.5 segundos en cada prueba. Al ejecutar la misma ruta para resolver el laberinto la distancia recorrida en todas las pruebas resultó ser la misma con un valor de 457 cm y con un tiempo promedio de llegada de 43.6 segundos.

Al realizar una comparación entre los resultados y promedios obtenidos en las diferentes tablas al ejecutar los dos algoritmos de planeación de rutas programados sobre el robot móvil EV3, se puede evidenciar que se obtuvieron menores valores en cuanto a tiempo y distancia recorrida mediante el algoritmo A-Star.

CONCLUSIONES

Al comparar los resultados contenidos en la Tabla 4.1 hasta la Tabla 4.4, se puede evidenciar que el algoritmo de planeación de rutas A-Star determina la solución del laberinto con menores valores en cuanto a tiempo y distancia recorrida en comparación del algoritmo tipo Insecto, para las diferentes pruebas realizadas en las dos posiciones de inicio del entorno físico.

En las pruebas realizadas para la primera ruta del laberinto empleando el algoritmo A-Star se obtuvo una distancia recorrida por el robot móvil de 457 cm con un tiempo promedio de llegada de 43.2 segundos, en contraparte mediante la aplicación del algoritmo tipo Insecto la distancia resultó ser de 886 cm con un tiempo promedio de llegada de 141.2 segundos, es decir, que cuando el robot móvil está programado con el algoritmo tipo Insecto recorre 1.74 veces más de distancia (796÷457) que cuando está ejecutando el algoritmo A-Star y se demora 3.27 veces más de tiempo (141.2÷43.2).

En las pruebas realizadas para la segunda ruta del laberinto empleando el algoritmo A-Star se obtuvo una distancia recorrida por el robot móvil de 457 cm con un tiempo promedio de llegada de 43.6 segundos, en contraparte mediante la aplicación del algoritmo tipo Insecto la distancia resultó ser de 663 cm con un tiempo promedio de llegada de 87.1 segundos, es decir, que cuando el robot móvil está programado con el algoritmo tipo Insecto recorre 1.45 veces más de distancia (663÷457) que cuando está ejecutando el algoritmo A-Star y se demora 2 veces en culminar la ruta (87.1÷43.6).

El fundamento teórico del principio de funcionamiento de todos los elementos que conforman al robot diferencial móvil EV3, especialmente el conocimiento del manejo y configuración de los sensores de proximidad, el sensor giroscópico y el bloque inteligente EV3 permitieron adquirir un conocimiento total a nivel del hardware del robot móvil, que combinado con el conocimiento del modelo cinemático del robot facilitaron la implementación de los dos algoritmos de planeación de rutas.

Mediante el empleo del software LabView, así como los módulos adicionales Robotics y Lego Mindstorms, se logró implementar la lógica de programación y las interfases gráficas de los diferentes subVIs desarrollados, utilizando para ello diferentes estructuras de selección y repetición (como por ejemplo while loop, for loop, case estructure, flaq sequence, entre otras), así como las diferentes funciones de la paleta Mindstorms Robotics para adquirir y enviar datos entre LabView y el controlador EV3 del robot.

En el desarrollo del algoritmo A-Star, se debió implementar varios subVIs en el software LabView con la finalidad de disminuir el tamaño del código debido a las diferentes acciones que se debían programar. Inicialmente se implementó el subVI A-Star el cual contiene toda la lógica correspondiente a este algoritmo de planeación de rutas y que proporciona como resultado la ruta de menor costo del entorno tomado en consideración. También se implementó el subVI Direccionamiento cuyo objetivo es el de interpretar la ruta proporcionada por el algoritmo A-Star y posteriormente ejecutarla mediante la activación adecuada de los motores del robot haciendo uso de las funciones del módulo Lego Mindstorms. Finalmente se desarrolló el VI Implementación el cual contiene los dos subVIs antes mencionados y además se ha desarrollado la interfaz gráfica que simula el laberinto físico y en donde se muestra la ruta obtenida mediante el algoritmo A-Star.

RECOMENDACIONES

Antes de ejecutar programas de cualquier tipo sobre el robot vehículo EV3, se recomienda revisar todos los fundamentos teóricos mencionados en el Capítulo 2 en donde se detalla el funcionamiento de los componentes que conforman al robot (sensores, motores y bloque inteligente), así como iniciar la comunicación inalámbrica entre el ordenador y el bloque EV3 siguiendo los pasos indicados en el apartado 3.2.

Se debe verificar constantemente que el nivel de la batería del robot móvil este por encima del 70% de carga para obtener mejores resultados en las distintas pruebas aplicando los dos algoritmos de planeación de rutas.

Considerando que el robot móvil EV3 cuenta con un sensor infrarrojo para detectar las paredes laterales del laberinto, este puede llegar a topar estas paredes debido a cambios de iluminación del entorno, por dicha razón se recomienda modificar la programación en cuanto a la distancia detectada para resolver este inconveniente.

Se recomienda configurar la potencia de los motores en un 30% cuando se aplica el algoritmo A-Star sobre el robot móvil, debido a que con una potencia mayor la velocidad de respuesta del sensor giroscópico resulta inadecuada y se obtendrían valores erróneos en la medida y ejecución de los giros programados sobre el chasis del robot.

Hay que recalcar que, hasta la fecha de la presentación de este proyecto, el entorno gráfico de programación LabView solo puede interactuar con el bloque EV3 utilizando el módulo Lego Mindstorms 2016. Se espera que dicho módulo sea actualizado para futuras versiones de LabView.

REFERENCIAS

An Introduction to A Path Planning (using LabVIEW) - NI Community*. (2017). https://forums.ni.com/t5/LabVIEW-Robotics-Documents/An-Introduction-to-A-Path-Planning-using-LabVIEW/ta-p/3521668?profile.language=en

Girosensor EV3 - EsMindstorms. (2019). https://www.esmindstorms.com/girosensor-ev3/

Gregor Klancar Andrej Zdesar Saso Blazic Igor Skrjanc. (2017). *Wheeled Mobile Robotics Wheeled Mobile*.

LEGO® MINDSTORMS® EV3 31313 - Sets LEGO® MINDSTORMS® - LEGO.com para niños. (2019). https://www.lego.com/es-ar/kids/sets/mindstorms/mindstorms-ev3-0729302d5ae04b5d88a30c2dd6d7afba

Tzafestas, S. G. (2014). *Introduction to mobile robot control*. Elsevier.

Vizcaíno Juan. (2017). *"Desarrollo de un prototipo de robot educacional tipo Segway con control remoto.*

Aguilar Campos, J. P., & Sandoval Suquillo, H. I. (2018). *Desarrollo de un robot móvil diferencial controlado mediante un algoritmo de búsqueda con redes neuronales.* Quito: Universidad Politécnica Salesiana - Carrera de Ingeniería Electrónica.

Aldaz Andrade, L. A., & Orellana Torres, R. D. (2017). *Desarrollo de un prototipo de robot móvil de competencia multi-categoría.* Quito: Universidad Politécnica Salesiana - Carrera de Ingeniería Electrónica.

Barahona Guamani, E. S. (2019). *Navegación autónoma basada en maniobras bajo estimación de posturas humanas para un robot omnidireccional Kuka YouBot.* Ambato: Universidad Técnica de Ambato, Ingeniería en Sistemas, Electrónica e Industrial.

Baturone, A. O. (2001). *Robótica: Manipuladores y Robots Móviles.* Barcelona: Marcombo BOIXAREU.

Benavent Pla, A. (2017). *Vehículo autónomo con visión artificial utilizando OpenCV.* Valencia: Universitat Politècnica de València. Departamento de Ingeniería Electrónica - Departament d'Enginyeria Electrònica.

Buendia Ríos, A. A. (2017). *Navegación Autónoma de un vehículo Pequeño en Interiores Empleando Visión Artificial y Diferentes Sensores.* Texcoco: Universidad Autónoma del Estado de México.

CEDIA. (28 de Noviembre de 2019). *CER 2019.* Recuperado el 14 de Enero de 2021, de Reglamento Seguidor De Linea Destreza: https://cedia.edu.ec/dmdocuments/CER/2019/Reglamentos/ReglamentoSeguidorDeLineaDestreza.pdf

Cortéz, F. R. (2011). *Robótica CONTROL DE ROBOTS MANIPULADORES.* México: Alfaomega .

EdgeImpulse. (6 de Diciembre de 2020). *EDGE IMPLUSE*. Obtenido de EdgeImpulse Inc.: https://www.edgeimpulse.com/

Hossian, A., Cejas, L., Carbajal, R., Echeverría, C., Alveal, M., & Olivera, V. (2018). *Obtención de rutas de navegación óptimas de robot móvil mediante la aplicación de Redes Neuronales Artificiales (RNA) y Algoritmos de Búsqueda Local (ABL) en entornos estructurados.* Tucumán: Grupo de Investigación de Robótica - Universidad Tecnológica Nacional – Facultad Regional del Neuquén.

Ramírez Díaz, J. A. (2019). *Aprendizaje del Comportamiento Humano en el Espacio Articular del Robot Utilizando Redes Neuronales*. Ciudad de México: Centro de Investigacion y Estudios Avanzados del Instituto Politecnico Nacional, Unidad Zacatenco, Departamento de Control Automático.

Rida, B., Bhawani, C., & Reehan, S. (2018). PSO Based Localization of Multiple Mobile Robots. *IEEE, I*(2), 5.

Suárez Romero, A., & del Pozo Quintero, A. (2016). Estrategia de navegación para robots móviles mediante redes neuronales. 1.

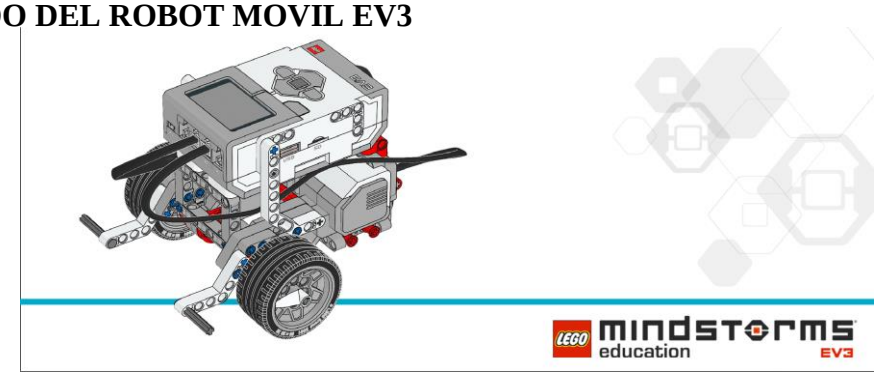
Trabes, E. (2018). *Desarrollo de algoritmos para la exploración submarina mediante vehículos autónomos con visión artificial monocular*. Bahía Blanca: Universidad Nacional del Sur.

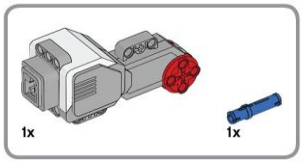
Vera Arenas, J. A., & Alejandro Proaño, E. A. (2016). *Diseño e implementación de dos dos robots seguidores de línea modalidad velocista y destreza para participaciones en concursos de robótica.* Guayaquil: Universidad Católica de Santiago de Guayaquil. Obtenido de http://repositorio.ucsg.edu.ec/handle/3317/5445

ANEXOS

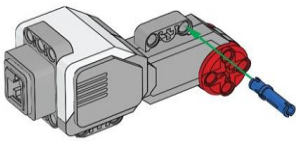
ANEXO 1

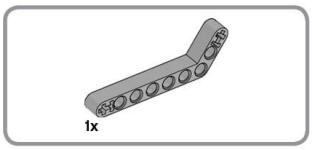
ARMADO DEL ROBOT MOVIL EV3



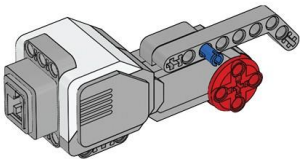


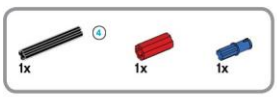
1



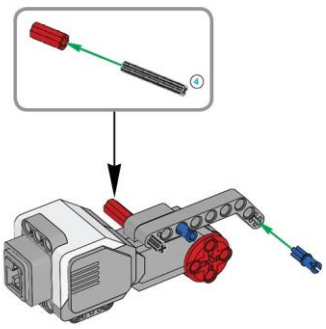


2

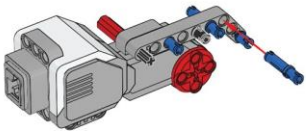


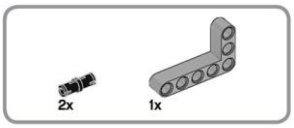


3

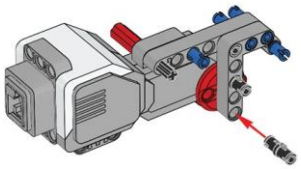


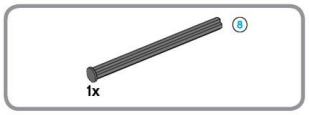
4



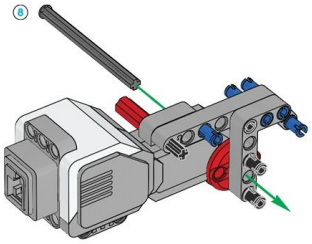


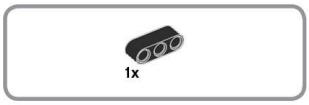
5



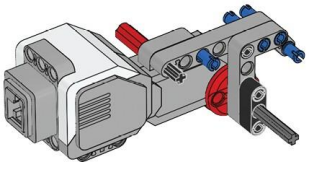


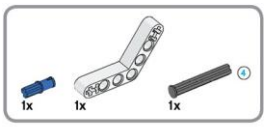
6



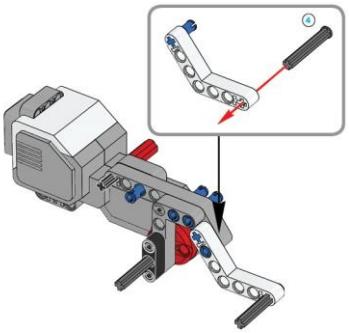


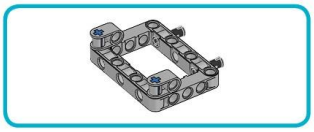
7

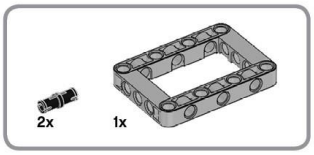




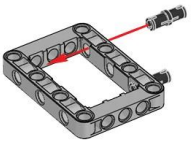
8

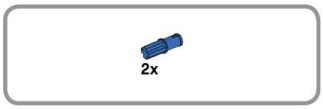




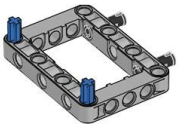


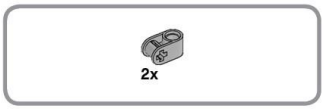
9



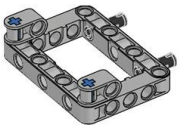


10

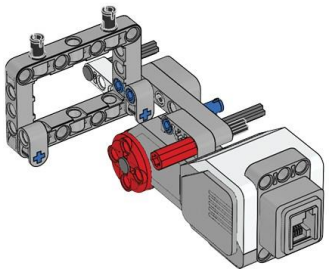


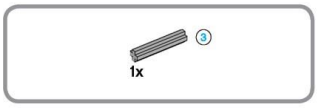


11

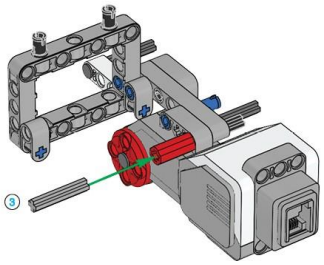


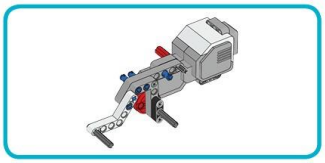
12

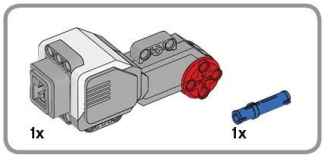


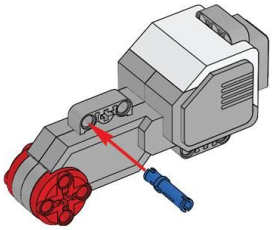


13

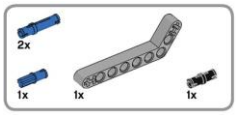




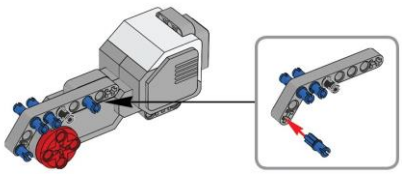


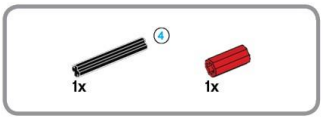


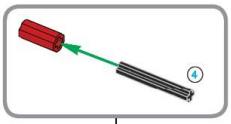
14



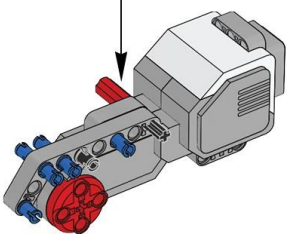
15

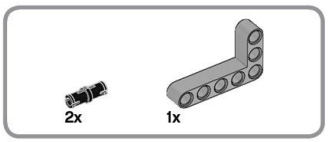




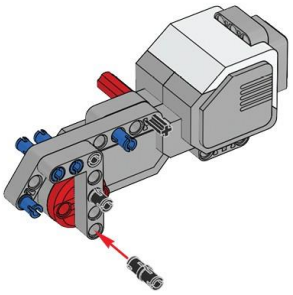


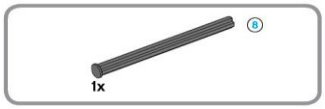
16



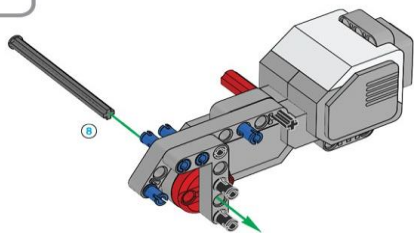


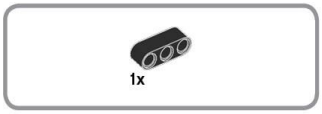
17



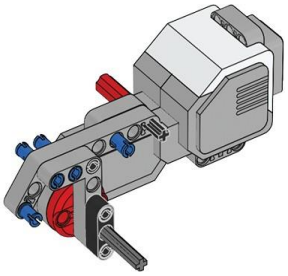


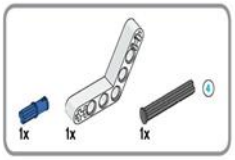
18



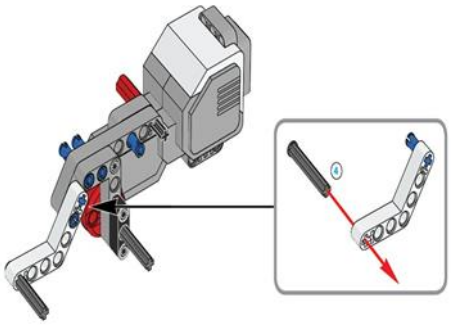


19

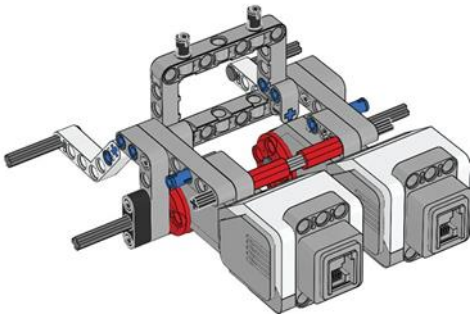


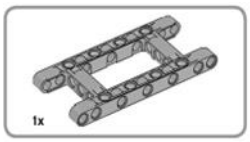


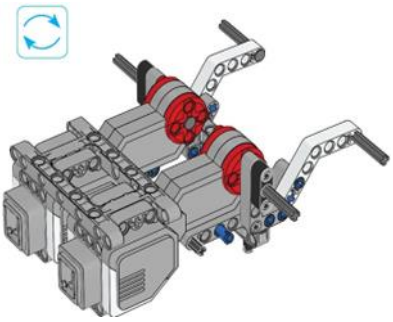
20



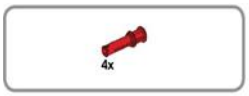
21

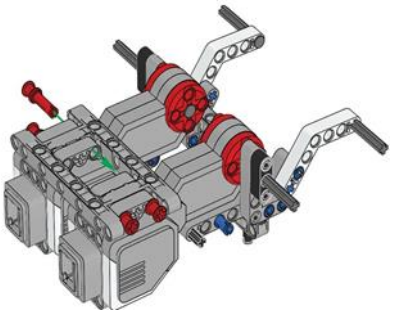




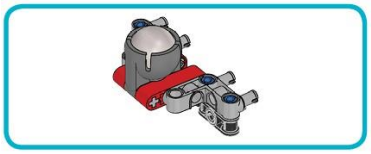


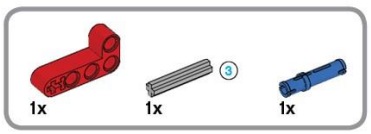
22

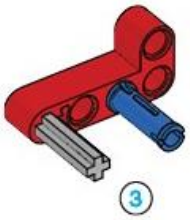




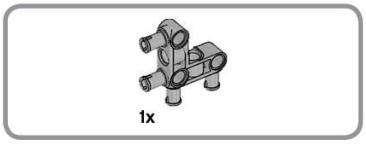
23

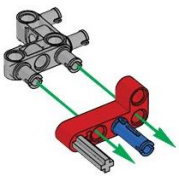




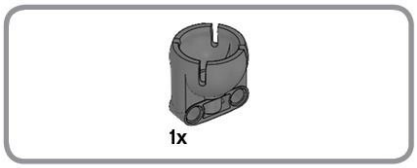


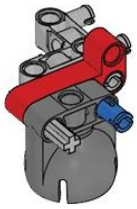
24



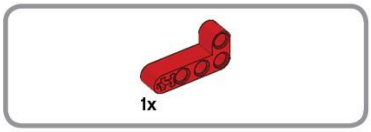


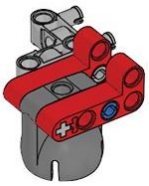
25



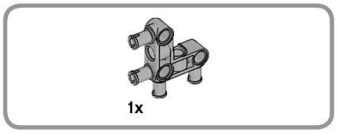


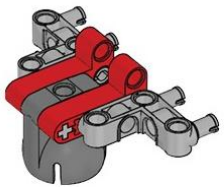
26



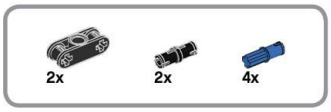


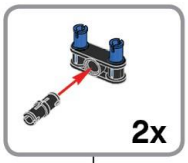
27

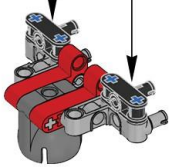




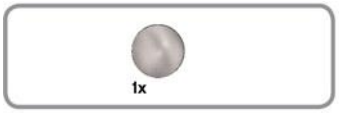
28



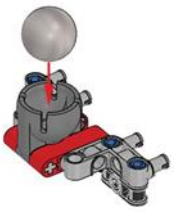




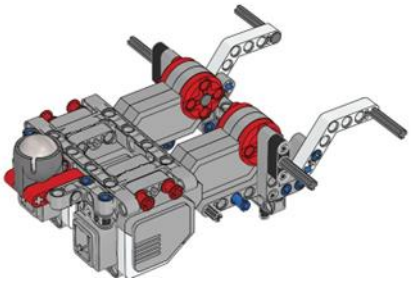
29



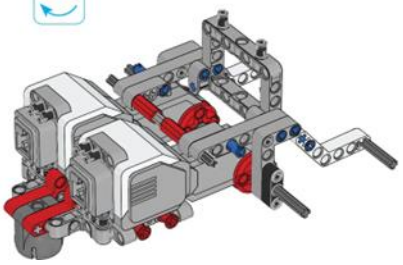
30

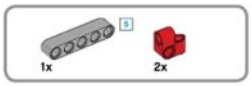


31

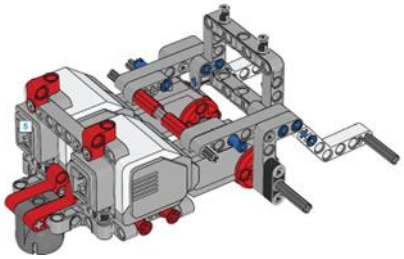


32

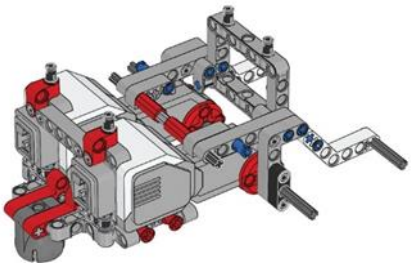


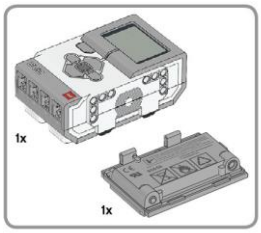


33

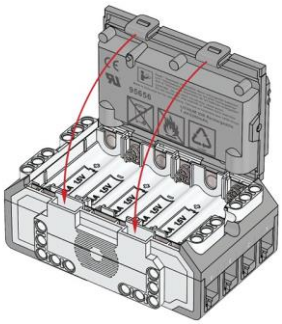


34

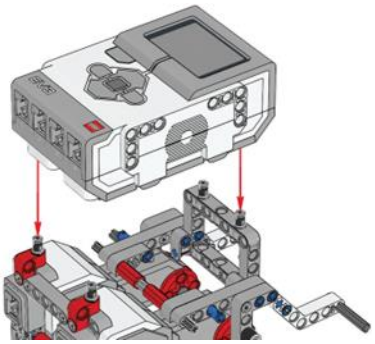


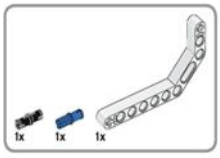


35

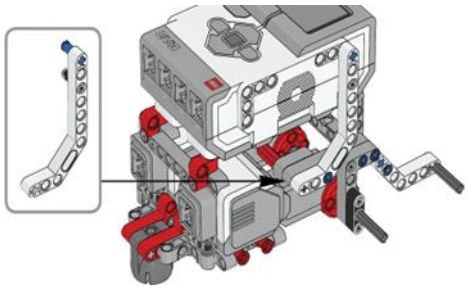


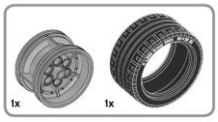
36



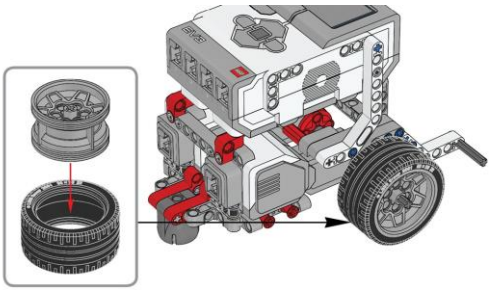


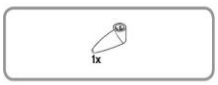
37



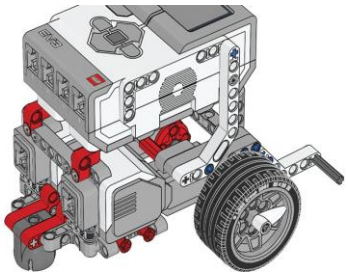


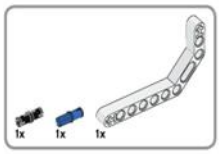
38



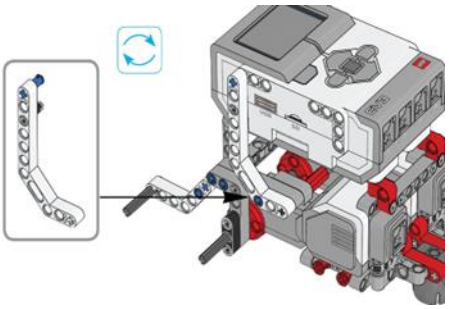


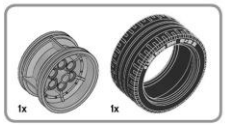
39



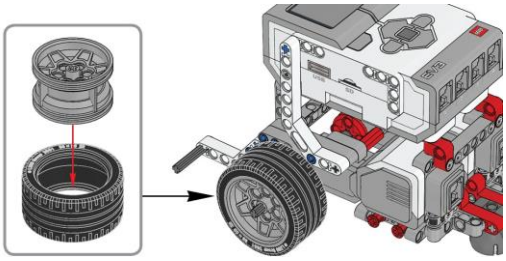


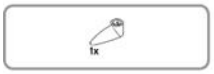
40



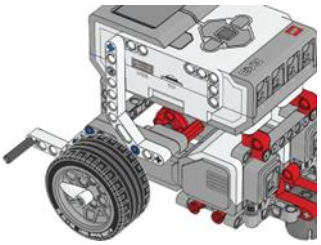


41

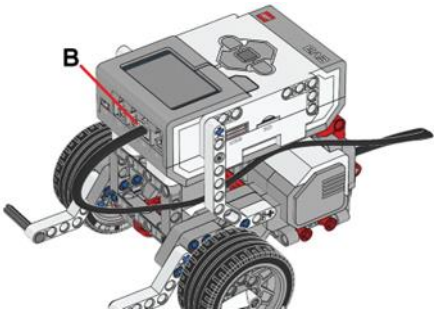




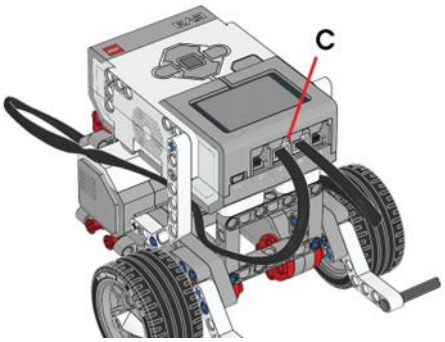
42



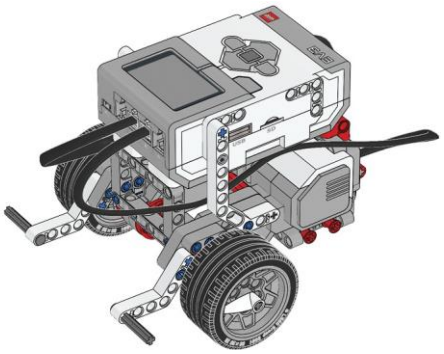
43

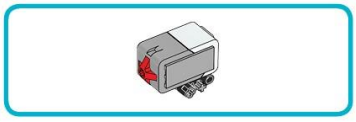


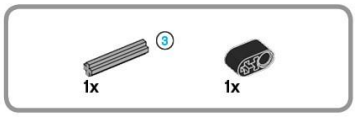
44

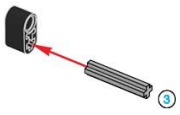


45

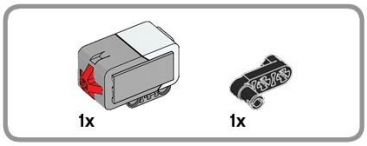




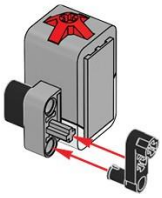




1

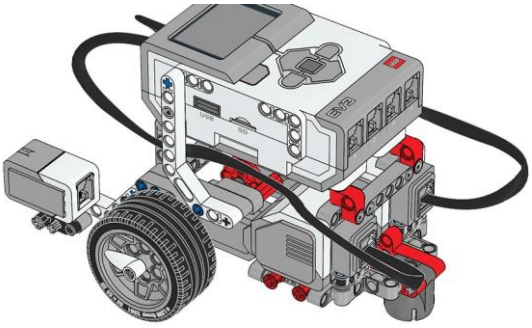


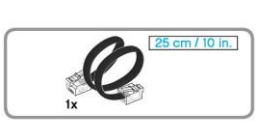
2



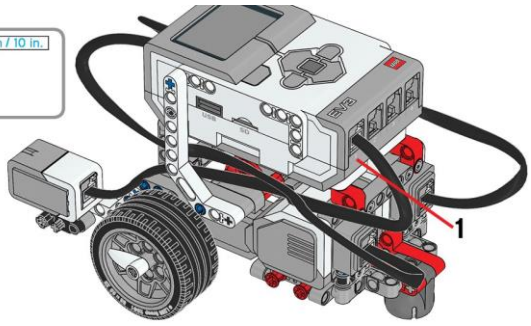
3

4

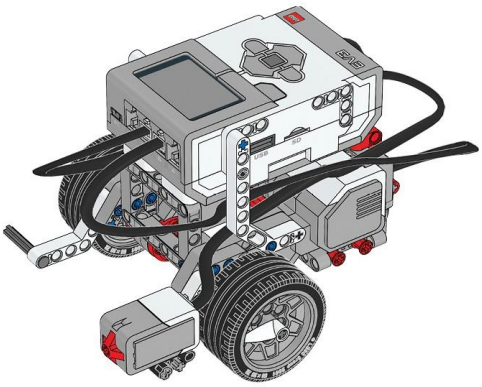


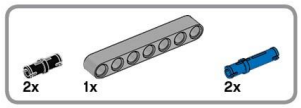


5

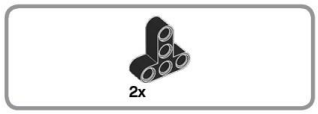


6

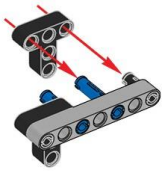


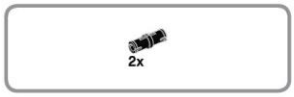


1

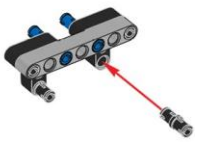


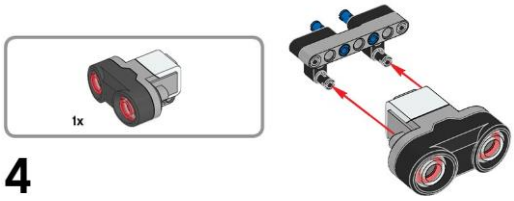
2





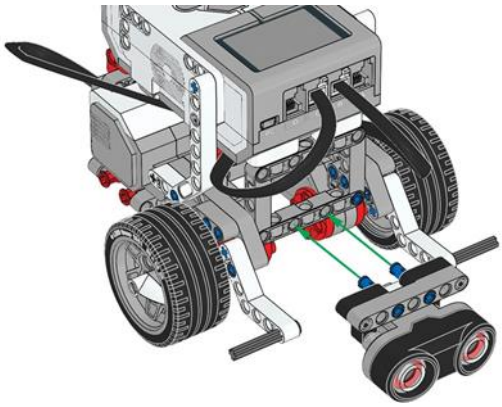
3



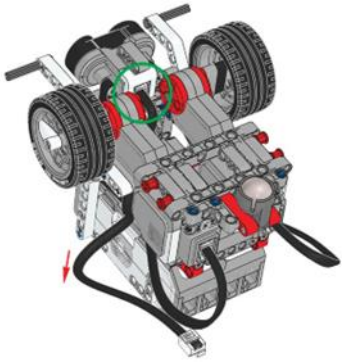


4

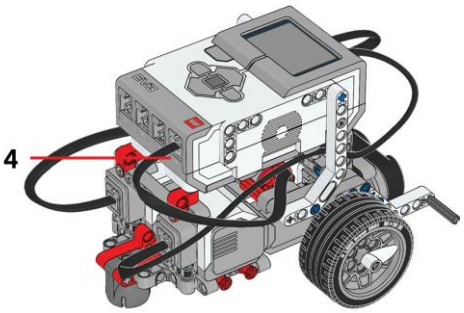
5



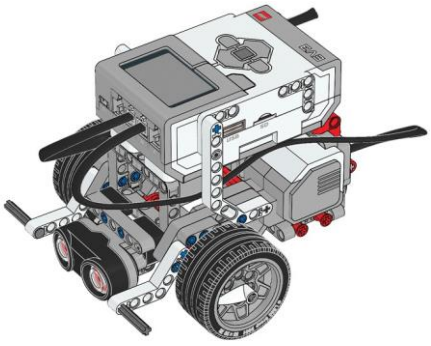
6



7

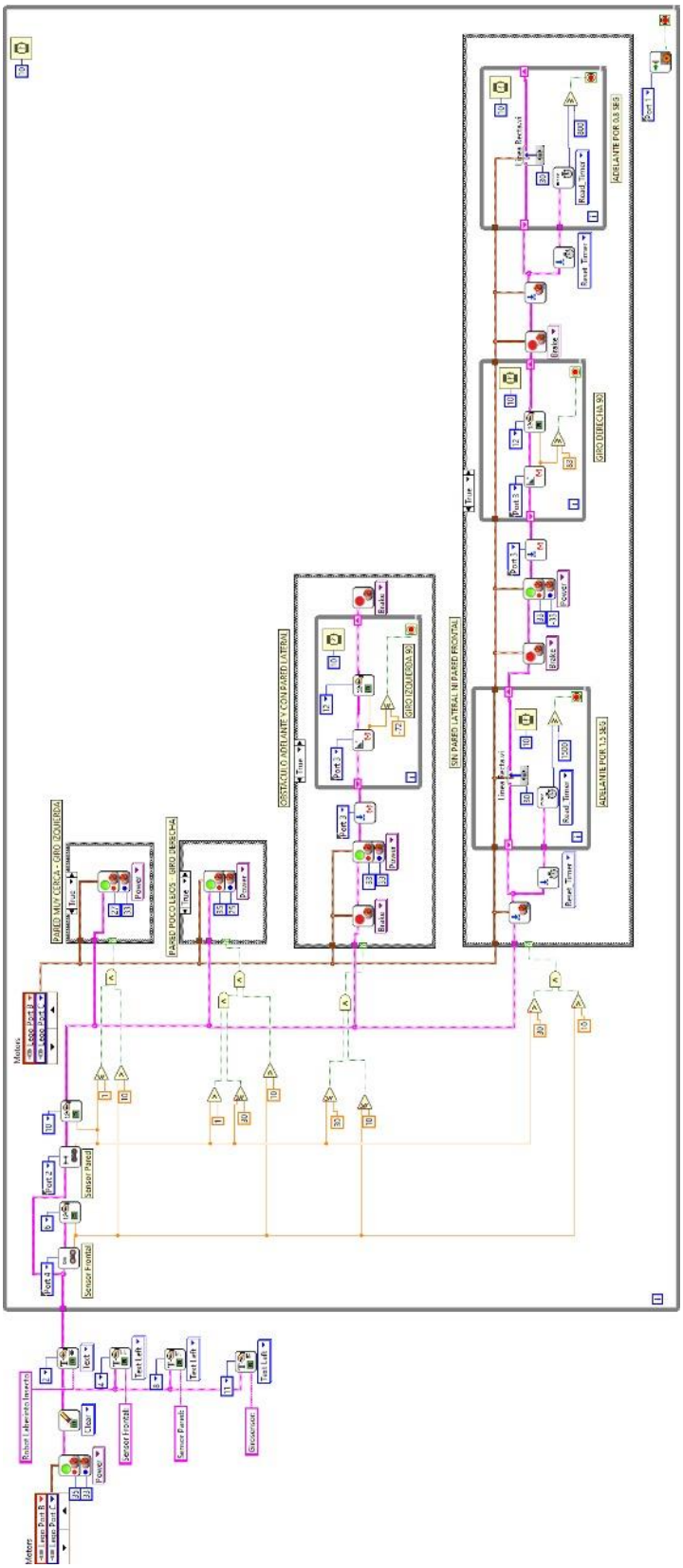


8



ANEXO 2

CÓDIGO DE PROGRAMACIÓN EN LABVIEW DEL ALGORITMO TIPO INSECTO



ANEXO 3

FUNCIONES DE LABVIEW ROBOTICS PARA PLANEACIÓN DE RUTAS

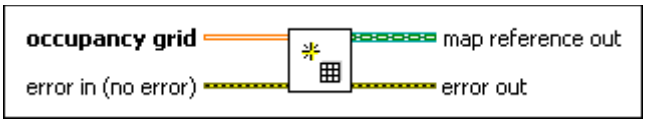
Los VIs empleados para la planeación de rutas se los puede encontrar en la paleta *Robotics* > *Path Planning*, de estos VIs solo se van a emplear los VIs *Occupancy Grid Map* y el VI *A**.

Los VIs *Occupancy Grid Map* se usan para crear y controlar un mapa cuadriculado de ocupación, a través del cual un robot móvil puede navegar. Los mapas cuadriculados de ocupación son útiles para representar ambientes robóticos en donde los programadores conocen el costo de recorrer cada punto del mapa. A continuación, se va a explicar el funcionamiento de los VIs más empleados de este grupo.

Create Occupancy Grid Map

Este VI (Figura A3.1) genera un mapa en forma de una cuadrícula de variables o células, que describen el entorno del robot.

Figura A3.1 VI Create Occupancy Grid Map.



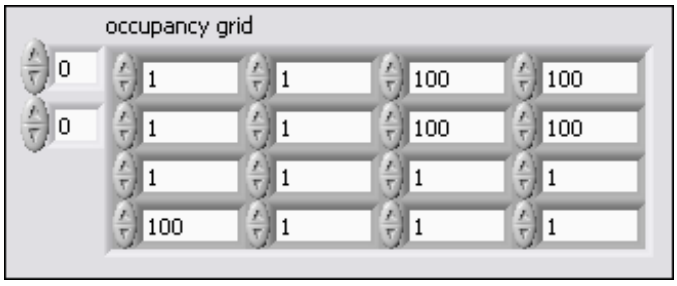
occupancy grid: es un arreglo de números positivos que especifican el costo de entrada en cada celda del mapa cuadriculado de ocupación desde una celda adyacente. Se puede construir una matriz de valores en LabVIEW, o leer una cuadrícula de valores desde una hoja de cálculo delimitado por tabuladores (por ejemplo, un archivo con extensión .txt).

map reference out*:* es una referencia al mapa que representa al entorno del robot. Se puede conectar esta salida a otro VI de planeación de rutas.

En los VIs *Occupancy Grid Map*, el origen de los mapas cuadriculados de ocupación es la celda localizada en la esquina inferior izquierda del mapa. Este comportamiento difiere de la indexación de arreglos en LabView, donde el índice del primer elemento de una matriz 2D empieza en la parte superior izquierda. Por lo tanto, cuando se conecta una matriz 2D de valores a la entrada *occupancy grid* de este VI, LabView desplaza el elemento del índice (0,0) para convertirlo en el origen del mapa cuadriculado de

ocupación (localizado en la esquina inferior izquierda) y ajusta el resto de los elementos de la matriz de entrada en relación con el nuevo origen. Por ejemplo, considere el arreglo de costos de celda (4×4) mostrado en la Figura A3.2.

Figura A3.2 Ejemplo de un arreglo de costos



Para crear un mapa cuadriculado de ocupación desde este arreglo, LabView desplaza el origen del arreglo y construye el mapa como se muestra a continuación:

```
100  100  1   1
100  100  1   1
 1    1   1   1
 1    1   1  100
```

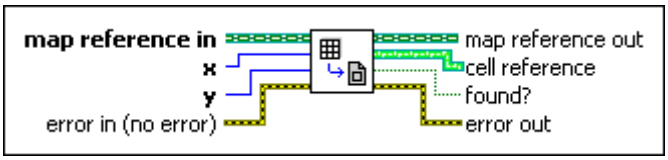
La siguiente ilustración muestra cómo los VIs *Occupancy Grid Map* definen las coordenadas de celdas x e *y* en un mapa cuadriculado de ocupación de 4 × 4. Como se observa, primero se realiza un cambio de columnas pasando la última de ellas a ser la primera columna, la penúltima columna se convierte en la segunda y así sucesivamente. El resultado final se consigue transponiendo la matriz anterior. Esta reubicación de elementos es realizada internamente por estos VIs y no es necesario que el programador se preocupe por dichos detalles.

```
(0,3) (1,3) (2,3) (3,3)
(0,2) (1,2) (2,2) (3,2)
(0,1) (1,1) (2,1) (3,1)
(0,0) (1,0) (2,0) (3,0)
```

Get Cell Reference

Retorna una referencia de la celda en el mapa cuadriculado de ocupación en la posición que se especifique. Se puede cablear la referencia de celda devuelta por este VI (Figura A2.3) a otros VIs de planeación de rutas para especificar la posición actual o la meta de un robot navegando a través de un mapa.

Figura A3.3 VI Get Cell Reference.



map reference in: es una referencia al mapa que representa el entorno del robot.

x: especifica la coordenada *x* (fila) de una celda en el mapa cuadriculado de ocupación.

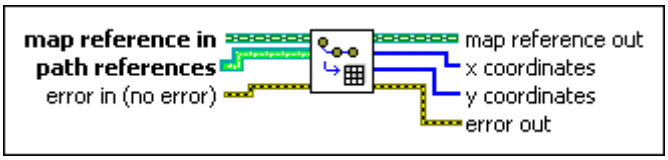
y: especifica la coordenada *y* (columna) de una celda en el mapa cuadriculado de ocupación.

cell reference: contiene una referencia a la celda del mapa en las coordenadas especificadas. Se puede conectar esta salida a otrsos VIs de planeación de rutas.

Get Cells in Path

Este VI (Figura A3.4) retorna las coordenadas de celda de los puntos a lo largo de un camino a través de un mapa cuadriculado de ocupación.

Figura A3.4 VI Get Cells in Path



map reference in: es en una referencia al mapa que representa el entorno del robot.

path references: contiene las referencias de los puntos localizados a lo largo de un camino a través del mapa.

map reference out: es una referencia al mapa que representa el entorno del robot. Se puede conectar esta salida a otros VIs de planeación de rutas.

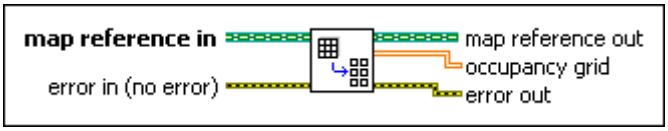
x: contiene las coordenadas *x* (filas) de las celdas localizadas a lo largo de un camino a través del mapa.

y: contiene las coordenadas *y* (columnas) de las celdas localizados a lo largo de un camino a través del mapa.

Get All Occupancy Grid Cells

Retorna una matriz (arreglo) de las celdas en un mapa cuadriculado de ocupación.

Figura A3.5 VI *Get All Occupancy Grid Cells*



map reference in: es una referencia al mapa que representa el entorno del robot

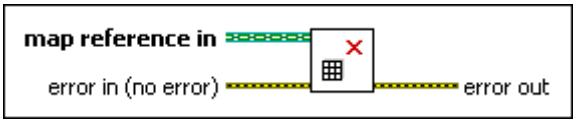
***map reference out*:** es una referencia al mapa que representa el entorno del robot

occupancy grid: contiene el costo de entrada en cada celda del mapa cuadriculado de ocupación desde una celda adyacente.

Close Occupancy Grid Map

Cierra una referencia abierta a un objeto de tipo *occupancy grid map* (mapa cuadriculado de ocupación).

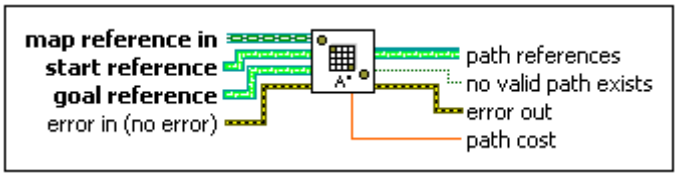
Figura A3.6 VI *Close Occupancy Grid Map*



VI A*

Este VI (Figura A3.7) planea una ruta de menor costo a través de un mapa estático, mediante un objetivo de referencia (punto de llegada) localizado dentro del mapa. Este VI usa el algoritmo A-Star para planificar una ruta óptima a través de un mapa estático y retorna un único camino óptimo.

Figura A3.7 VI A*



map reference in: es una referencia al mapa que representa el entorno del robot.

 ***start reference:*** especifica la posición de inicio del robot dentro del mapa. Si este VI opera sobre un mapa gráfico dirigido o sobre un mapa cuadriculado de ocupación, se puede utilizar el VI *Get Node Reference* o el VI *Get Cell Reference*, respectivamente, para generar esta clase de objetos en LabView.

 ***goal reference:*** especifica la posición a la cual el robot debe llegar y navegar por el mapa. Se debe tomar en cuenta las mismas consideraciones que para la entrada *start reference*.

 ***path references:*** es un arreglo de referencias a los puntos en el mapa a lo largo de la ruta que este VI calcula. Si no existe una ruta válida es *TRUE* y LabView retorna una ruta parcial. De lo contrario, esta salida contiene una ruta completa para llegar al nodo meta.

 ***path cost:*** es el costo de recorrer el *path references* (referencias de ruta). Esta salida retorna *Inf* si no existe un camino posible o si el mapa cambia de manera significativa.