FACULTAD DE INGENIERÍA Y CIENCIAS APLICADAS

IMPLEMENTACIÓN DE UN ROBOT RESUELVE LABERINTOS CON ALGORITMOS DE TOMA DE DECISIONES

AUTOR

ANDRE EDUARDO TORRES CUEVA

AÑO

2019

FACULTAD DE INGENIERÍA Y CIENCIAS APLICADAS

IMPLEMENTACIÓN DE UN ROBOT RESUELVE LABERINTOS CON ALGORITMOS DE TOMA DE DECISIONES

Trabajo de Titulación presentado en conformidad a los requisitos establecidos para optar por el título de Ingeniero Electrónico y Redes de Información.

Profesor Guía

MSc. David Fernando Pozo Espín

Autor

Andre Eduardo Torres Cueva

Año

2019

DECLARACIÓN PROFESOR GUÍA

"Declaro haber dirigido el trabajo, Implementación de un Robot Resuelve Laberintos con Algoritmos de Toma de Decisiones, a través de reuniones periódicas con el estudiante Andre Eduardo Torres Cueva, en el semestre 201910, orientando sus conocimientos y competencias para un eficiente desarrollo del tema escogido y dando cumplimiento a todas las disposiciones vigentes que regulan los Trabajos de Titulación".

_____

Pozo Espín David Fernando
Máster en Automática y Robótica
C.I.: 1717340143

DECLARACIÓN PROFESOR CORRECTOR

"Declaro haber revisado este trabajo, Implementación de un Robot Resuelve Laberintos Laberinto con Algoritmos de Toma de Decisiones, del estudiante Andre Eduardo Torres Cueva, en el semestre 201910, dando cumplimiento a todas las disposiciones vigentes que regulan los Trabajos de Titulación".

_____

Jorge Luis Rosero Beltrán

Máster en Ciencias con Especialidad en Automatización

C.I.: 1803610185

DECLARACIÓN DE AUTORÍA DEL ESTUDIANTE

“Declaro que este trabajo es original, de mi autoría, que se han citado las fuentes correspondientes y que en su ejecución se respetaron las disposiciones legales que protegen los derechos de autor vigentes.”

_____

Andre Eduardo Torres Cueva

C.I.: 1721933974

AGRADECIMIENTOS

Agradezco a cada uno de mis amigos y compañeros que estuvieron a mi lado durante este trayecto universitario. A mis profesores quienes supieron compartir sus conocimientos y experiencias para formarme como un profesional de excelencia, a mis padres quienes con paciencia y cariño supieron guiarme en mi vida como un hombre de bien, siendo mi soporte y apoyándome en cada una de mis metas.

DEDICATORIA

Este trabajo de titulación lo dedico a YHVH por darme la sabiduría y fortaleza para sobrepasar cada uno de los obstáculos que se me han presentado y ser una luz en mi camino; a mis padres por ser las bases fundamentales para que pueda lograr cumplir esta meta y ser la guía en mi vida para llegar a ser el hombre que soy, la finalización de esta etapa ha sido el resultado de su trabajo y esfuerzo, por lo cual les dedico este peldaño de mi vida con mucho amor y cariño.

RESUMEN

A lo largo de los años, se han desarrollado diferentes técnicas y algoritmos para la resolución de laberintos, estos van desde los más simples; como el algoritmo de *mano izquierda,* hasta los más complejos; como el algoritmo *Flood Fill.* Estas técnicas se han ido desarrollando y aplicando en diferentes áreas de investigación, siendo una de estas la robótica recreacional. A su vez, en dicha área, estos algoritmos son aprovechados para obtener los mejores resultados en cuanto al tiempo y distancias del recorrido.

Por tal motivo, se llevará a cabo una revisión de los algoritmos de toma de decisiones comúnmente utilizados en la robótica recreativa, con el fin de entender el principio de funcionamiento de cada uno y analizar cuáles son mejores para la implementación de un prototipo de competencias.

También, a través de la revisión de los distintos algoritmos, se aplicarán dos de éstos a dos recorridos del robot a través del laberinto, con la finalidad de optimizar el tiempo de recorrido en la última trayectoria del prototipo. A través de diagramas de flujo y pseudo código, se podrá ver el funcionamiento de los algoritmos de toma de decisiones en cada caso que el prototipo encuentre dentro de las paredes del laberinto. Para el primer recorrido del robot se aplica un algoritmo simple con el objetivo de que el robot explore el laberinto, durante la segunda pasada se utilizará un algoritmo de optimización de ruta previamente recorrida con el objetivo de mejorar el primer trayecto.

Finalmente, a través de las pruebas realizadas con el prototipo en diferentes condiciones, se demostrará la ventaja en la reducción de tiempo al utilizar los algoritmos de optimización, mediante los resultados obtenidos. Dicha ventaja, al final resulto en una disminución promedio del 50% del tiempo de recorrido, demostrando así la utilidad de esta técnica en competencia y llevándose a cabo una comparación tanto al usar o no dichos algoritmos. Por último, se podrá concluir como la toma de decisiones afectan directamente al comportamiento y al resultado final de cualquier prototipo robótico.

ABSTRACT

Over the years, different techniques and algorithms have been developed for the resolution of labyrinths, ranging from the simplest, such as the left-hand algorithm, to the most complex, such as the Flood Fill algorithm. These techniques have been developed and applied in different research areas, one of which is recreational robotics. At the same time, in this area, these algorithms are used to obtain the best results in terms of time and distances of travel.

For this reason, a review of the decision-making algorithms commonly used in recreational robotics will be carried out, to understand the principle of operation of each one and analyze which are best for the implementation of a prototype of competencies.

Also, through the revision of the different algorithms, two of these will be applied to two robot routes through the labyrinth, to optimize the travel time in the last trajectory of the prototype. Through flow diagrams and pseudo code, it will be possible to see how the decision-making algorithms work in each case that the prototype finds within the walls of the labyrinth. For the first route of the robot a simple algorithm is applied with the objective that the robot explores the labyrinth, during the second pass a route optimization algorithm will be used with the objective of improving the first route.

Finally, through the tests carried out with the prototype in different conditions, it will be demonstrated the advantage in time reduction when using the optimization algorithms, through the results obtained. This advantage, in the end resulted in an average reduction of 50% of travel time, thus demonstrating the usefulness of this technique in competition and carrying out a comparison both to use or not such algorithms. Finally, it can be concluded how decision making directly affects the behavior and the result of any robotic prototype.

ÍNDICE

1. Capítulo I. Introducción 1

1.1 Introducción 1
1.2 Objetivo General 3
1.3 Objetivos específicos 3
1.4 Alcance 4
1.5 Justificación 4

2. Capítulo II. Marco Teórico. 5

2.1 Algoritmo de Lee 5
2.1.1 Etapa de llenado 5
2.1.2 Etapa de retorno 6
2.2 Algoritmo *Flood Fill* 7
2.3 Algoritmo *Wall Follower*. 8
2.4 Algoritmo de Optimización. 9
2.5 Casos para la Toma de Decisiones 11
2.7 Controlador PID 13
2.8 Componentes electrónicos 14
2.7.1 Microcontrolador 15
2.7.3 Sensores 17

3. Capítulo III. Implementación Algoritmos de toma de decisiones 20

3.1 Desplazamiento del prototipo 20
3.2 Implementación de Algoritmo *Wall Follower* 23
3.2.1 Captura de datos 25

3.2.2 Etapa de filtrado 26
3.3 Implementación del Algoritmo de Optimización 28
3.4 Implementación del Recorrido Optimizado 30

4. Capítulo IV. Análisis y Resultados. 32

4.1 Primera Prueba 33
4.2 Segunda prueba 37

5. CONCLUSIONES Y RECOMENDACIONES 40

5.1 Conclusiones 40
5.2 Recomendaciones 41

REFERENCIAS 43

1. Capítulo I. Introducción

1.1 Introducción

Actualmente, la robótica es un tema de vanguardia y es una de las áreas de estudio más relevantes en la sociedad, pues científicos e ingenieros de varios países han ido trabajando para hacer más eficientes las actividades cotidianas gracias al uso de los robots. En el Ecuador la robótica es un campo que ha ganado terreno gracias al acceso a la tecnología. Esto se ve reflejado en los proyectos que se desarrollan en diferentes universidades e instituciones privadas y públicas del país. Los robots han sido creados con varios fines, por ejemplo, se enfocan a la salud, en actividades industriales o simplemente son desarrollados con la finalidad de participar en competencias recreativas (IBEC Corporation, 2018). Una de las categorías más desafiantes en la robótica recreativa es la categoría “Robot Laberinto”, la cual consiste en implementar un prototipo capaz de salir de un laberinto en el menor tiempo posible.

La resolución de un laberinto tiene aplicaciones directas como: la búsqueda de rutas a través de robots (Popirlan & Dupac, 2009), evacuación de emergencia (Robinette, Li, Allen, Howard, & Wagner, 2016), estudios de la activación mental (Kirsh et al., 2006), etc. (Venkata et al., 2011). Los métodos para tratar un laberinto pueden incluir algoritmos simples o complejos, esto dependerá del área en que se lo aplique. En el área de la robótica, normalmente la mayoría de los robots desarrollados para resolver un laberinto son autómatas, es decir, que son capaces de navegar por el laberinto sin intervención humana. Por lo que, un robot laberinto autómata deberá ser capaz de encontrar la salida de un laberinto por sí mismo, y en el menor tiempo posible (Hualong, Hongqi, & Yonghong, 2011).

Para que un robot autómata pueda resolver un laberinto se aplica uno de los conceptos más importantes de la robótica llamado “Toma de decisiones”; el cual consiste en la acción que realizará un robot cuando éste se encuentre en un punto de decisión (Rahman, 2017). La decisión que tome el prototipo es el

resultado del algoritmo que se esté utilizando para resolver dicha tarea, siendo así el camino recorrido por el robot, el resultado directo del algoritmo aplicado. Éste funciona de la forma en la que ve la estructura interna del laberinto; la cual puede ser como una matriz bidimensional (Mosa & Saad, 2005), como un grafo de puntos interconectados (Sedgewick & Wayne, 2018) o como un arreglo de decisiones para llegar a la meta (Al-Khawaldah, Badran, & Al-Adwan, 2011). Para que algunos de estos funcionen, el robot debe haber recorrido el laberinto al menos una vez previamente. Para ello, se puede utilizar un procedimiento para el primer recorrido y otro distinto para el segundo; a esto se le conoce como optimización de una ruta previamente conocida.

Existen otras técnicas para resolver laberintos, por ejemplo, la cooperación entre múltiples robots (Rodríguez, 2014), aplicar redes mesh para identificar el camino más corto entre los diferentes nodos en un laberinto (Rühr up & Schindelhauer, 2006) o utilizando un enfoque numérico basado en la mecánica de fluidos (Venkata et al., 2011). Estas técnicas como muchas otras se van mejorando y adaptando cada vez más a los diferentes tipos de robots autómatas utilizados con diferentes finalidades.

Existen diversos algoritmos que pueden ser aplicados a un robot para la resolución de laberintos, como el algoritmo *Wall Follower* (Seguidor de pared); consiste en seguir la pared izquierda o la pared derecha hasta encontrar la salida del laberinto sin importar los caminos falsos del recorrido. El segundo algoritmo es conocido como *Lee's Algorithm*, el cual utiliza "búsqueda en primer lugar" (Garg, 2016) para la fase de llenado; donde las celdas son marcadas, mientras que en la fase de retorno se utiliza el concepto de recursividad para encontrar el camino más corto al objetivo. (Gupta & Sehgal, 2015). También se puede usar el algoritmo *Flood Fill*, el cual consiste en aplicar a cada celda del laberinto un valor que represente la distancia a la que se encuentra la celda de la meta. (Gupta & Sehgal, 2015).

Para el propósito de la categoría "Robot Laberinto", se utiliza un prototipo de locomoción diferencial, es decir, con dos ruedas en un solo eje y como variables

de entrada se utilizan sensores de distancia, encoders y una IMU (Unidad de Medición Inercial). Esté prototipo utiliza como unidad de procesamiento un microcontrolador, el cual será capaz de tomar las variables de entrada y aplicar el algoritmo elegido para encontrar la solución al laberinto.

En el siguiente trabajo de titulación se revisarán los diferentes tipos de algoritmos para la resolución de un laberinto, así como su aplicación tanto para el primer recorrido del laberinto, como para el segundo recorrido. Para el segundo trayecto se aplicará un algoritmo de optimización de una ruta previamente recorrida; en este caso el primer recorrido. Por último, se compararán los diferentes tiempos y resultados al aplicar diferentes algoritmos (mano izquierda y mano derecha) para el primer recorrido, así como al aplicar el algoritmo de optimización para el segundo recorrido. Entre estos y otros temas serán desarrollados a lo largo de este trabajo de titulación que tiene como finalidad la implementación de un robot laberintos con algoritmos de toma de decisiones.

1.2 Objetivo General

Implementar un robot capaz de resolver laberintos utilizando algoritmos de toma de decisiones.

1.3 Objetivos específicos

- Revisar los diferentes tipos de algoritmos para la resolución de un laberinto.
- Aplicar los algoritmos de mano derecha o mano izquierda para el primer recorrido del robot.
- Aplicar un algoritmo de optimización para la resolución del laberinto en base a un camino recorrido previamente.
- Comparar los resultados obtenidos aplicando los diferentes algoritmos en los recorridos del laberinto.

1.4 Alcance

Este trabajo de titulación está centrado en la implementación y aplicación de varios algoritmos para la resolución de un laberinto. Los algoritmos que se van a revisar están basados en el concepto de “toma de decisiones”, a la cual están sometidos los robots del tipo autónomos. Por lo que se utilizará un prototipo de robot laberinto para aplicar los algoritmos que ayudarán al robot a memorizar el camino más optimo del laberinto y posteriormente recorrerlo.

Para la navegación del robot dentro del laberinto, se utilizarán algoritmos que le permitan al robot navegar por el centro de las paredes del laberinto, así como algoritmos que le permitan girar correctamente. También utilizaremos algoritmos sencillos para que el robot tome datos del laberinto la primera vez que lo recorra, y guarde estos datos en su memoria.

Se implementará un algoritmo que, basado en el camino recorrido anteriormente, permita optimizar la ruta guardada en su memoria, esto con la finalidad de que el robot no entre en caminos falsos dentro del laberinto y logre realizar un menor tiempo en el segundo recorrido.

1.5 Justificación

En el Ecuador, las competencias de robots laberintos no se encuentran tan desarrolladas como en otros países. Este proyecto de titulación motiva e ínsita a que las competencias de laberintos crezcan en cuanto a competitividad y desarrollo de nuevas técnicas o algoritmos para resolver un laberinto.

El primer prototipo realizado por estudiantes de la UDLA ha ganado varios reconocimientos y premios a lo largo de su trayectoria en competencias. Este proyecto sirve como muestra de lo que los estudiantes de la Universidad de Las Américas pueden hacer, esto motiva a los estudiantes a participar en diferentes tipos de concursos de robótica o innovación. Además, existen diferentes tipos de categorías de robótica que utilizan los mismo o similares algoritmos, por lo que este proyecto será un gran avance para el desarrollo de prototipos dentro de la

Universidad. Por lo que el prototipo quedará como un ejemplo o guía para futuros proyectos de nuevos estudiantes.

Los algoritmos que se desarrollan para el prototipo pueden ser aplicados a diferentes campos de la robótica o proyectos con autonomía. Por ejemplo, se los puede aplicar en robots de exploración o rescate con el fin de que estos robots sean autónomos en la toma de decisiones, se los puede aplicar en el muestro de lugares o habitaciones con la finalidad de tener una orientación exacta de donde se encuentra el robot. Otro ejemplo, estos algoritmos se los puede desarrollar para ser aplicados en robots industriales, que van de lado a lado en las fábricas y tienen que evadir personas u objetos.

2. Capítulo II. Marco Teórico.

En este capítulo se revisará el funcionamiento de los algoritmos que se utilizarán para los recorridos del robot a través del laberinto, así como otros algoritmos para la resolución de laberintos que se pueden utilizar. Además, se hablará del controlador PID y como este ayuda a controlar sistemas o procesos a través de su fórmula matemática. También se revisará los casos formados por las paredes interiores del laberinto, los cuales servirán para la toma de decisiones. Por último, se verá los componentes electrónicos para la implementación de un robot laberinto.

2.1 Algoritmo de Lee

El algoritmo de Lee funciona cuando el robot ya conoce el laberinto, y lo ve como una red fija de puntos. Para encontrar la solución el algoritmo avanza en dos etapas:

2.1.1 Etapa de llenado

En la etapa de llenado, el algoritmo empieza a dar valores a las celdas continuas del punto de inicio (*S*). Las celdas vecinas al punto de inicio (*S*) serán marcadas con “1”, mientras que las siguientes celdas vecinas serán marcadas con “2”;

estos valores representan la distancia de cada celda hasta el punto de inicio. Este proceso se repite hasta que se logre alcanzar el punto final o la meta (*D*).

2.1.2 Etapa de retorno

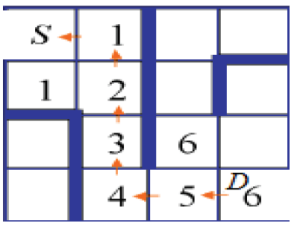

Figura 1. Algoritmo de Lee

Tomado de *(Gupta & Sehgal, 2015)*

La fase de retorno consiste en regresar al punto de inicio, por lo que si se está en el punto final (D) en el paso “N” se avanza por la ruta dirigiéndose a la celda “N - 1”. Este proceso se repite hasta que se logre llegar al punto inicial (S). Dicho procedimiento se lo puede apreciar en la Figura 1.

Una de las ventajas de este algoritmo, es que se trata de un algoritmo BFS (búsqueda en primer lugar); una aplicación del algoritmo de Dijkstra (Mosa & Saad, 2005). Este algoritmo proporciona la ruta más corta entre dos puntos, por lo que su mayor ventaja es la utilización de la ruta ideal a diferencia de otros algoritmos que no devuelve la ruta más optima dependiendo de la estructura del laberinto que se esté utilizando.

Entre las desventajas de esté algoritmo es que requiere una gran capacidad de almacenamiento para estructuras complejas, además que garantiza encontrar la solución óptima, pero en una gran cantidad de tiempo; debido a su alto nivel de procesamiento. La mayor desventaja de esté algoritmo es que espera que la estructura del laberinto sea conocida para empezar con su estrategia de búsqueda, por lo que para una competencia robótica no sería lo más óptimo (Gupta & Sehgal, 2015).

2.2 Algoritmo *Flood Fill*

El Algoritmo *Flood Fill* es una alternativa al algoritmo de *Lee,* es uno de los algoritmos más adoptados por los prototipos conocidos como "*MicroMouser*". Esté algoritmo espera que al principio del recorrido no haya divisores o paredes en el laberinto y luego hace use del algoritmo de *Lee* para dirigirse hacia el destino. A medida que va avanzando por el laberinto, recorre a través de nuevos divisores, estos divisores se van agregando a la memoria del robot y se va aplicando el algoritmo de *Lee* para moverse hacia la siguiente celda (Gupta & Sehgal, 2015).

Una vez que el robot haya alcanzado el punto final, el algoritmo empezará a dar valores a cada celda dependiendo la distancia a la que se encuentren de la meta. La meta será marcada con un valor de "0" y sus celdas vecinas (celdas a las que se puedo mover) serán marcadas con un "1" y las vecinas de estas celdas con un "2" y así sucesivamente hasta llegar al punto de inicio, esto se lo puede observar en la Figura 2.

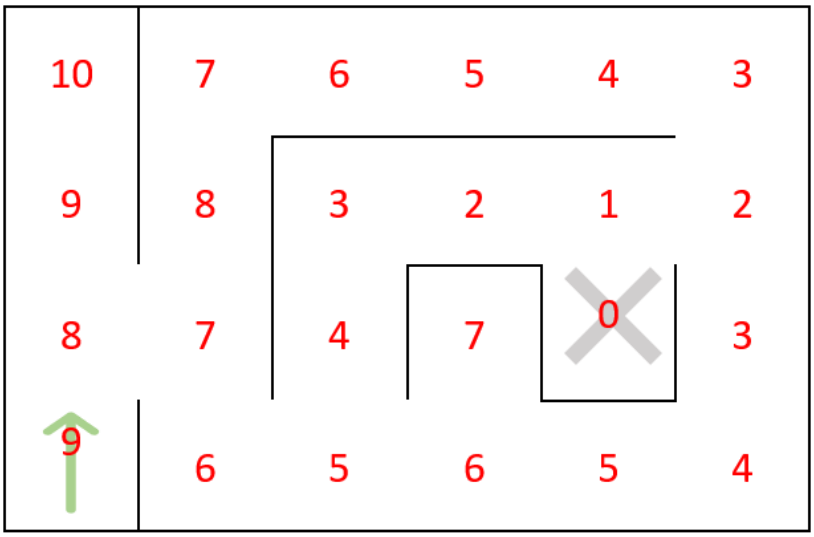

Figura 2. Algoritmo Flood Fill.

Una vez que todas las celdas hayan sido marcadas, el siguiente paso para el robot es moverse desde el punto de inicio hacia la siguiente celda con un valor menor a la actual, y así sucesivamente hasta encontrar el punto final del

laberinto. En el caso de que el corredor se encuentre entre dos celdas del mismo valor el algoritmo siempre preferirá avanzar recto ya que es más rápido, en el caso de que las dos celdas sean a los lados, dependerá de la preferencia que el programador haya dado al algoritmo.

Una de las ventajas de esté algoritmo es que no asume que el laberinto es conocido, al contrario, asume que el laberinto se encuentra vacío y luego aplica el algoritmo de *Lee* para encontrar la solución. Así mismo, es un algoritmo fácil de entender, además de que es eficiente y garantiza encontrar el camino más corto a través del laberinto. Por el contrario, el algoritmo es altamente recursivo lo que puede ocasionar un desbordamiento durante su ejecución; ocasionado por innumerables llamadas así mismo. Y, además requiere una gran cantidad de memoria para laberintos más grandes, esto se traduce en que la pila en la que se guarden las celdas requerirá ser mayor (Gupta & Sehgal, 2015).

2.3 Algoritmo Wall Follower.

El algoritmo *Wall Follower* es también conocido como “Mano Derecha” o “Mano Izquierda”, dependiendo de la pared que se vaya a seguir. Este algoritmo es la técnica más antigua, conocida y sencilla para resolver un laberinto. Consiste en seguir la pared izquierda o derecha del laberinto hasta encontrar la salida, sin importar los caminos cerrados que se vayan encontrando en el trayecto. Con esté algoritmo se logra que el robot encuentre la salida al laberinto; caso contrario, regresará al inicio del laberinto después de haber cruzado cada pasaje del laberinto al menos una vez. En el caso de que el laberinto no tenga las paredes conectadas, esta técnica no garantiza que se logre el objetivo (Gupta & Sehgal, 2015). Un ejemplo de dicho procedimiento se encuentra en a Figura 3.

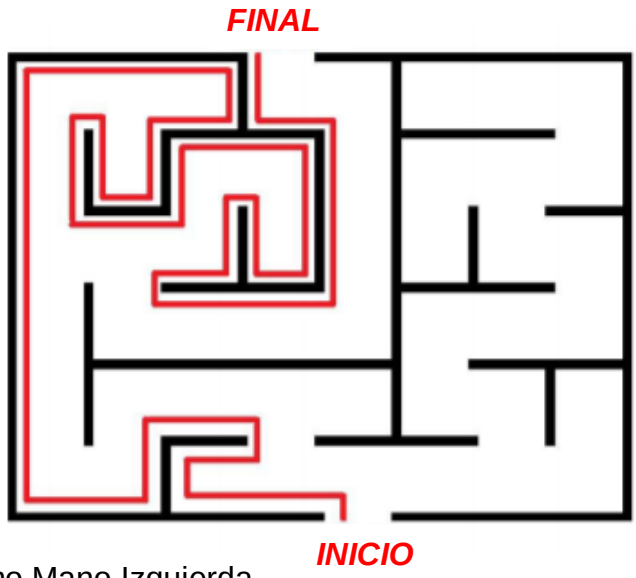

Figura 3. Algoritmo Mano Izquierda.
Tomado de *(Bienias, Szczepański, & Duch, 2016)*

La principal ventaja de esté algoritmo es su bajo nivel de complejidad, por lo que se lo puede llegar a cumplir sin gran esfuerzo. Está técnica promete dar solución para los laberintos simplemente conectados; es decir un laberinto sin paredes que formen enlaces o bucles repetitivos.

También existen ciertas desventajas al hacer uso de este algoritmo, la primera es que, si el punto de partida o final se encuentran en el centro de la estructura, los caminos se cruzan o si el laberinto no está esencialmente asociado, este algoritmo no garantiza la resolución del laberinto. La segunda desventaja es que toma demasiado tiempo y esfuerzo llegar a la meta, lo que implica: que el algoritmo ocupe más memoria en el caso de que se esté guardando el trayecto, que el consumo de energía por parte del robot sea mayor y que el robot tome más tiempo en llegar a la meta.

2.4 Algoritmo de Optimización.

Este algoritmo es utilizado comúnmente en laberintos de líneas sin bucles repetitivos, pero se lo puede adaptar para usarlo en laberintos con estructuras de paredes físicas. Para que el algoritmo de optimización funcione, el robot necesita haber cruzado el laberinto al menos una vez previamente. En la primera pasada, el robot a travesará varios caminos sin salidas, pero los guardará en

memoria como caminos “malos” para que puedan ser evitados durante el segundo recorrido. Para este algoritmo se tiene que representar cada punto de decisión como un caso; el cual puede ser “R”, “L”, “S” o “B”, la “R” hace referencia a un giro de 90 grados hacia la derecha, la “L” hace referencia a un giro de 90 grados hacia la izquierda, la “S” hace referencia a avanzar recto y la “B” hace referencia a un camino sin salida o un giro de 180 grados (Vannoy, 2009).

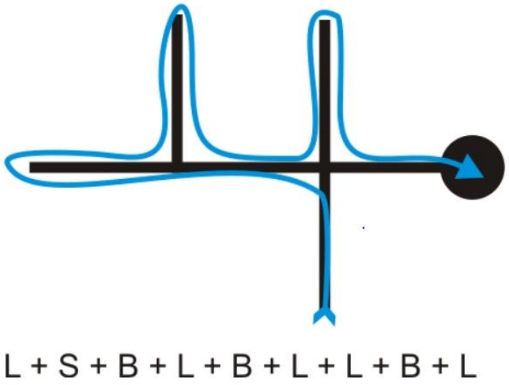

Figura 4. Caso utilizando Mano Izquierda.
Tomado de *(Pololu Corporation, 2018)*

Como se puede apreciar en la Figura 4, ya tenemos guardado el recorrido del laberinto en una cadena de caracteres. Lo primero que se tiene que hacer es buscar el carácter “B” con sus respectivos caracteres a cada lado, en este caso primero se encuentra “SBL”. Para evitarnos este camino falso se puede remplazar la cadena “SBL” con un solo giro a la derecha “R”, como se ve en la Figura 5.

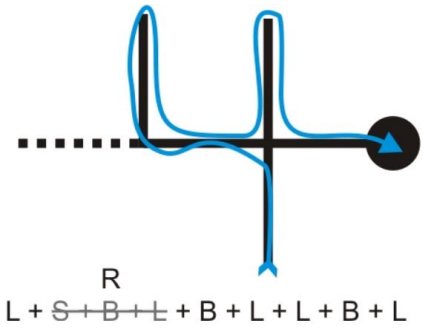

Figura 5. Eliminación “SBL”.

Tomado de *(Pololu Corporation, 2018)*

Seguido nos encontraríamos con la secuencia “RBL”, la cual se reduce a un simple “B”, la cual combinada con la siguiente acción produce la secuencia “LBL”, la cual se reduce a avanzar recto “S”, como se puede apreciar en la Figura 6.

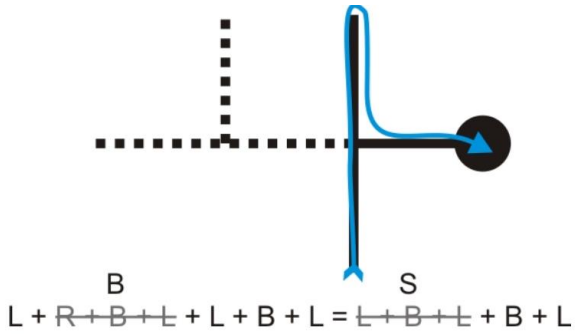

Figura 6. Eliminación de “RBL” y “LBL”.
Tomado de *(Pololu Corporation, 2018)*

La última interacción da como resultado “SBL”, que se puede reducir con un giro a la derecha “R”. Por lo que nuestra cadena de casos nos queda con solo “R” y está representa al camino más corto entre el inicio y final del laberinto, como se muestra en la Figura 7.

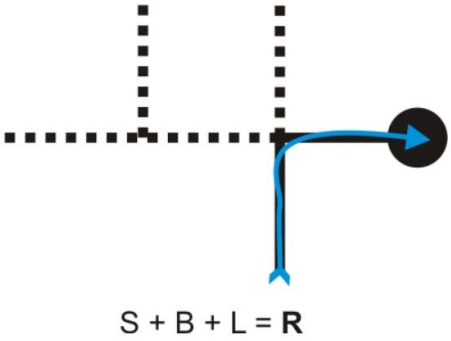

Figura 7. Camino más corto.
Tomado de *(Pololu Corporation, 2018)*

2.5 Casos para la Toma de Decisiones

Durante el recorrido de un robot dentro del laberinto, esté se va encontrando con diferentes casos los cuales están formados por las paredes del laberinto. En el caso de un laberinto simple de una entrada y una salida, el robot se puede

encontrar con ocho casos (Morales, Pozo, Rosero, Sandobalin, & Rodríguez, 2014), los cuales se muestran en la Figura 8.

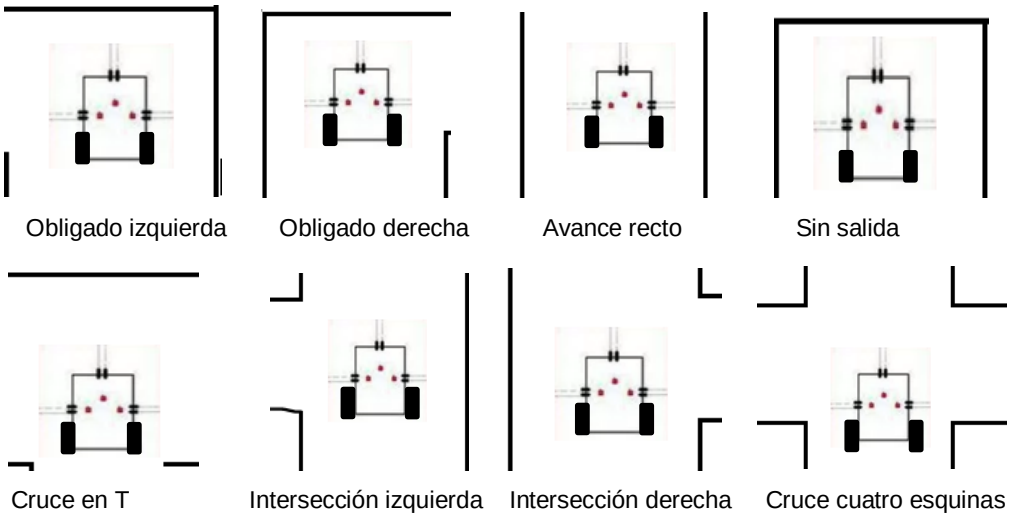

Figura 8. Casos dentro del laberinto.

Cada caso representa para el robot una toma de decisión, está decisión es el resultado del algoritmo que se esté aplicando. Por ejemplo, si estamos usando el algoritmo de “mano izquierda” tendremos que apegarnos a la norma de preferir siempre girar a la izquierda o mantenernos en línea recta en el caso de que no se pueda girar a la izquierda. Por el contrario, si se utiliza el algoritmo “mano derecha”, se tiene que mantener la regla de sobre todo preferir girar a la derecha o seguir recto en el caso de que no se pueda girar a la derecha. En el caso “sin salida”, sin duda en cualquier algoritmo se realiza un giro de 180 grados (Morales et al., 2014).

Estos casos también se los puede guardar en una colección de datos, para posteriormente optimizar la ruta hacia la salida del laberinto. En el caso de un prototipo robótico, este tiene que ser capaz de identificar de manera correcta cada uno de los casos, para que el algoritmo que se aplique pueda funcionar correctamente. En el siguiente capítulo se observará como el prototipo de este

trabajo de titulación mira los casos del laberinto según el algoritmo que le estemos aplicando.

2.6 Controlador PID

Un controlador PID (Proporcional, Integral y Derivativo) es una técnica o mecanismo de control por realimentación, éste calcula la variación o error entre un valor censado y el valor deseado. Es decir, mide cuanto el sistema a controlar se está desviando de su estado deseado, para así poder corregirlo y mantenerlo en el estado adecuado. Este algoritmo se alimenta de la o las entradas del sistema; típicamente uno o varios sensores, después el algoritmo aplica las diferentes fórmulas de cada parámetro del PID para realizar su cálculo, por último el resultado se envía a los accionadores del sistema para aplicar el control deseado (Ogata, 1998). En la Figura 9, se aprecia el esquema clásico de PID para un sistema o planta el cual usa realimentación para realizar el respectivo control.

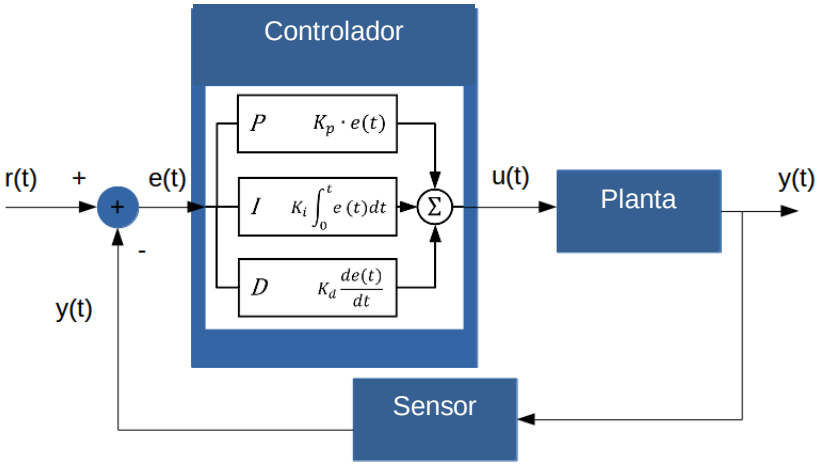

Figura 9. Esquema controlador PID.
Tomado de *(PUCP, 2018)*

El control PID consiste en tres parámetros como su nombre lo indica: Proporcional, Integral y Derivativo. El Proporcional depende del error actual que exista en la entrada. El Integral va a depender de los errores pasados y el Derivativo ayuda a tener una predicción de los errores futuros. Estos tres parámetros en combinación se usan para ajustar el proceso o el sistema de

posibles desviaciones u obstáculos, manteniendo así en control la planta o proceso al que se lo esté aplicando. El controlador PID es un tipo de control simple, ya que no requiere de un modelo matemático de la planta para ser implementado. A pesar de lo dicho anterior, este control es bastante amplio y existen diferentes variaciones de éste que se pueden implementar.

Como se puede apreciar en la Figura 9, el controlador obtiene el error e(t) de la diferencia entre el valor deseado r(t) y el valor censado y(t), luego el controlador aplica la fórmula con cada una de sus constantes Kp, Ki y Kd. Estas constantes se las puede modificar para obtener una salida u(t) deseada, la cual es resultado de la suma algebraica de los componentes proporcional, integral y derivativo. A continuación, se muestra la fórmula general de un controlador PID.

$$u\,(t) = Kp.e(t) + Ki.\sum_{i=0}^{n} e(ki) + Kd\,(k_t - k_{t-1})$$

Ecuación 1. Fórmula PID.

2.7 Componentes electrónicos

En esta sección, se revisará los componentes electrónicos principales que se requieren para la implementación de un robot laberinto, como lo son: un microcontrolador, sensores de distancia, IMU (unidad de medición inercial), encoders y *Driver*. Estos elementos no son los únicos que se requieren; así como no todos son indispensables, pero serán los que se necesitan para aplicar los algoritmos que se plantean en este trabajo de titulación. En la Figura 10, se encuentra el diagrama de conexión de los diferentes componentes electrónicos.

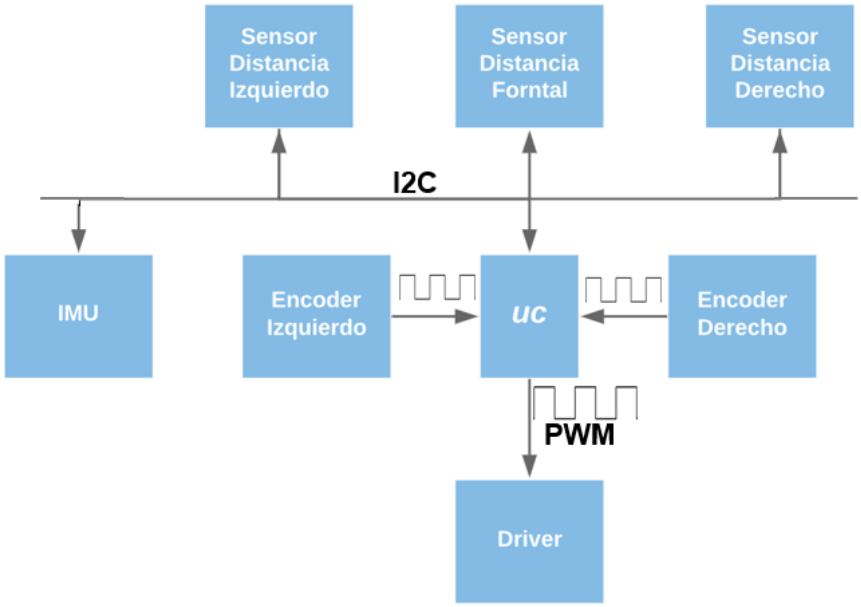

Figura 10. Diagrama de conexión.

2.7.1 Microcontrolador

Un microcontrolador es una unidad de procesamiento que trabaja como una computadora económica de un solo chip. Principalmente, un microcontrolador es capaz de almacenar y correr un programa, lo cual es su característica más importante. Está compuesto por una CPU (*unidad central de procesamiento*), RAM (*memoria de acceso aleatorio*), ROM (*memoria solo de lectura*), líneas I/O (entradas/salidas), puertos en serie y paralelos, temporizadores y en algunas ocasiones otros periféricos incluidos como conversores A/D (análogo a digital) y D/A (digital a analógico) (Iovine, 2000).

Tabla 1.
Características Atmega328p.

Nombre	Valor
Líneas I/O	23
Periféricos de comunicaciones digital	1-UART, 2-SPI, 1-I2C
PWM	6 PWM
Temporizadores	2 x 8-bits, 1 x 16-bits

Frecuencia máxima de operación	20 MHz
Voltaje de operación	1.8 hasta 5.5
Canales ADC	8

Tomado de (Microchip Technology Inc., 2018).

Con el objetivo de aplicar los algoritmos de toma de decisiones, el prototipo que se utilizará para esté trabajo de titulación contará con el microcontrolador Atmega 328P; se lo puede observar en la Figura 10 como bloque "*uc"*. Consiste en un microcontrolador de alto rendimiento que se basa en una arquitectura AVR de 8 bits, junto con 32 KB de memoria flash, 1 KB de memoria EEPROM y 2 KB de SRAM (Microchip Technology Inc., 2018), entre otras especificaciones mostradas en la Tabla 1.

2.7.2 Driver

Driver o Puente H, es un circuito electrónico que permite controlar el sentido de giro de un motor. Funciona utilizando transistores para controlar el flujo de corriente que va hacia el motor, haciendo uso de cuatro transistores NPN o PNP. El Driver sirve como interfaz entre el microcontrolador y el motor, permitiendo entregar más corriente al motor y a su vez controlar la velocidad mediante pulsos PWM (Caicedo & Trijillo, 2016).

Para el prototipo se utilizó el drive TB6612FNG, el cual incorpora dos puentes H, para manejar dos motores, y permite un rango de voltaje para los motores de 4.5 a 13,5 V. Este driver incorpora un pin de frenado rápido, además que permite manejar las velocidades de los motores independientemente. Para ver más detalles del TB6612FNG referirse a la Tabla 2.

Tabla 2.
Características TB6612FNG.

Nombre	**Valor**
Voltaje de motor	4.5 a 13.5 V

Voltaje lógico	2.7 a 5.5 V
Salida de corriente máx. por canal	3.2 A
Salida de corriente continua	1.2 A
Máxima frecuencia de PWM	100kHz

Tomado de (Pololu Corporation, 2019).

2.7.3 Sensores

Un sensor se puede definir como un dispositivo que mide un atributo del mundo, por lo cual se los puede clasificar en dos tipos de acuerdo con el tipo de información medida que nos otorgue a través de su salida. Existen los sensores propioceptivos; los cuales miden un atributo con respecto a su propio estado, y los sensores exteroceptivos; los cuales miden un atributo de un objeto externo presente en la escena (Discant & Rogozan, 2007). Para la implementación de este trabajo de titulación, se usará los dos tipos de sensores: Un sensor exteroceptivo para medir las distancias de las paredes del laberinto con respecto al prototipo, y sensores propioceptivos tanto para medir, cambiar o mantener la orientación en el espacio del prototipo, como para medir las revoluciones de éste al recorrer el laberinto. Estos sensores se encuentran detallados a continuación.

2.7.3.1 Sensores de distancia

En este caso se utilizarán sensores de distancia infrarrojos, los cuales se pueden dividir en activos y pasivos. Los activos son los que generan una luz de longitud de onda infrarroja para detectar la reflexión de posibles objetivos, mientras que los pasivos son los que responde al cambio de luz infrarroja reflejada por el objetivo (Discant & Rogozan, 2007). Para la implementación de un robot laberinto se utilizará un sensor infrarrojo activo, el cual funciona al emitir un haz de luz infrarrojo por su diodo emisor y recibir el reflejo del haz por su diodo receptor. Este calcula la distancia al objetivo utilizando la velocidad de la luz y el tiempo de vuelo del haz de luz en este caso.

Tabla 3.
Características V6180X.

Nombre	Valor
Resolución	1 mm
Rango máximo	60 cm
Interfaz	I2C
Voltaje mínimo de operación	2.7 V
Voltaje máximo de operación	5.5 V
Voltaje de operación	1.8 hasta 5.5
Corriente de suministro	5 mA

Tomado de *(Pololu Corporation, 2018).*

El sensor que se usará es el VL6180X; se los puede observar en la Figura 10 como *Sensores de Distancia*, el cuál mide la distancia al objetivo hasta un máximo de 20 cm, o 60 cm con resolución reducida. El VL6180X utiliza mediciones de tiempo de vuelo de los pulsos infrarrojos para medir el alcance, lo que le permite brindar resultados precisos independientemente del color y la superficie del objetivo. Los resultados del sensor se pueden leer a través de su interfaz digital I2C (Pololu Corporation, 2018). Se utilizarán tres sensores VL6180X en el prototipo, uno a cada lado del robot y uno en la parte frontal, estos servirán para identificar los casos del laberinto.

2.7.3.2 IMU

“Una unidad de medición inercial o IMU es un dispositivo electrónico cuyo objetivo es obtener mediciones de velocidad, rotación y fuerzas gravitacionales, usando una combinación de acelerómetros, giróscopos y a veces magnetómetros.” (RobdosTeam, 2016). Una IMU es capaz de detectar los cambios de orientación utilizando uno o más giróscopos, para lo cual se mide las variaciones angulares en un eje determinado. Estos sensores entregan la velocidad angular de cada uno de los tres ejes (x, y, z), mediante este dato entregado por el sensor y el tiempo que se establezca para el cálculo, se puede obtener los grados recorridos en un eje con respecto a un determinado tiempo.

Tabla 4.

Características MPU-6050.

Nombre	Valor
Resolución ADC	16-bits
Corriente de operación	3.5 mA
Interfaz	I2C
Rango de escalas	±250, ±500, ±1000 y ±2000 º/sec
Voltaje de operación	2.375 V – 3.46 V

Tomado de (Ave, Number, & Date, 2013).

MPU-6050 es la primera IMU de seguimiento de movimiento integrada de seis ejes, la cual combina un giroscopio de tres ejes, un acelerómetro de 3 ejes y procesador de movimiento digital (Ave et al., 2013). Esta IMU entrega sus resultados a través de la interfaz I2C, los cuales pueden ser medidos fácilmente por un microcontrolador con distintas finalidades. En el caso de nuestro prototipo servirá para comprobar que el robot gire el número de grados establecido para cada caso. Se puede observar el IMU MPU-6050 en la Figura10 como bloque *IMU*.

2.7.3.3 Encoders rotatorios

Un encoder rotario es un dispositivo electrónico que permite detectar la posición y velocidad de los sistemas de control de motores, son utilizados a gran escala en la automatización por su fácil aplicación en circuitos digitales. Existen dos tipos de encoders rotatorios: ópticos y magnéticos (Takahashi & Yamanaka, 1987). Todos tienen el mismo principio de funcionamiento, el cual consiste en contar las revoluciones de giro del eje del motor dependiendo si su sensor activa o no la señal al interactuar con el tambor del encoder, el cual se encuentra dividido en múltiples secciones, de las cuales se establece el número de revoluciones. Los encoders magnéticos son más compactos y tienen una velocidad de respuesta mucho más rápida que los encoders ópticos (Takahashi & Yamanaka, 1987).

Para esté trabajo de titulación se utilizará los encoders magnéticos de Pololu, los cuales emiten su salida en cuadratura; es decir nos permite ver el sentido de giro de los motores. Los micromotores de engranaje metálico de la misma marca se pueden adaptar con este kit de encoders, los cuales utilizan un disco magnético y un sensor de efecto hall para brindar 12 pulsos por revolución del motor. Se puede observar esté kit en la Figura 10 como *Encoders*.

Tabla 5.
Características Magnetic Encoder Kit.

Nombre	Valor
Voltaje de operación mínima	2.7 V
Voltaje de operación máximo	18 V
Salida	Digital
Cuadratura	Si

Tomado de *(Pololu Corporation, 2018b).*

3. Capítulo III. Implementación Algoritmos de toma de decisiones.

En este capítulo, se explicará la implementación de los algoritmos de toma de decisiones para resolución de un laberinto. Se empezará por la navegación básica del prototipo dentro de éste, hasta llegar a la optimización de una ruta previamente recorrida con la finalidad de mejorar el tiempo del recorrido previo.

3.1 Desplazamiento del prototipo

Para la navegación del robot dentro del laberinto, éste hace uso de sensores de distancia, los cuales le permiten tener una percepción de entorno en el que se encuentra. Los sensores de distancia infrarrojos VL6180X tienen un alcance de 200mm, lo que permite tener un rango amplio de medición debido a que por reglamento el ancho de los pasillos interiores del laberinto es de 250mm. En nuestro caso, se utilizarán tres sensores de distancia, uno a cada lado del prototipo y uno apuntando hacia al frente; cómo se puede apreciar en la Figura 11.

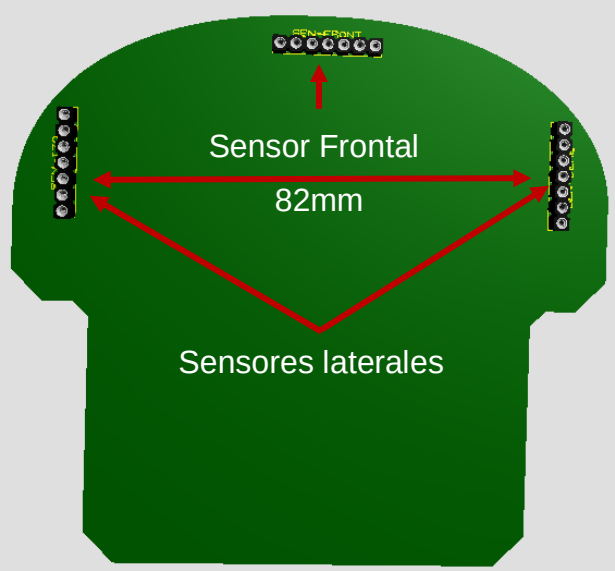

Figura 11. Ubicación de los sensores de distancia.

La separación de los sensores laterales es de 82mm, los cuales se deben restar de los 250mm del ancho de los pasillos, dando como resultado 168mm. Éste dato significa el espacio vacío que existe entre las paredes del laberinto con el robot, por lo tanto, dividiendo el resultado para dos se logra ubicar al prototipo exactamente entre las dos paredes del pasillo. El resultado de dicho cálculo es 84mm, valor que representa la distancia deseada que deberán medir los sensores laterales para que el robot se mantenga en el centro de los pasillos mientras recorre el laberinto.

Con la finalidad de que el robot no se desvié y se mantenga en línea recta por los pasillos del laberinto, se implementa un control PID capaz de controlar la velocidad de los motores mientras en función de la lectura de los sensores. Para dicho controlador PID se establece el *set point* con el valor calculado previamente, es decir 84mm; lo que permitirá que el controlador PID mantenga el robot a esa distancia de la pared. Igualmente, se tiene que establecer una velocidad máxima y una mínima para que el controlador PID varié la velocidad de los motores entre dichos valores. Si se aplica la fórmula general para un controlador PID se obtendrá el pseudocódigo de la Figura 12, para implementar dicho controlador.

```
PID() {
    p = set_point - sensor
    i += ∑p;
    d = p - last_p;
    last_p = p;

    dif = (p * Kp) + (i * Ki) + (d * Kd);

    if (dif > vel_max)
        dif = vel_max;
    if (dif < -vel_max)
        dif = -vel_max;

    if (dif > 0) {
        motor_left = vel + dif;
        motor_right = vel;
    }
    else {
        motor_left = vel;
        motor_right = vel + dif;
     }
 }
```

Figura 12. Pseudocódigo controlador PID.

Las variables *Kp*, *Kd* y *Ki*, se obtienen heurísticamente en base a experimentación. La forma más fácil de obtener dichas variables es empezar a probar con valores no muy altos, e ir variando su magnitud dependiendo como favorezcan al sistema. Como se puede ver en la Figura 12, si el resultado del algoritmo PID (diferencia) sobrepasa la velocidad máxima, a éste se le da el valor de la velocidad máxima, ya que la acción de corrección no puede ser demasiado grande. Dependiendo si la “diferencia” es negativa se aplicará al motor derecho, mientras que si es positiva se aplicará al motor izquierdo; en el caso de que se esté usando el sensor lateral contrario la diferencia será aplicada en los motores de forma inversa.

El algoritmo de la Figura 12, será utilizado en los casos de línea recta, mientras el sensor de la parte delantera no encuentre ningún obstáculo. Por otro lado, los giros que realiza el robot se encuentran relacionas al número de grados que recorren desde que inicia el giro hasta cumplir 90º. La medición de grados se calcula con la entrega de datos de la IMU ubicada en el centro del prototipo, de esta forma se conseguirá que los giros sean lo más exactos posibles evitando choques y giros mal realizados. Los giros se pueden realizar aplicando velocidad en un sentido para un motor y en el otro sentido para el otro motor, o

disminuyendo la velocidad de un motor proporcionalmente para que el robot realice una curva recorriendo un arco formado por la esquina de la curva hasta el robot.

3.2 Implementación de Algoritmo *Wall Follower*

Para el primer recorrido del robot dentro del laberinto se utilizará el algoritmo *Wall Follower,* mientras el prototipo vaya aplicando este algoritmo, irá guardando en un arreglo los casos que va recorriendo hacia la salida. Además de guardar los casos, éste también guardará las distancias recorridas entre cada caso; con la finalidad de luego aplicar un filtrado de casos para así tener un mapeo del laberinto más confiable.

Lo primero que se tiene en cuenta para aplicar el algoritmo de “Mano Izquierda”, es identificar cada caso posible para la configuración de los sensores que disponemos. En este algoritmo se han encontrado seis posibles casos, los cuales siguiendo la lógica de “Mano Izquierda” obligan al robot a girar a la izquierda, seguir recto o dar media vuelta. En el software del prototipo, se ha dado un valor numérico a cada caso con la finalidad de guardar este valor en un arreglo de enteros. Los casos con sus respectivos valores se encuentran en la Figura 13.

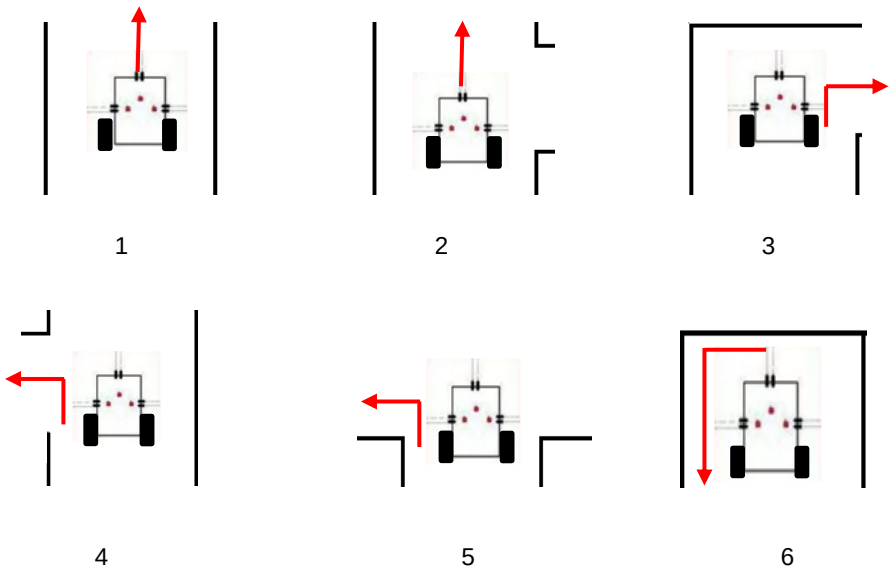

Figura 13. Casos con valores mano izquierda.

Para guardar las distancias recorridas por el prototipo, hacemos uso de los encoders magnéticos. Las salidas de estos encoders se encuentran conectadas a pines digitales que son capaces de generar interrupciones de cambio de estado. Es decir, cada vez que el encoder emita un pulso positivo se generará la interrupción en el microcontrolador donde se utilizará un contador para calcular la distancia recorrida. Este cálculo se lo realiza dividiendo perímetro de las ruedas para el número de pulsos por el revolución del encoder, por el fabricante sabemos que el número de pulsos por revolución del encoder es 12 (Pololu Corporation, 2018b). Cada vez que pasemos de un caso a otro, el contador se reiniciará desde cero para que el siguiente caso guarde las distancia adecuadamente.

Con la finalidad de aplicar el algoritmo de *mano izquierda* se estableció un lazo de programa con varios condicionales, los cuales evalúan las distancias medidas por los sensores. Para esto, se tienen que definir las distancias en las que existen paredes o no, para el análisis. Debido a que los sensores VL6180X pueden medir hasta 200mm, ésta será la distancia en la que se considera que existe una pared para los sensores laterales, mientras que para el sensor delantero se considerará que existe una pared a la distancia de 80mm; debido a que el robot tiene que avanzar una distancia considerable antes de girar para evitar choques después del giro. Está lógica se la puede apreciar en la Figura 14.

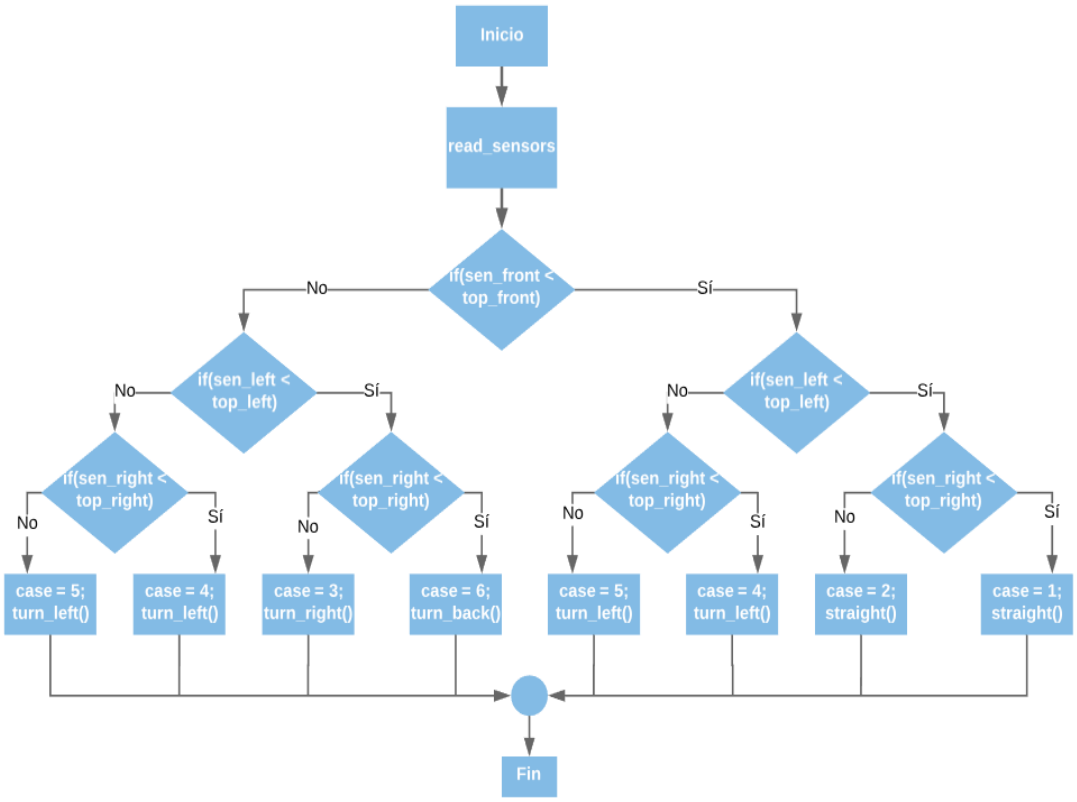

Figura 14. Lógica “Mano Izquierda”.

3.2.1 Captura de datos

Como se puede ver en el diagrama anterior, en cada caso que se entra se coloca en la variable “estado” el número correspondiente de cada caso para luego ser envía a la función “llenarCamino”. Esta función sirve para guardar los casos y las distancias entre casos en dos arreglos globales; llamados “laberinto” y “pasos” respectivamente, los cuáles finalizado el recorrido del laberinto serán guardados en la memoria EEPROM del prototipo. Para lograr esto, se debe tener una variable que lleve el índice o el número de casos que se vayan guardando, para lo cual se ha definido una variable global llamada “posicion”. Además, tenemos que filtrar los casos que se repiten debido a que su solución es seguir en línea recta aplicando el control PID, estos casos son el “1” y el “2”. Por último, se debe guardar el estado nuevo en un estado pasado, para aplicar el filtrado mencionado anteriormente. En la Figura 15 se encuentra el pseudocódigo de esta función.

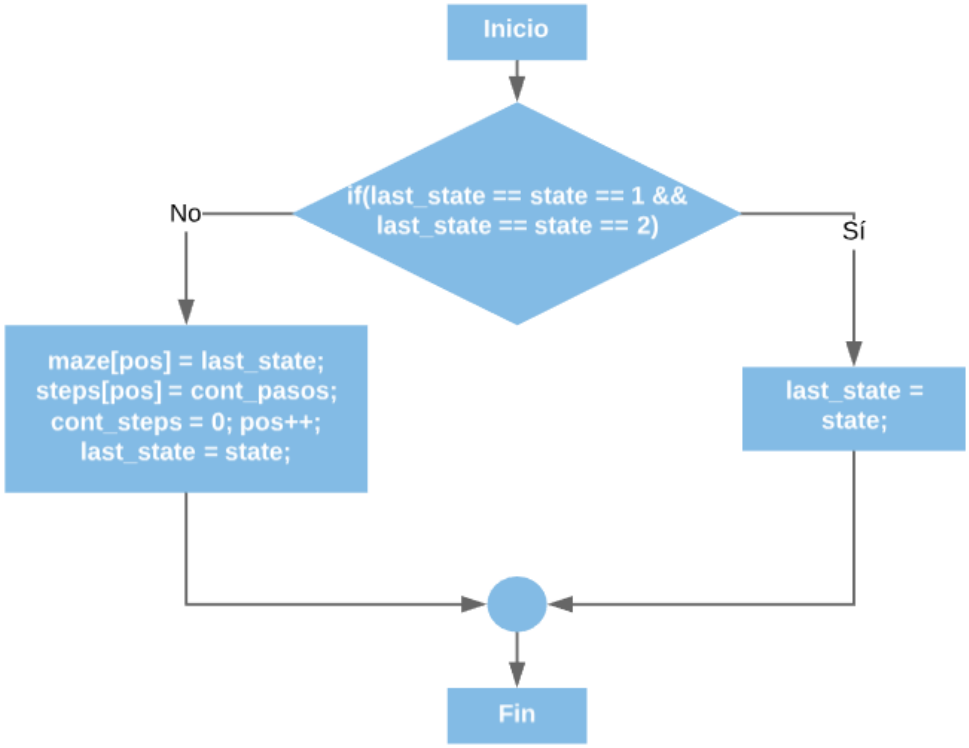

Figura 15. Función “llenarCamino”.

3.2.2 Etapa de filtrado

Al final del recorrido del prototipo a través del laberinto, éste guarda todo el recorrido en el arreglo llamado *“laberinto”;* al cual se lo tiene que pasar por una etapa del filtrado para que pueda ser utilizado por el algoritmo de optimización durante el segundo recorrido. A continuación, se detalla con un ejemplo los pasos de esta etapa:

- Cuando el prototipo registra un caso 1, significa que se encuentra navegando en línea recta con paredes en ambos lados. No es necesario filtrar esté caso ya que no tiene relevancia para el algoritmo de optimización
- Mientras el robot avanza en línea recta se puede detectar que no existe una pared derecha, esto significa que es un caso 2. Pero que sucede si en realidad es un caso 3 donde tendría que girar a la derecha obligatoriamente, el robot continúa avanzando en línea recta si encuentra una pared frontal se convierte en un caso 3 por lo que no se registra el

caso 2, pero en el caso de que no haya una pared frontal y los encoders marquen más de 200mm se registrará un caso 2.

- Existen ocasiones en las que el robot se despega de la pared izquierda, pero continuando con el algoritmo de *mano izquierda* éste hace un pequeño giro a la izquierda pudiendo registrar un caso 4 o un caso 5. Por lo que mediante los encoders se confirma si se trata de un giro real o solo una corrección de trayectoria, siendo este último caso el que se ignora.

El resultado de la etapa de filtrado es un arreglo que consta de los casos para la optimización de la toma de decisión, esté arreglo funciona como entrada para el algoritmo de optimización; desarrollado en el literal 2.4 de este trabajo de titulación.

A continuación, se utilizará un ejemplo ficticio para ver el funcionamiento de la etapa de filtrado.

Como se puede ver en la Figura 16, se registran los casos: 2, 5, 6, 1 y 2, estos casos tienen que pasar por la etapa de filtrado, en el orden en que fueron registrados. El primer caso 2 tiene que verificar que la distancia recorrida es mayor a 10cm, por lo tanto, se valida que es un caso 2 por lo que no se procede a eliminarlo. En el caso 5 se verifica la distancia recorrida en la curva, esta decisión es correcta por lo que no se procede a eliminar este caso. El caso 6 no necesita validación, por lo que solo procede con el giro de 180º.

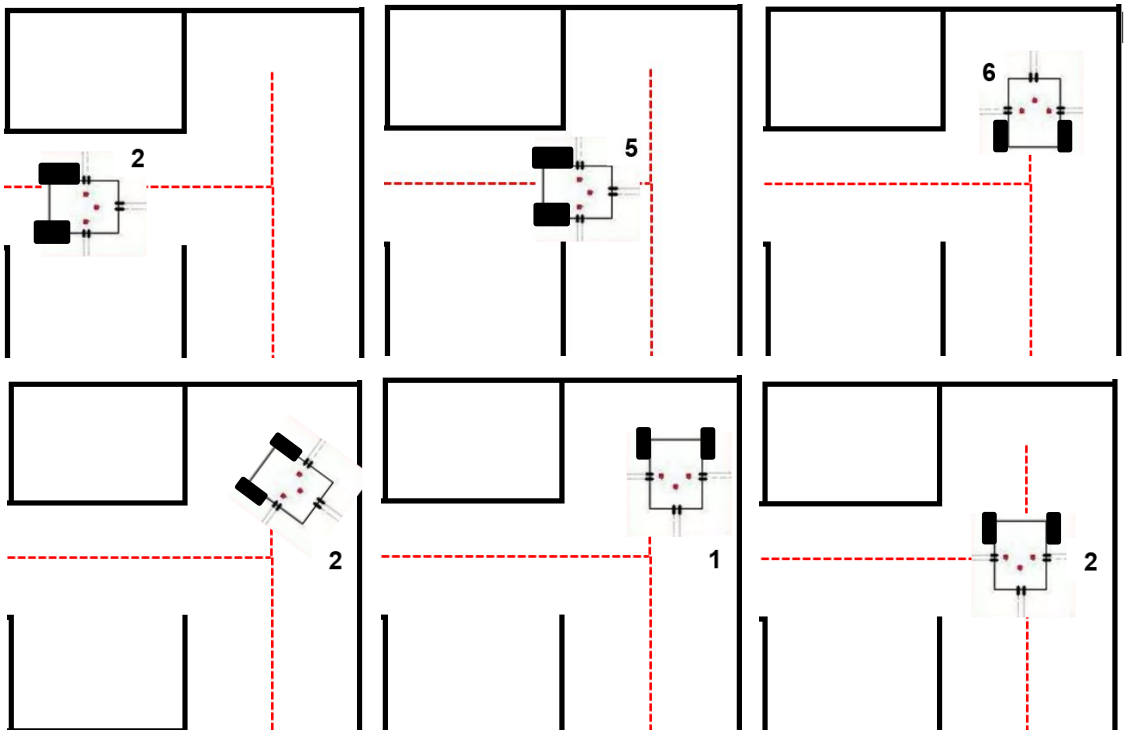

Figure 3.6. Ejemplo de ruta para la etapa de filtrado.

Al terminar el caso 6, el prototipo quedó en una “mala posición” por lo que detecta un caso 2, la validación verifica que la distancia recorrida es menor a 10cm, por lo tanto, éste caso es eliminado. Seguido el robot detecta un caso 1, el cual no se toma en cuenta ya que es un caso sin toma de decisión. Por último, en este ejemplo, el robot detecta un caso 2, el cual queda registrado ya que el prototipo ha recorrido más de 10cm dentro de éste.

El arreglo final en este ejemplo es el siguiente: 2, 5, 6, 2.

3.3 Implementación del Algoritmo de Optimización

Tabla 6.
Pesos de la decisión.

Decisión	Valor de Giro	Casos
S	0º	2
R	90º	3
L	270º	4, 5
B	180º	6

Para la implementación de este algoritmo se da valores a cada decisión, por ejemplo, la decisión “R” es equivalente a girar a la derecha y tiene un valor de 90º; estos valores están detallados en la Tabla 5. El objetivo del algoritmo es recorrer todo el arreglo en búsqueda de las decisiones “B”, las cuales son los que se busca optimizar. Al encontrar esta decisión se tendrá que sumar los valores de “B” más la decisión anterior y la siguiente, está suma nos da un valor que debe ser dividido para 360º, dando como resultado un ángulo el cual representa el caso que optimiza la ruta. Las tres decisiones seleccionadas para la suma de ángulos serán reemplazadas por la decisión que arroje el resultado, mientras tanto el arreglo va disminuyendo de tamaño y se va armando el camino optimizado. Se puede dar que la decisión resultado resulte “B”, por lo que es necesario siempre volver una decisión atrás para verificar dicho valor. Este algoritmo se encuentra detallado en la Figura 17.

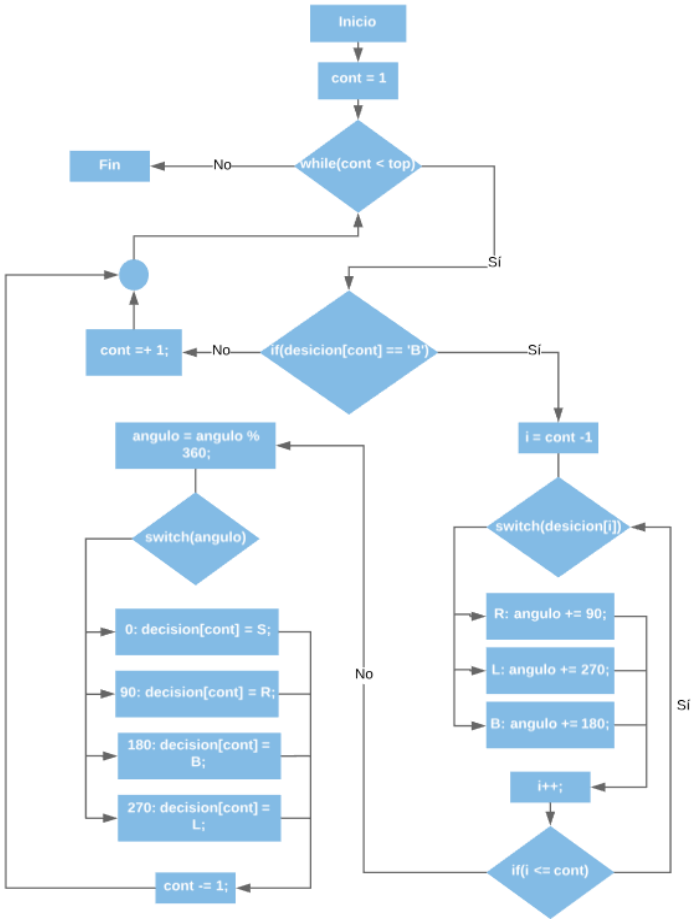

Figura 17. Algoritmo de optimización.

Una vez que este algoritmo haya finalizado, se tendrá como resultado un arreglo con las “*decisiones*” para la toma de decisiones que deberá recorrer el prototipo durante el segundo trayecto. Este arreglo final se enviará mediante comunicación serial al computador, para posteriormente cargar dicho arreglo al programa para el recorrido dos.

3.4 Implementación del Recorrido Optimizado

Para implementar el camino optimizado, se tiene que recorrer el arreglo resultado del algoritmo de optimización. En cada iteración que recorre dicho arreglo, se llamará a una función “solucionIzquierda” que permite aplicar el controlador PID, para que el robot avance en línea recta mientras detecte pared a cada lado; debido a que en este caso no se necesita de una toma de decisión. En cuanto el robot deje de detectar una pared izquierda o derecha, procede a verificar en que decisión del arreglo se encuentra para saber qué acción aplicar. Inmediato, procede enviar la decisión a una función, la cual mediante un condicional ejecuta el proceso respectivo para cada decisión.

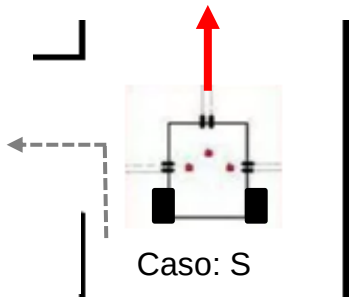

Figura 18. Ejemplo caso S.

A continuación, se detalla dicha lógica con un ejemplo. Mientras el robot se encuentra avanzando por el medio de los pasillos del laberinto y llega a detectar un espacio vació a su lado izquierdo, el prototipo se da cuenta que ha llegado a un punto de toma de decisión por lo que el programa salta a otra función para verificar que decisión es la siguiente. En este supuesto ejemplo, el siguiente caso es “S”, por lo que deberá avanzar recto en vez que curvar hacia la izquierda, como se ve en la Figura 18.

Para lograr aplicar este método, el prototipo tiene que navegar lo mejor posible por el centro de las paredes del laberinto y realizar los giros de una manera casi perfecta. Se logró esto gracias a la ayuda de los encoders, los cuales permiten colocar al robot en posición óptima para llevar a cabo un giro; ya que si utilizamos el método aplicado en el algoritmo de “mano izquierda” el prototipo puede llegar a detectar decisiones falsas. Gracias a los encoders, se puede hacer que el robot avance cierta distancia antes o después de realizar una curva; estas distancias dependerán del laberinto que se esté utilizando.

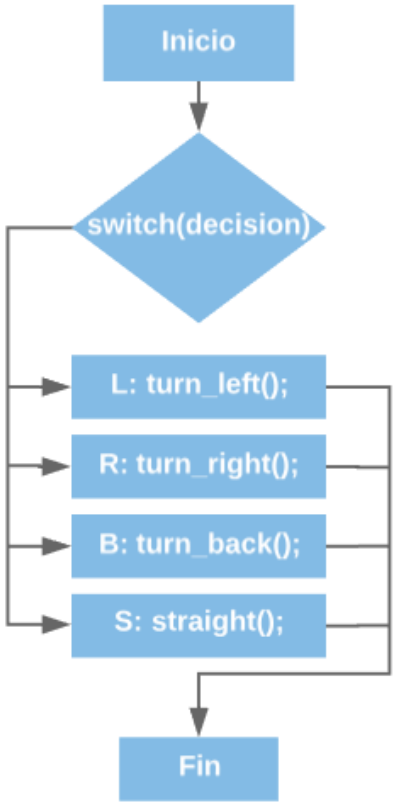

Figura 19. Función “recorrerCamino”.

La función “*seguirRecto*” es importante tenerla en cuenta, ya que, si no se aplica un control correcto durante el avance en línea recta del prototipo, éste puede avanzar en diagonal y posteriormente chocar con una pared del laberinto. Para esto, dicha función verifica si existe primero una pared a la izquierda para utilizarla como *set point.* Por el contrario, si no existe pared izquierda buscará una pared derecha para utilizar, caso contrario simplemente se ponen los

motores a velocidades iguales para que estos avancen en línea rectar durante cierta distancia. Cuando se tiene alguna de las dos paredes, se utiliza el control PID para avanzar en línea recta hasta encontrar la otra pared faltante; ya que se ha logrado pasar la decisión “S” con éxito.

A continuación, se explicará con un ejemplo ficticio el proceso de la optimización:

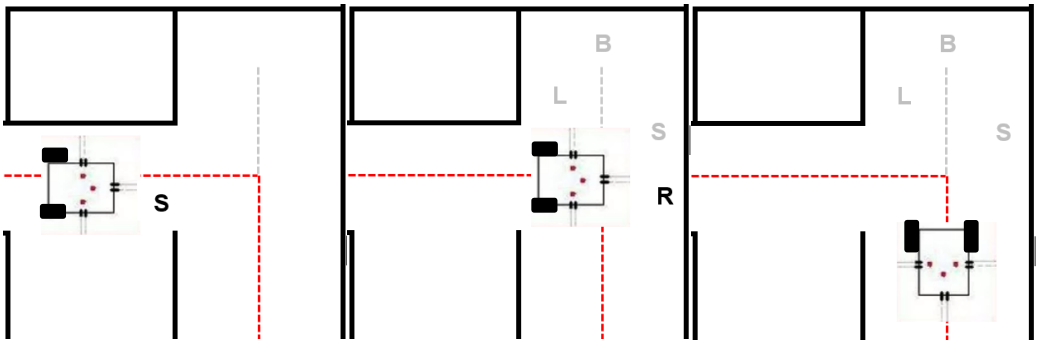

Figura 20. Recorrido ejemplo de ruta optimizada.

En la Figura 20, se puede apreciar el camino optimizado que recorre el prototipo en este ejemplo, este camino es el resultado de aplicar el algoritmo de optimización al arreglo “S”, “L”, “B”, “S”. Dado que ya se aplicó el algoritmo de optimización de tiene un arreglo “S”, “R” como resultado. Por lo tanto, al momento de que el prototipo culmina la primera decisión “S” verifica el arreglo con el camino optimizado, observando que la siguiente decisión es “R”, evitando así un camino innecesario y evitando entrar a 3 decisiones innecesarios.

4. Capítulo IV. Análisis y Resultados

En este capítulo, se realiza un análisis de los resultados durante la etapa de experimentación, en la cual se realizaron pruebas para comparar los algoritmos desarrollados anteriormente. Todas las pruebas se las realiza en un mismo ambiente, con la finalidad de que los resultados se han evaluados en función del tiempo. Finalmente, se demuestra la funcionalidad final del prototipo y su ventaja al aplicar algoritmos de optimización. El ambiente de pruebas se puede apreciar en la Figura 21.

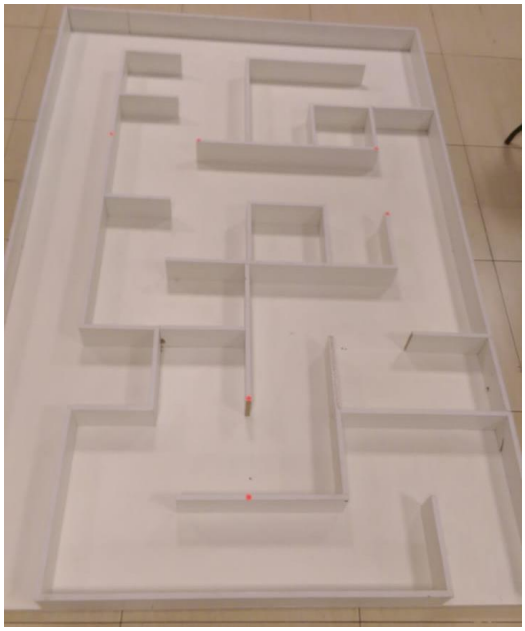

Figura 21. Laberinto de pruebas.

4.1 Primera Prueba

Para la primera prueba, se colocó al prototipo por la entrada izquierda del ambiente de pruebas. Se parametrizo que el tiempo debe empezar a correr cuando el robot inicie su desplazamiento y termine cuando el mismo salga del laberinto.

El primer recorrido dio como resultado un total de 24 segundos de recorrido, durante el cual el prototipo se desplazó por todas las decisiones del algoritmo *mano izquierda*; incluyendo caminos sin salida. A continuación, en la Figura 22 se muestra el trayecto del prototipo durante el primer recorrido.

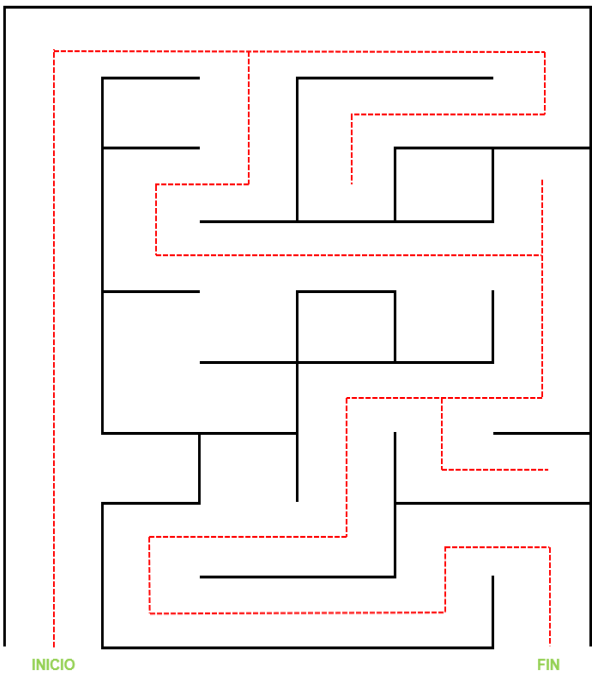

Figura 22. Recorrido algoritmo mano izquierda.

4.1.1 Etapa de filtrado

El primer recorrido dio como resultado el siguiente arreglo de casos antes de la etapa de filtrado; los datos se encuentran en el formato “[caso] – [distancia recorrida]”.

Casos antes de la etapa de filtrado:

[1 – 46, 2 – 22, 1 – 134, 2 – 13, 3 – 5, 1 – 0, 2 – 5, 1 – 27, 2 – 20, 1 – 51, 2 – 14, 3 – 4, 2 – 3, 1 – 1, 2 – 3, 1 – 4, 2 – 11, 3 – 4, 1 – 1, 2 – 4, 1 – 29, 4 – 1, 1 – 32, 6 – 4, 1 – 3, 6 – 11, 1 – 12, 2 -12, 3 – 5, 1 – 32, 4 – 2, 1 – 3, 4 – 1, 1 – 1, 2 – 1, 5 – 6, 1 – 1, 4 – 6, 1 – 9, 2 – 3, 1 – 59, 4 – 2, 1 – 6, 2 – 0, 1 – 0, 4 – 2, 2 – 12, 1 – 4, 2 – 14, 1 – 4, 2 – 13, 3 – 4, 1 – 0, 2 – 0, 1 – 1, 2 – 5, 4 – 2, 1 – 1, 4 – 2, 1 – 2, 4 - 2, 1 – 2, 4 – 3, 1 – 0, 4 – 6, 1 – 4, 4 – 2, 1 – 2, 2 – 4, 1 – 0, 2 – 1, 1 – 10, 2 – 18, 1 – 28, 2 – 18, 1 – 5, 5 – 2, 2 – 3, 5 – 1, 2 – 2, 1 – 23, 6 – 4, 1 – 2, 6 – 2, 1 – 0, 6 – 8, 2 – 0, 1 – 10, 2 – 22, 1 – 31, 2 – 12, 3 – 4, 1 – 4, 2 – 1, 5 – 13, 2 – 3, 5 – 10, 1 – 5, 2 – 6, 1 – 17, 6 – 3, 1 – 2, 6 – 3, 1 – 0, 6 – 8, 2 – 0, 1 – 12, 2 – 10, 3 – 5, 2 – 0, 1 – 0, 2 – 5, 5 – 6, 2 – 3, 5 – 8, 1 – 0, 4 – 8, 1 – 1, 2 – 7, 1 – 29, 2 – 17, 3 – 5, 2 – 1, 1 – 0, 2 – 5, 1 – 3, 2 – 22, 4 – 1, 1 – 4, 4 – 1, 1 – 3, 4 – 1, 1 – 2, 4 – 2, 4

– 2, 1 – 0, 4 – 2, 4 – 3, 4 – 2, 1 – 69, 4 – 1, 1 – 25, 2 – 12, 3 – 1, 3 – 0, 3 – 0, 3 – 0, 3 – 0, 3 – 0, 3 – 0, 3 – 0, 3 – 1, 2 – 0, 1 – 1, 2 – 1, 1 – 9, 2 – 9, 3 – 1, 3 – 1, 3 - 0, 3 – 0, 3 – 0, 1 – 2, 2 – 2, 1 – 21, 6 – 1, 6 – 1, 3 – 1, 3 – 2, 8 – 2, 8 – 2, 0 – 0]

Una vez que el programa corre la etapa de filtrado entrega como resultado un arreglo de decisiones, mostrado en la Figura 23.

Arreglo resultado de la etapa de filtrado:

[S, R, S, R, R, L, B, R, L, L, L, S, R, L, L, S, L, B, S, R, L, L, B, R, L, L, R, L, S, L, L, L, R, R]

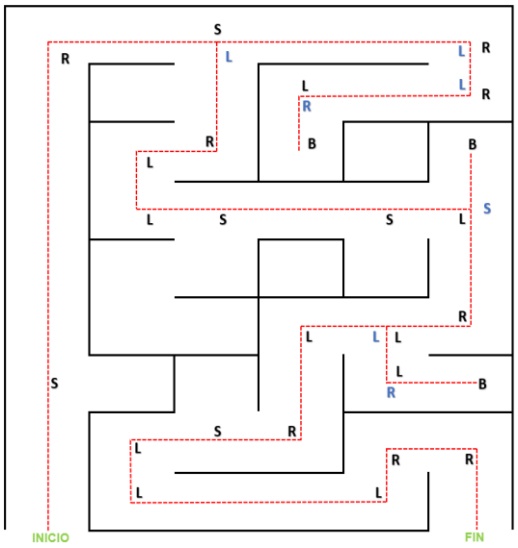

Figura 23. Recorrido de exploración con decisiones.

Una vez que los datos ingresaron al algoritmo de optimización, éste arrojo como resultado el arreglo de decisiones para la ruta optimizada; mostrada en la Figura 24.

Ruta optimizada:

[S, R, R, S, R, L, L, S, S, R, R, S, L, R, S, L, L, L, R, R, S]

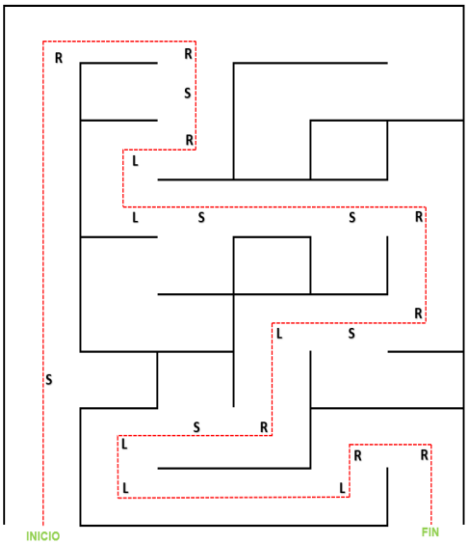

Figura 24. Recorrido a través de la ruta optimizada.

4.1.2 Segundo recorrido

Gracias a los resultados obtenidos en la etapa de filtrado y al algoritmo de optimización, se utilizó para el segundo recorrido el arreglo con la ruta más corta hacia el final del laberinto. En este caso el prototipo realizo todo el recorrido en un total de 14 segundos, al evitar los caminos "*sin salida*" el prototipo se pudo desplazar directamente a la salida recorriendo una menor distancia. El trayecto realizado por el robot se muestra en la Figura 25.

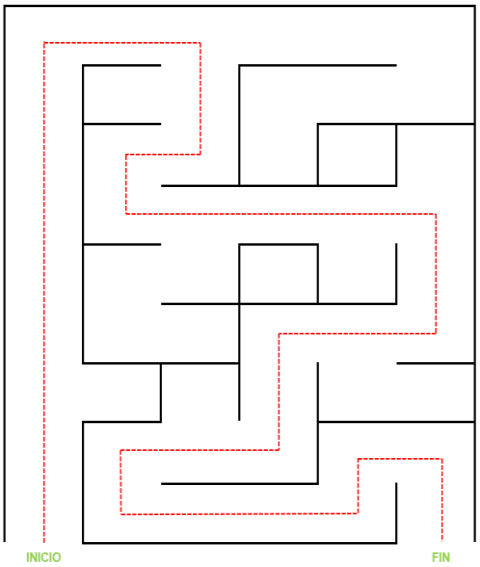

Figura 25. Recorrido de ruta optimizada.

4.2 Segunda prueba

La segunda prueba se llevó a cabo en el mismo laberinto, pero se cambió el punto de partida para validar que el robot es capaz de aplicar el mismo procedimiento en distintos ambientes. El primer recorrido con el algoritmo de *Mano Izquierda* resulto en un tiempo total de 29 segundos; cómo se puede apreciar en la Figura 26.

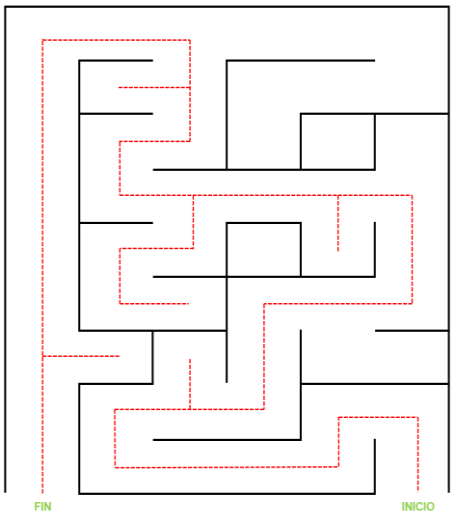

Figura 26. Algoritmo Mano izquierda.

4.2.1 Etapa de filtrado

En esta segunda prueba, el primer recorrido dio como resultado el siguiente arreglo:

[1 – 20, 4 – 1, 1 – 2, 4 – 6, 1 – 2, 4 – 3, 1 – 0, 4 – 8, 1 – 7, 4 – 4, 1 – 13, 2 – 6, 3 – 4, 1 – 55, 2 – 11, 3 – 4, 2 – 3, 3 – 3, 2 – 0, 1 – 6, 2 – 11, 3 – 4, 1 – 1, 2 – 3, 4 – 3, 1 – 0, 4 – 4, 1 – 2, 2 – 0, 1 – 1, 2 – 13, 1 – 12, 6 – 4, 1 – 2, 6 – 2, 1 – 0, 6 – 6, 2 – 0, 1 – 6, 4 – 3, 2 – 3, 5 – 3, 2 – 2, 5 – 12, 1 – 3, 2 – 0, 1 – 1, 2 – 6, 1 – 27, 2 – 17, 3 – 4, 2 – 2, 1 – 0, 2 – 5, 1 – 4, 2 – 19, 1 – 3, 4 – 2, 1 – 6, 2 – 0, 1 – 1, 2 – 1, 1 – 40, 4 – 6, 1 – 1, 2 – 1, 5 – 5, 2 – 3, 5 – 10, 2 – 15, 1 – 16, 6 – 16, 1 – 9, 4 – 3, 2 – 16, 1 – 35, 4 – 2, 1 – 4, 2 – 14, 1 – 2, 2 – 13, 3 – 5, 1 – 0, 2 – 7, 4 – 4, 1 – 2, 4 – 8, 1 – 2, 4 – 5, 1 – 9, 2 – 0, 1 – 1, 2 – 4, 1 – 20, 6 – 5, 1 – 1, 6 – 8, 1 – 14, 2 – 7, 3 – 5, 1 – 1, 2 – 5, 1 – 4, 2 – 13, 3 – 6, 1 – 0, 2 – 3, 4 – 7, 1 – 3, 4 – 13,

1 – 2, 2 – 2, 5 – 4, 2 – 7, 1 – 9, 2 – 6, 3 – 5, 1 – 0, 2 – 8, 1 – 4, 2 – 9, 3 – 5, 1 – 1, 2 – 3, 4 – 9, 1 – 1, 2 – 0, 5 – 10, 1 – 0, 4 – 6, 1 – 1, 2 – 15, 1 – 9, 6 – 6, 1 – 0, 6 – 7, 2 – 0, 1 – 6, 4 – 4, 2 – 3, 5 – 6, 2 – 0, 1 – 1, 5 – 6, 1 – 1, 4 – 5, 2 - 10, 1 – 195, 2 – 0, 1 – 2, 2 – 12, 1 – 10, 6 – 1, 6 – 0, 6 – 0, 6 – 0, 6 – 0, 6 – 0, 6 – 0, 6 – 0, 6 – 0, 1 – 4, 6 – 0, 6 – 0, 6 – 0, 6 – 0, 6 – 0, 6 – 0, 6 – 0, 3 – 0, 3 – 0, 3 – 0, 3 – 0, 3 – 0, 3 – 0, 3 – 0, 3 – 0, 3 – 0, 3 – 0, 3 – 0, 3 – 0, 3 – 0, 3 – 0, 2 – 0, 1 – 8, 4 – 4, 2 – 11, 1 – 55, 6 – 2, 8 – 1, 8 – 2, 8 – 2, 8 – 2]

Así como en la primera prueba, el arreglo de casos tiene que pasar por la etapa de filtrado y posteriormente por el algoritmo de optimización. El arreglo resultado de la etapa de filtrado se muestra en la Figura 27 y el recorrido optimizado en la Figura 28.

Arreglo resultado de la etapa de filtrado:

[L, L, R, R, R, L, B, L, L, R, S, L, L, L, B, L, L, R, L, L, B, R, R, L, L, R, R, L, L, B, L, L, L, L]

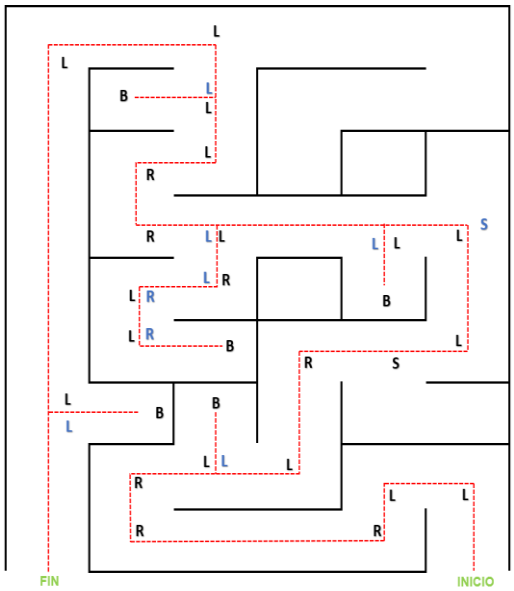

Figura 27. Recorrido con toma de decisiones.

Ruta optimizada:

[L, L, R, R, R, S, L, R, S, L, L, S, S, R, R, L, S, L, L, S]

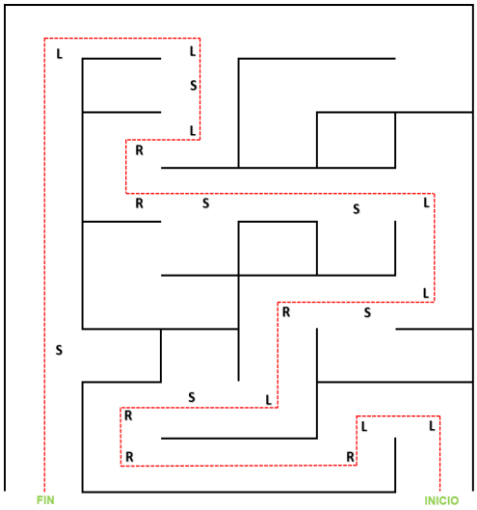

Figura 28. Recorrido a través de la ruta optimizada.

4.2.2 Segundo recorrido

En el segundo recorrido de esta prueba, el prototipo realizo todo el recorrido en un total de 16 segundos, al evitar los caminos "*sin salida*" el prototipo se pudo desplazar directamente a la salida recorriendo una menor distancia. El trayecto realizado por el robot se muestra en la Figura 29.

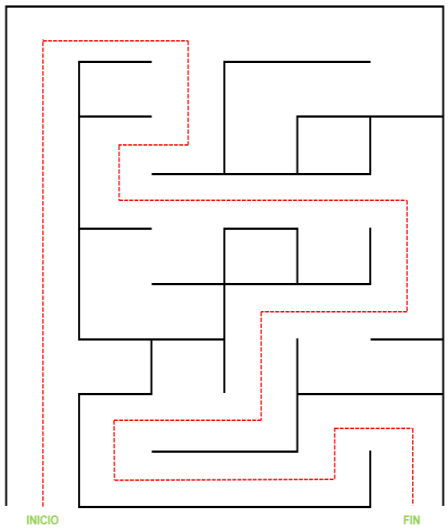

Figura 29. Recorrido de ruta optimizada.

Como se puede observar, el resultado de aplicar el algoritmo de optimización beneficia en mejorar el tiempo y la ruta en la que el prototipo llega a la meta, evitando así seguir casos que lleven a caminos "falsos" o "sin salida".

5. CONCLUSIONES Y RECOMENDACIONES

5.1 Conclusiones

Se llego a la conclusión, que el uso de un controlador PID es indispensable para un robot laberinto, éste permite al prototipo navegar a través del laberinto de una mejor manera, evitando colisiones con las paredes y permitiendo al robot moverse por el centro de los pasillos.

Existen diferentes maneras de representar un laberinto matemáticamente, puede ser como una matriz bidimensional, un arreglo de casos para llegar a la meta, una red de puntos interconectados, etc. Depende de la manera en la que se representa al laberinto, se pueden aplicar diferentes distintos algoritmos para cada caso.

Se concluye que el material de las ruedas del prototipo, el material de la base del laberinto y el polvo que exista dentro del mismo, afectan directamente a la tracción del robot.

Se concluye que, el trayecto del prototipo por el laberinto es el resultado directo del algoritmo que se aplique, dicho algoritmo será el encargado de manejar la toma de decisiones del robot en los diferentes casos que se llegue a encontrar.

Se llego a la conclusión de que el microcontrolador Atmega328p llega a su máximo de procesamiento al aplicar algoritmos de optimización, evitando así el aumento de velocidad motriz del prototipo.

En un prototipo de locomoción diferencial, los giros que se realizan pueden llevarse a cabo variando proporcionalmente una de las ruedas o invirtiendo el sentido de una de las ruedas. Debido al tiempo de reacción del microcontrolador es mejor utilizar la variación proporcional de una rueda cuando el giro es en la dirección de la pared que el robot utiliza como *Set Point,* por el contrario, si el giro es en la otra dirección es mejor que el prototipo se coloque en posición y gire en su propio eje.

La velocidad a la que se desplaza el prototipo influye en la detección de los casos en los dos algoritmos utilizados, esto depende también de los sensores que se utilicen, la velocidad de procesamiento y adquisición de datos del microcontrolador. Todo este conjunto de variables define la velocidad máxima a la que se puede desplazar el prototipo e influyen directamente en la fiabilidad para detectar todos los casos.

Se concluye que la lectura de los sensores a través de I2C tiene un tiempo de lectura muy alto para realizar una implementación más fiable.

5.2 Recomendaciones

Se recomienda la utilización de sensores de distancia infrarrojos por su tiempo de respuesta y fiabilidad en la medición, permitiendo al prototipo tener una mejor captura de datos y detectar correctamente todos los casos.

Se recomienda utilizar un microcontrolador con mejores prestaciones que el Atmega328p de 8 bits, el cual tiene una velocidad de procesamiento de 16MHz. Por lo que, recomiendo utilizar un microcontrolador de 32 bits con 76MHz de velocidad.

Se recomienda utilizar sensores que utilicen una interfaz análoga, esto se debe a que es más rápido realizar una conversión AD, que utilizar el protocolo I2C para que el microcontrolador se comunique con los sensores.

Para una futura implementación, recomiendo utilizar un control PID para mantener un nivel de velocidad constante del prototipo en función del nivel de voltaje de la batería, con la finalidad de que el prototipo no decrezca en velocidad mientras la batería se descarga.

Es recomendable tomar en cuenta la directividad de los sensores, si se colocan los sensores de forma diagonal apuntando hacia el frente, el robot puede saber de manera adelantada a que caso se acera, por lo que podría aumentar su velocidad.

Se recomienda la utilización de ruedas de espuma o de caucho para el prototipo, ya que estás permiten tener una mejor tracción en la superficie de madera de los laberintos convencionales. Además, se recomienda mantener limpia la superficie del laberinto, para evitar la acumulación de polvo o suciedad en las ruedas durante el trayecto del robot.

REFERENCIAS

Al-Khawaldah, M., Badran, O., & Al-Adwan, I. (2011). Exploration algorithm technique for multi-robot collaboration. *Jordan Journal of Mechanical and Industrial Engineering*, *5*(2), 177–184.

Ave, B., Number, D., & Date, R. (2013). MPU-6050_DataSheet_V3 4.pdf, *1*(408).

Bienias, Ł., Szczepański, K., & Duch, P. (2016). Maze Exploration Algorithm for Small Mobile Platforms. *Image Processing & Communications*, *21*(3), 15–26. https://doi.org/10.1515/ipc-2016-0013

Caicedo, D., & Trijillo, L. (2016). Diseño e Implementación de Dos Robots de Competencia, 244. Recuperado de http://bibdigital.epn.edu.ec/handle/15000/16546

Discant, A., & Rogozan, A. (2007). Sensors for obstacle detection-a survey. *… Spring Seminar on*, 100–105. Recuperado de http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4432828

Garg, P. (2016). Breadth First Search Tutorials & Notes | Algorithms | HackerEarth. Recuperado el 31 de octubre de 2018, de https://www.hackerearth.com/practice/algorithms/graphs/breadth-first-search/tutorial/

Gupta, B., & Sehgal, S. (2015). Survey on Techniques used in Autonomous Maze Solving Robot Survey. *2014 5th International Conference - Confluence The Next Generation Information Technology Summit (Confluence)*, (SEPTEMBER 2014), 323–328. https://doi.org/10.1109/CONFLUENCE.2014.6949354

Hualong, J., Hongqi, W., & Yonghong, T. (2011). Design and realization of a maze robot. *2011 International Conference on Consumer Electronics, Communications and Networks, CECNet 2011 - Proceedings*, 201–

204. https://doi.org/10.1109/CECNET.2011.5768942

IBEC Corporation. (2018). REVOLUCIÓN ROBÓTICA EN ECUADOR Time To Talk eEducation - IBEC TECNOLOGÍA / GADGETS Editor. Recuperado el 19 de diciembre de 2018, de http://www.ibecmagazine.com/TECNOLOGÍA/TabId/459/ArtMID/1165/ArticleID/1067/REVOLUCI211N-ROB211TICA-EN-ECUADOR.aspx

Iovine, J. (2000). Mcgraw-Hill - Pic Microcontroller Project Book By John Lovine - (2001).Pdf. McGraw-Hill. Recuperado de http://mirror.thelifeofkenneth.com/lib/electronics_archive/PIC-microcontroller-project-book-With-PICBASIC.pdf

Kirsh, P., Lis, S., Esslinger, C., Gruppe, H., Danos, P., Broll, J., … Gallhofer, B. (2006). Brain Activation during Metal Maze Solving. *Neuropsychobiology*, *54*, 51–58. Recuperado de https://doi.org/10.1159/000095742

Microchip Technology Inc. (2018). ATmega328P - 8-bit AVR Microcontrollers - Microcontrollers and Processors. Recuperado el 12 de noviembre de 2018, de https://www.microchip.com/wwwproducts/en/ATmega328P

Morales, L., Pozo, D., Rosero, J., Sandobalin, S., & Rodríguez, M. (2014). Mapeo de Laberintos y Búsqueda de Rutas Cortas MedianteTres Mini Robots Cooperativos. *Revista Politécnica*, *34*(1), 101–106.

Mosa, A. N., & Saad, A. M. (2005). MicroMouse : An Intelligent Maze Solver Robot, 1–7.

Ogata, K. (1998). *Ingeniería de Control Moderna*. *Pearson* (Vol. 3).

Pololu Corporation. (2018). Pololu - 8.e. Simplifying the Solution. Recuperado el 31 de octubre de 2018, de https://www.pololu.com/docs/0J21/8.e

Pololu Corporation. (2018). Pololu - Magnetic Encoder Pair Kit for Micro Metal Gearmotors, 12 CPR, 2.7-18V (HPCB compatible). Recuperado el 13 de noviembre de 2018, de https://www.pololu.com/product/3081/specs

Pololu Corporation. (2018). Pololu - VL6180X Time-of-Flight Distance Sensor Carrier with Voltage Regulator, 60cm max. Recuperado el 12 de noviembre de 2018, de https://www.pololu.com/product/2489/specs

Pololu Corporation. (2019). TB6612FNG Dual Motor Driver Carrier. Recuperado el 30 de enero de 2019, de https://www.pololu.com/product/713

Popirlan, C., & Dupac, M. (2009). An Optimal Path Algorithm for Autonomous Searching Robots. *Annals Math. Comp. Sci. Ser*, *36*(1), 37–48.

Rahman, M. (2017). *Autonomous maze solving robot*. University ofLiberal Arts Bangladesh. https://doi.org/10.13140/RG.2.2.34525.82403

RobdosTeam. (2016). ¿Qué es una IMU? Recuperado de http://www.robdosteam.com/wp-content/uploads/2016/12/08.IMU_.pdf

Robinette, P., Li, W., Allen, R., Howard, A. M., & Wagner, A. R. (2016). Overtrust of robots in emergency evacuation scenarios. *ACM/IEEE International Conference on Human-Robot Interaction*, *2016–April*(February 2018), 101–108. https://doi.org/10.1109/HRI.2016.7451740

Rührup, S., & Schindelhauer, C. (2006). Online multi-path routing in a maze. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *4288 LNCS*, 650–659. https://doi.org/10.1007/11940128_65

Sedgewick, R., & Wayne, K. (2018). Undirected Graphs. Recuperado el 30 de octubre de 2018, de https://algs4.cs.princeton.edu/41graph/

Takahashi, T., & Yamanaka, M. (1987). Features of a magnetic rotary encoder. *IEEE Transactions on Magnetics*, *M*(5), 2182–2184. https://doi.org/10.1109/TMAG.1987.1065634

Vannoy, R. (2009). Design a Line Maze Solving Robot, (April).

Venkata, P. P. K., Bose, S. K., Sarode, D. M., Shete, P. P., Apte, A. G., & Shaik, K. (2011). Automated maze solving using fluid mechanics based numerical approach. 2011 International Conference on Image Information Processing, (Iciip), 1–6. https://doi.org/10.1109/ICIIP.2011.6108920