Desarrollo de un algoritmo para la evasión de obstáculos dinámicos con un robot móvil con técnica de aprendizaje por refuerzo

AUTOR

Lucas Gándara Vega

Trabajo de grado presentado como requisito para optar al título de:

INGENIERIA MECATRÓNICA

Director:

IE Velasco Vivas Alexandra PhD.

Codirector:

ING Velasco Toledo Nelson MSc.

UNIVERSIDAD MILITAR NUEVA GRANADA

FACULTAD DE INGENIERÍA

PROGRAMA INGENIERÍA MECATRÓNICA

BOGOTÁ, 11 DICIEMBRE 2020

Universidad Militar Nueva Granada

Facultad de Ingeniería

Programa de Ingeniería Mecatrónica

Desarrollo de un algoritmo para la evasión de obstáculos dinámicos con un robot móvil con técnica de aprendizaje por refuerzo

Bogotá D.C., 30 de octubre de 2020

Gándara Vega Lucas

Velasco Vivas Alexandra

Velasco Toledo Nelson

Dedicatoria

Dedico este proyecto a mis padres, que siempre me han apoyado y quienes trabajan todos los días para brindarme la educación que hizo posible este trabajo.

TABLA DE CONTENIDO

1 INTRODUCCIÓN

La conducción autónoma permite a un vehículo el desplazamiento de un lugar a otro sin la intervención humana. Esta tarea de definición simple conlleva todo un reto tecnológico para el cual están definidos niveles en los que se clasifica cuan versátil es algún tipo de intento por alcanzar esta autonomía con números del 1 al 5, donde 1 es un vehículo sin autonomía alguna y 5 es un vehículo con autonomía completa [1]. Actualmente los vehículos más avanzados han sido desarrollados por empresas como Waymo o Tesla y se encuentran en el nivel 3, estos cuentan con la capacidad de monitorizar el entorno para responder con ciertas situaciones imprevistas por lo que el vehículo será capaz de seguir una ruta y evadir cierto tipo de obstáculos inesperados. Este interés de lograr la autonomía de los vehículos se genera a partir de las cifras de muertes en accidentes automovilísticos que asciende a más de 1.3 millones en todo el mundo en el año 2016 donde el 99% de estos accidentes se atribuyen a un error humano o a una desatención al camino [2].

Antes de atacar directamente este problema en vehículos urbanos, se realizan experimentos más controlados en ambientes de laboratorio con robots móviles más pequeños donde la problemática sigue siendo la misma, la de llegar de un lado a otro sin la necesidad de la supervisión de un humano. A lo largo de los años se han desarrollado varias técnicas que solventan estas problemáticas. En cuanto a la planeación de trayectorias existen muchos métodos como el de los campos potenciales que es una buena aproximación, sin embargo, es también conocida porque presenta problemas de estancamientos en los puntos donde se presentan mínimos locales que son circunstancias en las que incluso con obstáculos estáticos llevan a que el móvil en cuestión no llegue nunca a su destino final [3]. De la misma forma en el campo de la evasión de obstáculos, existen algoritmos avanzados como el aprendizaje por refuerzo con redes neuronales los cuales han mostrado avances sorprendentes en los que se empieza con un móvil en un ambiente totalmente desconocido y se llega a que este se desplace por el entorno de manera muy acertada [4] aunque no perfecta, debido a que el desempeño dependerá de las condiciones a las que fue sometido el entrenamiento de la red neuronal. Con base en todo lo anterior se infiere cuanto trabajo aún queda por delante para llegar al nivel 5 de autonomía. Es por esto que aún se necesitan propuestas y soluciones para esta problemática.

Así como existen muchos algoritmos que realizan estas tareas de planeación y evasión de obstáculos, existen varias opciones de robots móviles disponibles en el laboratorio de robótica de la Universidad Militar Nueva Granada. El presente trabajo está estructurado de manera que primero se haga la selección del robot de acuerdo con el espacio, recursos presentes y disponibilidad de estos, luego se

realizan pruebas con el robot seleccionado para comprobar su funcionamiento, detectar y solucionar posibles comportamientos que puedan causar errores futuros en lo que respecta al movimiento. Después se realiza la selección, implementación y pruebas del algoritmo de planeación de trayectorias teniendo en cuenta el estado del arte de este campo y comparando los posibles candidatos. Posteriormente, se escogen que circunstancias serán necesarias para que el móvil deje de seguir su trayectoria y ceda el control al algoritmo de evasión de obstáculos y se implementa un algoritmo para este propósito. Por último, se selecciona el algoritmo de evasión de obstáculos teniendo en cuenta que este debe utilizar algún tipo de inteligencia artificial, con estos 3 algoritmos se realizan las pruebas con las que se dará el análisis y conclusiones del presente documento.

1.1 OBJETIVOS

1.1.1 Objetivo general

Diseñar una estrategia de toma de decisiones que se integre con un algoritmo de planeación de trayectorias para la evasión de obstáculos dinámicos específicos.

1.1.2 Objetivo especifico

- Implementar un algoritmo de planeación de trayectorias el cual suministre, a partir de un entorno establecido, un camino a seguir evitando estancamientos.
- integrar un algoritmo de evasión obstáculos tipo toma de decisiones para la evasión de ciertos obstáculos dinámicos.
- Evaluar el algoritmo desarrollado para cada uno de los obstáculos propuestos y medir su desempeño con criterios como tiempo de ejecución, distancia recorrida, entre otros.

2 INVENTARIO Y SELECCIÓN DE PLATAFORMA MÓVIL:

La selección del robot móvil para el presente trabajo tiene dos criterios con diferente importancia. El primero de ellos son los recursos con los que cuenta la Universidad Militar Nueva Granada, institución donde se realizará el mismo. Para su evaluación se hace una descripción de los robots presentes en el laboratorio de robótica de la Universidad Militar Nueva Granada y sus características, descritas en las subsecciones siguientes, además de un cuadro comparativo de los detalles técnicos, mostrados en la Tabla 1. Seguidamente, se procede a evaluar su disponibilidad dado que éstos son utilizados para el desarrollo de algunas asignaturas y sus prácticas de laboratorio, se hace la consulta directamente con el profesor encargado y con el laboratorista sobre cuáles de estos robots tienen alguna restricción de uso y cuáles son los más solicitados por los estudiantes.

Otro de los criterios a evaluar es la programación del robot, algunos tienen sus propias interfaces o cuentan con sensores únicos, para este criterio se tiene en cuenta cuáles son las más enfocadas y utilizadas para la investigación en aplicaciones de robótica móvil.

2.1 ROBOTS

2.1.1 Bioloid

El robot BIOLOID, mostrado en la Figura 1 es un kit de robots educacional, el cual permite aprender las estructuras básicas y principios de teoría de robótica como las junturas de los robots. Se pueden estudiar sus aplicaciones a la Ingeniería trabajando conceptos como cinemática inversa y directa. Debido a que consta de muchas partes y está pensado para ser ensamblado a gusto del usuario, tiene varias configuraciones móviles entre las cuales está la llamada “Smart Car” [5].

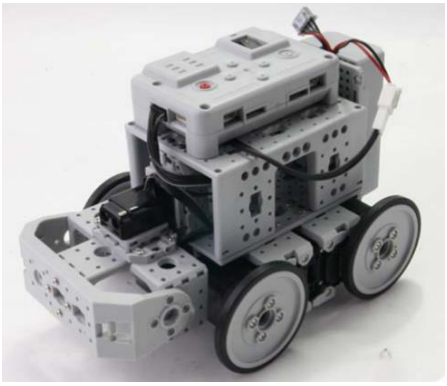

Figura 1. Smart Car. Tomada de: http://emanual.robotis.com/docs/en/edu/bioloid/premium/

Para su control, este robot utiliza su propio controlador llamado CM-530, el cual es el encargado de todos sus controles de movimiento y gestión de sus periféricos como entradas ADC, GP I/O, o su control remoto. Para su sustento cuenta con una batería Li-Po de 11.V 1000mA. Para su programación, debe hacerse por medio de su propio software llamado Roboplus el cual contiene módulos de animación de movimiento 3D, motor de cinemática inversa, programación de tareas por medio de lenguaje C, entre otros y su interfaz de comunicación es vía USB o Bluetooth.

2.1.2 PIONEER P3-DX

El Pioneer p3-dx, mostrado en la Figura 2 es un robot móvil diferencial completamente programable utilizado en la investigación. Debido a que cuenta con un control de alta precisión y unos sensores de ultrasonido en la parte delantera, es apto para tareas de navegación y evasión de obstáculos.

Figura 2. Robot PIONEER P3-DX. Tomado de: https://www.generationrobots.com/en/402395-robot-mobile-pioneer-3-dx.html

Cuenta con su propio framework para su programación, el cual incluye simulador, interfaz de usuario para monitoreo y manipulación del robot. Además, es personalizable, su plataforma superior soporta una carga de hasta 17kg de peso, esto permite su acople con diferentes accesorios como un manipulador convirtiéndolo en un robot hibrido o una cámara para tareas de mapeo, estas dos siendo las dos configuraciones más populares para el robot [6].

2.1.3 Arduino robot

El Arduino robot, mostrado en la Figura 3 es un robot móvil diseñado por Arduino, fue diseñado con la premisa de ser una puerta fácil de atravesar para la introducción a la robótica, su diseño es completamente abierto por lo cual se puede obtener información de cada una de sus partes e incluso hacer modificaciones a conveniencia. Cuenta con dos controladores Atmega32u4 que son programables con Arduino IDE.

Figura 3. Arduino robot. Tomado de: https://store.arduino.cc/usa/arduino-robot4

Cada controlador tiene asignada una tarea, un controlador se encarga del control de los motores y el otro tiene como tarea decidir la operación de acuerdo con lo captado con los sensores. Los sensores con los que cuenta el Arduino Robot son una brújula y 5 sensores infrarrojos ubicados todos mirando hacia el suelo. Debido a su conformación, este robot, tal y como viene, puede realizar tareas básicas de movimiento, seguimiento de líneas, entre otras [7].

2.1.4 Lego Mindstorm NTX

El Mindstorm es un kit de robótica didáctica creado por LEGO. Con este kit se pueden crear variedades de configuraciones de robots a gusto del usuario. Se controla mediante un procesador ARM9 instalado en una tarjeta llamada ladrillo programable NTX el cual soporta hasta 4 sensores y 3 motores.

Figura 4. Lego Mindstorm NTX. tomado de https://www.lego.com/es-es/product/lego-mindstorm

Su programación se realiza con un programa propio llamado NTX Mindstorms, pero también se pueden hacer aplicaciones en Simulink de Matlab [8].

2.1.5 Robotino

El Robotino es un sistema didáctico móvil para la investigación educativa creada por Festo. Debido a sus características es un robot muy completo para fines tanto didácticos como de investigación. Su sistema es adecuado para aplicaciones de nivel superior en robótica y debido a su completo equipamiento de actuadores y sensores lo hacen un candidato apto para fines de investigación. Una de sus ventajas es su variedad de entornos para su programación, entre los cuales está LabVIEW, Robotino view, C++, Matlab y soporta frameworks como ROS o .Net.

Figura 5. Robotino. Tomado de https://ip.festo-didactic.com/InfoPortal/Robotino/Overview/EN/index.html

Su sistema de transmisión omnidireccional le permite la traslación en cualquier sentido, además de giros sobre sí mismo. Cada una de sus tres unidades de accionamiento son independientes y se componen de un motor, un encoder incremental, un reductor y ruedas. Su lista de sensores incluye: parachoques, sensores de distancia, giroscopio, encoder, cámara, sensores ópticos, sensores inductivos, entre otros. Tiene interfaces WLAN, USB, Ethernet, VGA, Motor/encoder, entre otros. Para su alimentación utiliza dos baterías de plomo/ácido de 12V con capacidad de 9.6 Ah, cada una para un suministro de 24 V al Robotino. Tiene su propio PC interno con sistema operativo Linux Ubuntu [9].

2.1.6 Turtlebot 2

Turtlebot 2 es un robot móvil de bajo costo y de código abierto. Está pensado para navegar por entornos cerrados como una casa y debido a su interfaz y hardware, se utiliza principalmente para fines investigativos.

Figura 6. Turtlebot 2. Tomado de: https://www.turtlebot.com/turtlebot2/

Con su adquisición se incluye un entorno de desarrollo para escritorio, librerías de visualización, planeación de trayectorias, percepción y control de movimiento. Su principal ventaja es su sensor Kinect, el cual le permite "ver" su entorno y es utilizado para su navegación. Su programación se hace a través del framework ROS, el cual contiene módulos control, navegación, planeo de trayectorias, etc.

Al igual que todos los Turtlebot, este es un kit robótico de código abierto, es decir, su diseño es de libre acceso. El Turtlebot 2 se conforma de una base Kobuki [10] que desempeña velocidades máximas de hasta 70cm/s y cargas hasta de 5kg. Una netbook que tenga acceso a ROS (como una DragoBoard-410c), una estructura Turtlebot y una base para sensor Kinect [11].

2.1.7 Familia Turtlebot 3

Es el robot de acceso libre más popular para educación e investigación [12]. La generación TurtleBot3 son robots móviles pequeños de bajo costo, totalmente programable y basado en ROS. Está destinado a ser utilizado para la educación, la investigación, hobby y la creación de prototipos.

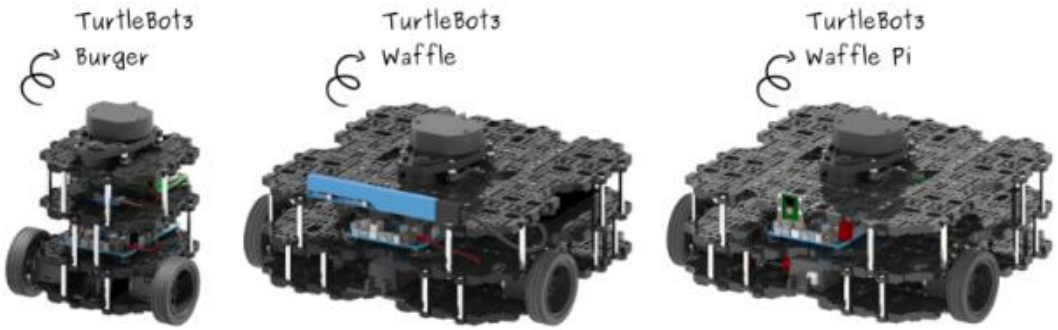

Figura 7. Familia Turtlebot 3. Tomado dehttps://www.turtlebot.com/learn/

Al igual que el Turtlebot 2, este incluye paquetes para navegación, SLAM e incluso planeación de trayectorias. Gracias a su sensor LIDAR es capaz de censar objetos a 360° lo que lo hace un candidato apto para tareas de detección y evasión de obstáculos. De los 3 modelos mostrados en la Figura 7, solo vienen el modelo Burger y Waffle Pi debido a que el modelo Waffle está descontinuado. Una de las diferencias entre los modelos son sus medidas y los sensores con los cuales vienen incluidos.

Toda la familia de Turtlebot mostrados en la Figura 7 son programables en ROS y son de configuración diferencial, cada uno tiene su propio sistema de alimentación por lo que pueden ser autónomos. Dado que todos en sus especificaciones de producto están diseñados para la navegación en entornos cerrados, la selección se realizará por la manera de reconocer su entorno debido que es necesario para la detección de los obstáculos que posteriormente deberá evitar. A continuación, se presenta una tabla con sus diferentes características las cuales serán evaluadas para la posterior selección para el presente trabajo, se tendrá en cuenta el objetivo final del mismo. Téngase en cuenta que los sensores serán divididos en dos grupos, los de percepción, que son utilizados por el robot para tener información del entorno y los de propiocepción los cuales son para captar información del estado actual del robot.

2.1.8 Detalles técnicos

Tabla 1. Características técnicas de los robots presentados. Elaboración propia.

Aspecto / Robot	Controlador/ PC	Percepción / Propiocepción	Voltaje de operación	Interfaces	Plataformas	Configuración
Bioloid	CM-530(propio)	IR, distancia. / Acelerómetro.	18v	USB, bluetoohth.	Roboplus	Diferencial
Pioner p3-dx	Pioner DX/AT	Distancia (ultrasonido) / Encoder.	12V	Usb, ethernet	ARIA, MobileSim, MobileEyes, Sonarnl, ROS.	Diferencial
Arduino Robot	2 x ATmega32u4	5 x Infrarrojos seguidores de línea. / No cuenta.	5 V	USB.	<5 Arduino IDE.	Diferencial
LEGO Mindstorm NTX	Ladrillo programable NTX	Infrarrojo. / No cuenta.	9V	USB, Bluetooth	LabView, Matlab, Robotlab, etc	Diferencial
Robotino	PC interno 4 núcleos, 64Gb SSD.	Distancia, cámara, óptico, inductivo. / Paragolpes, giroscopio,	24v	WLAN, USB, ETHERNET, VGA, PCI.	MATLAB, ROS, LabVIEW, API2 C/C++	Omnidirecciona l
TURTLEBOT 2	DragoBoard-410c	Kinect / Encoder.	3.3, 5 y 12v	USB	ROS.	Diferencial
TURTLEBOT 3 Wafle	OpenCR1.0, Raspberry pi	360° LiDAR, Intel real sense. / Inercial de 9 ejes (IMU).	3.3, 5 y 12v	USB	ROS.	Diferencial
TURTLEBOT 3 Wafle pi	OpenCR1.0,	360° LiDAR, cámara. / Inercial de 9 ejes.	3.3, 5 y 12v	USB	ROS.	Diferencial
TURTLEBOT 3. Burger	OpenCR1.0, Raspberry pi.	360° LIDAR, Inercial de 9 ejes. / Inercial 9 ejes.	3.3, 5 y 12v	USB	ROS.	Diferencial

2.2 SENSORES

En esta sección se presentan los sensores con los que cuentan los robots expuestos en la sección anterior.

2.2.1 LIDAR (Lase Imaging Detection and Ranging)

Disponible en la familia Turtlebot 3, es un sensor de luz el cual permite medir distancias desde su emisor láser hasta objetos o superficies, tiene un rango angular de 360° ± 1° y una distancia de detección de 12 cm a 350 cm. Los datos los adquiere midiendo el tiempo entre la transmisión de un haz de láser y su señal reflejada, lo cual se traduce en puntos de profundidad que hacen de este sensor un candidato apto para tareas de SLAM y navegación en robótica móvil [6].

Figura 8. LIDAR. Tomado de: https://www.unmannedtechshop.co.uk

2.2.2 Xtion Pro-Live

Disponible en Turtlebot 2, Xtion PRO-LIVE emplea un sensor de infrarrojos para la detección adaptativa de la profundidad, el color y sonido para capturar los movimientos del usuario en tiempo real. Esta solución integra unas herramientas para facilitar la creación de aplicaciones basadas en movimiento sin necesidad de hacer uso de complicados algoritmos. Es ideal para el diseño de juegos dado que incorpora módulos de reconocimiento de gestos, tracking corporal. Además, la captura de imágenes puede ser utilizada para aplicaciones con detección de personas o señalización digital [13]. Tiene un campo de visión de 58° H, 45° V, 70° D (Horizontal, Vertical, Diagonal), con rango de 0.8 y 2.5m de profundidad y su programación puede ser en c++/c# o java.

Figura 9. XTion Pro-Live. tomado de: https://www.asus.com/es/3D-Sensor/Xtion_PRO_LIVE/gallery/

2.2.3 Cámara Raspberry pi V2.1

Es un módulo diseñado especialmente para tarjetas Raspberry pi. Consiste en una cámara de foco fijo con resolución de 5Mpx, compatible con el sistema operativo Raspbian, específico de las Raspberry pi y dado que es una cámara, su utilidad depende del usuario [14].

Figura 10. Cámara Raspberry pi V2.1. Disponible en: https://nettigo.eu/products/raspberry-pi-camera-module-v2-8mpix

2.2.4 Intel Realsense R200

Es una cámara de alta definición con un LIDAR de estado sólido para la combinación de imágenes de alta calidad a color y mapas de profundidad. Es compatible con ROS; su programación puede ser en Python, c++, LabVIEW, entre otros. Dadas sus características la hacen un candidato óptimo para tareas de robótica autónoma [15] que requieran precisión en todo su rango de operación.

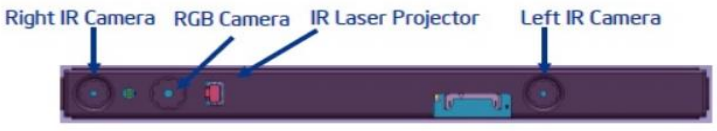

Figura 11. Intel Real Senste 200.

Tomado de: https://www.intel.la/content/www/xl/es/support/articles/000016214/emerging-technologies/intel-realsense-technology.html

2.3 SELECCIÓN DE LA PLATAFORMA MÓVIL

En la sección anterior se presentaron las plataformas móviles con los sensores disponibles en el laboratorio de robótica de la Universidad Militar Nueva Granada. Al tener una variedad de opciones posibles, fue necesario definir ciertas características con las que se seleccionó el robot móvil a utilizar. El objetivo último es el de la evasión de obstáculos dinámicos, por lo cual es necesario que el robot sea capaz de moverse a necesidad y que tenga un control de velocidad. Uno de los puntos a realizar para cumplir el objetivo es la validación del modelo del robot para luego ser utilizado, por lo que es necesario que el robot cuente con un control de actuadores que permitan esta tarea.

Además, es una ventaja que el móvil tenga visión 360° para la detección de cualquier movimiento no previsto en cualquier posición, esto con el fin de tener información del entorno siempre disponible para la posible aparición de nuevos obstáculos que son los que aparecerán para la comprobación del objetivo final. Aunque es indiferente el software con el cual interactúa el robot se tendrá en cuenta que este sea compatible con ROS, en la siguiente sección se explica la importancia de este framework y como funciona. Paralelamente, el tamaño del robot debe ser acorde con el espacio de trabajo y dado que el espacio otorgado en el laboratorio de robótica de la Universidad Militar Nueva Granada es relativamente pequeño para los tamaños de los robots en cuestión, un móvil pequeño sería adecuado.

Por último, se presenta una tabla con las características descritas y se comprueba si cada uno de los robots listados cumplen para la posterior selección de este. Además, se cuantifica la importancia de cada criterio por puntajes de la siguiente manera: El peso más alto para la selección lo tiene el control de cada actuador con 35 puntos debido a su importancia para la validación del modelo, la compatibilidad con Ros se le asignó un peso de 10 puntos por las ventajas que ofrece este framework, la visión 360 también tiene el peso más alto de 35 puntos por la naturaleza de la tarea a realizar es importante la detección en todas las direcciones y tanto las dimensiones del robot como sus suministros de sensores se le asignó un peso de 5 debido que son características adicionales que facilitan la tarea del desarrollo del trabajo, el tamaño porque entre más pequeño sea el robot más se aprovecha el espacio y entre más sensores más información del entorno se tiene. Para llenar la tabla se sombrea la casilla de color azul en caso de que se cumpla el criterio, de lo contrario se sombrea de color rojo. La suma de todos es 100, tal que 100 es el mejor.

Tabla 2. Comparación de robots presentados. Elaboración propia.

Característica Robot	**Control de cada actuador. [35]**	**Plataformas. [10]**	**Visión 360. [35]**	**Dimensiones (largo x ancho x alto) [cm] [5]**	**Sensores (percepción)/ Ampliación sensores [5]**	**Puntaje Total**
Bioloid	Si	Roboplus.	No	12 x 21 x 12	IR, distancia, acelerómetro. /No	40
Pioner P3-DX	Si	ARIA, MobileSim, MobileEyes, Sonarnl, ROS	No	45.5 x 38.1 x 23.4	Encoder, distancia (ultrasonido) / Si	
Arduino Robot	Si	Arduino IDE.	No	19cm diámetro x 10cm altura	Brújula, 5 x infrarrojos seguidor de línea. / Si.	45
Lego Mindstorm NTX	Si	LabView, Matlab, Robotlab, etc.	No	20 x 19 x 13	Infrarrojo. /No	40
Robotino	No	MATLAB, ROS, LabVIEW, API2 C/C++	No	37cm diámetro x 21cm altura	Paragolpes, Distancia, giroscopio, cámara, óptico, inductivo. /No	10
Turtletbot 2	Si	ROS.	No	31,5 x 43 x 34,7	Kinect/ Si.	50
Trutlebot 3 burger	Si	ROS.	Si	13,8 x 17,8 x 19,2	360° LIDAR/ Si	100
Turtlebot 3 waffle	Si	ROS.	Si	28,1 x 30,6 x 14,1	360° LIDAR, inercial de 9 ejes, Intel real sense. / /Si	95
Turtlebot 3 waffle pi	Si	ROS.	Si	28,1 x 30,6 x 14,1	360° LIDAR, inercial de 9 ejes, cámara. / Si	95

2.4 ¿QUÉ ES ROS?

ROS (Robot Operating System) es un framework de código abierto para el desarrollo de software especializado en robótica. Se compone de una serie de librerías, convenciones y herramientas para crear un entorno de trabajo especializado robusto. Cuenta con una variedad de características, el nombre sistema operativo (SO) se debe a que cuenta con funcionalidades tales como la abstracción

de hardware, paso de mensajes entre procesos, entre otros, similar a la de un SO. Sin embargo, un nombre más apropiado sería meta sistema operativo ya que ROS opera sobre un sistema operativo como Linux [16]. Una de sus partes más importantes es la infraestructura de comunicación. Ofrece una interfaz de traspaso de información en forma de *mensajes;* usualmente esta comunicación entre componentes es una de las primeras necesidades a la hora de crear cualquier robot.

Figura 12. Diagrama de ROS como meta sistema operativo. Tomado de: [16]

Su funcionamiento básico se explica muy bien como una arquitectura de grafos. De manera simplificada, ROS permite crear nodos de comunicación, que son archivos ejecutables con comandos de ROS dentro de un paquete de ROS. Estos nodos se comunican a través de tópicos. A su vez, un tópico puede *publicar* mensajes para que otro tópico se *subscriba* y viceversa. El nodo ROS Máster se iniciará con el primer nodo inicializado y será el encargado de la gestión de los demás nodos.

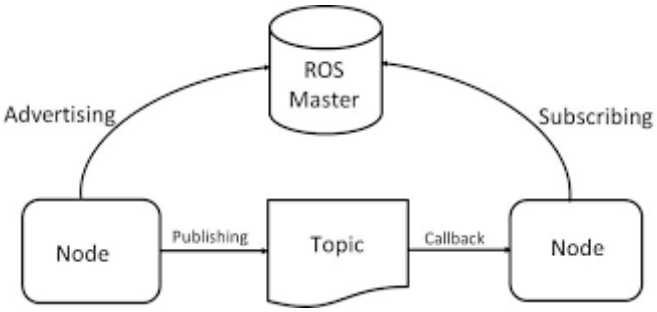

Figura 13. Grafo Computacional de un nodo de ROS. Tomado de www.theconstruct.com

Asimismo, ROS cuenta con un paquete llamado *gazebo_ros_pkgs* que permiten su integración con Gazebo, un simulador en gran medida fiel a la realidad puesto que incluye en su software motores de física como ODE [17], Bullet [18], Simbody [19] y DART [20]. Debido a esto el estado del ambiente simulado es altamente preciso y siempre está disponible, las covarianzas de las mediciones cuando

se simulan sensores se amplifican y el ruido gaussiano se agrega a los datos observados creando así escenarios realistas que permiten realizar simulaciones y tomar datos similares a los que se obtendrían en el amiente real [21].

2.5 MODELO CINEMÁTICO DEL ROBOT MÓVIL

Una vez comparados todos los robots móviles disponibles y teniendo en cuenta sus características, se optó por seleccionar el Turtlebot 3 modelo Burger, por su tamaño más pequeño comparado con el resto de la familia, lo que lo hace oportuno para navegar por el espacio del laboratorio de robótica. Además, cuenta con el sensor LIDAR que le permite obtener información de todas las direcciones de su entorno en cualquier posición y debido a que cuenta con la tarjeta Raspberry pi, es posible ampliar su gama de sensores en situación meritoria. Adicionalmente, es compatible cien por ciento con ROS y sus actuadores son servomotores con encoder de 12 bits de precisión.

A continuación, se presentan detalladamente las variables para tener en cuenta para obtener el modelo cinemático del robot:

Tabla 3. Dimensiones requeridas para el modelo.

R[cm]	33
L[cm]	160

Donde R y L se refieren al radio de las ruedas y la distancia entre estas respectivamente. Además, en la Figura 14 se exponen algunas características del robot que, si bien no son utilizadas para el modelo, son importantes para hacer un bosquejo de su geometría y aspecto físico.

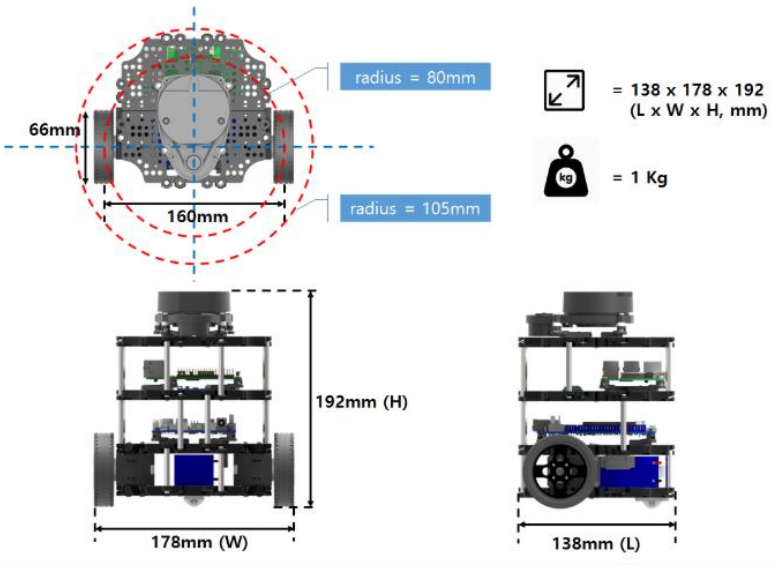

Figura 14. Turtlebot 3 modelo burger, tomado de: https://www.roscomponents.com/es/robots-moviles/214-turtlebot-3.html

Dado que se toma la superficie como un plano y teniendo en cuenta la configuración del robot (diferencial), las dimensiones de este para el modelo que nos interesan son el radio de las ruedas y la distancia entre estas (R y L). Las incógnitas que se buscan resolver son la orientación θ y la posición X, Y del móvil con respecto al marco de referencia global y estas variables a su vez con relación a las variables de entrada que serán las velocidades angulares V_I y V_D de las ruedas izquierda y derecha respectivamente. Gráficamente se observa de la siguiente manera:

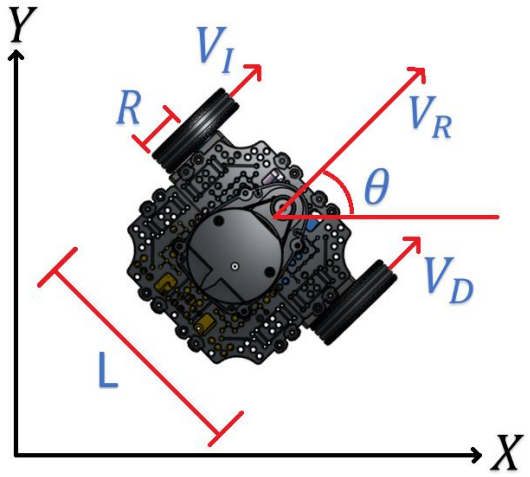

Figura 15. Variables del modelo cinemático del Turtlebot 3Burger. Elaboración propia.

Para lograr que el móvil se desplace en línea recta, es necesario que ambas ruedas tengan velocidades iguales, por tanto, la velocidad lineal V_R del móvil se define como el promedio de las velocidades lineales instantáneas de las ruedas, esto es:

$$V_R = \frac{R(V_D + V_I)}{2} \quad (1)$$

Para un movimiento sobre el centro de masa completamente angular, las velocidades de las ruedas deben ser de igual magnitud, pero de sentido opuesto. Por tanto, la velocidad angular del sistema se define como la diferencia de las velocidades lineales instantáneas de las ruedas, dividido sobre la distancia entre éstas, esto es:

$$w_R = \frac{R(V_D - V_I)}{L} \quad (2)$$

Este movimiento lineal se puede descomponer en movimientos en los ejes *X* y *Y*, además de poderse calcular las posiciones siguientes para instantes pequeños de tiempo t de la siguiente manera:

$$X' = X + V_R\, Cos(\theta) \quad (3)$$

$$Y' = Y + V_R\, Sin(\theta) \quad (4)$$

La nueva orientación está dada por:

$$w = \theta + w_R\, t \quad (5)$$

Estas ecuaciones se pueden representar en espacio de estados de la siguiente forma:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{w} \end{bmatrix} = \begin{bmatrix} Cos(\theta) & 0 \\ Sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} V_R \\ w_R \end{bmatrix} \quad (6)$$

Al reemplazar (2) y (3) en (6) se tiene:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{w} \end{bmatrix} = \begin{bmatrix} \frac{R\ Cos(\theta)}{2} & \frac{R\ Cos(\theta)}{2} \\ \frac{R\ Sin(\theta)}{2} & \frac{R\ Sin(\theta)}{2} \\ \frac{R}{L} & \frac{-R}{L} \end{bmatrix} \begin{bmatrix} w_D \\ w_I \end{bmatrix} \quad (7)$$

Donde w_D y w_I son las velocidades angulares de las llantas derecha e izquierda respectivamente. Por último, se reemplazan las dimensiones de la Tabla 3, donde se obtiene:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{w} \end{bmatrix} = \begin{bmatrix} \frac{33\ Cos(\theta)}{2} & \frac{33\ Cos(\theta)}{2} \\ \frac{33\ Sin(\theta)}{2} & \frac{33\ Sin(\theta)}{2} \\ \frac{33}{160} & \frac{-33}{160} \end{bmatrix} \begin{bmatrix} w_D \\ w_I \end{bmatrix} \quad (8)$$

A partir de las ecuaciones del modelo [22], se puede hacer una simulación de la cinemática directa del sistema con ayuda del software Simulink. En esta simulación, se tienen como entradas las velocidades de las ruedas en rad/s, y como salida las posiciones del robot con respecto al eje de referencia global.

En este sistema si ambas ruedas tienen la misma velocidad, se espera que el móvil se desplace en la dirección actual, como en la simulación el ángulo inicial es 0°, la dirección actual es completamente positiva en el eje x como se observa en la Figura 16.

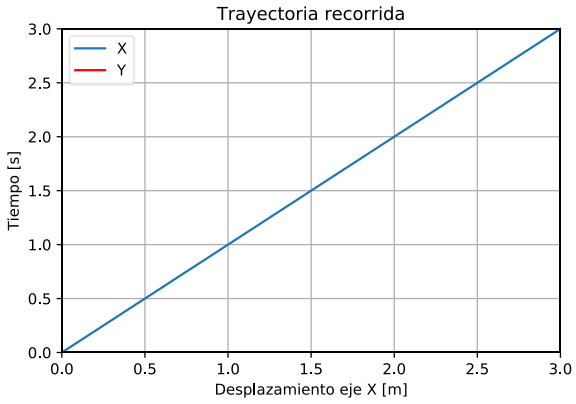

Figura 16. Comportamiento del sistema para velocidades de ruedas iguales. Elaboración propia.

2.6 ACONDICIONAMIENTO Y CONTROL

Para hacer una prueba de funcionamiento del Turtlebot, es necesario tener en cuenta el modo de funcionamiento ROS mencionado anteriormente. El tópico que se encarga de la velocidad del robot móvil es el llamado */cmd_vel* como se ve en la Figura 17 en donde además se muestran los tópicos necesarios para simular y controlar el robot.

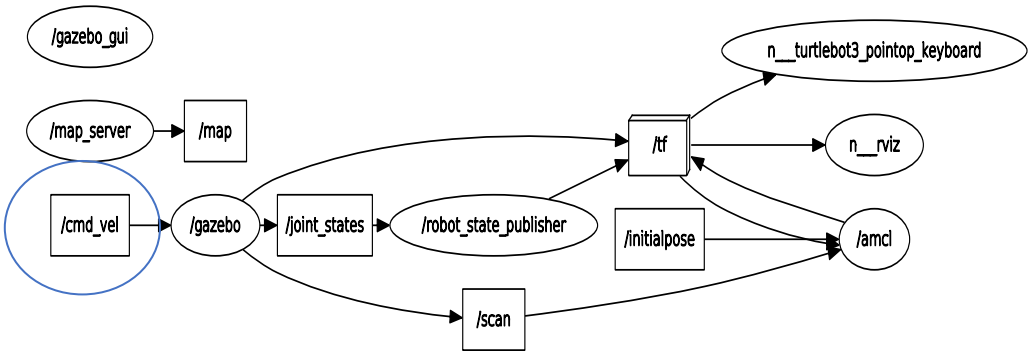

Figura 17. Estructura básica de tópicos para controlar un Turblebot 3. Elaboración propia.

El robot recibe órdenes de movimiento a través del tópico */cmd_vel* con mensajes tipo Twist que de acuerdo con la documentación [23] se compone de 2 vectores, uno de velocidad lineal y otro de velocidad angular, ambos en $\mathbb{R}^3$, por lo que tienen 3 componentes cartesianos cada uno. Estos componentes corresponden a los 3 ejes coordenados *x, y, z donde el* eje x es el del robot, es decir, está direccionado siempre con el móvil, los movimientos en los ejes *y* y z son ignorados por el robot, tanto en el vector de velocidad lineal como el angular por limitaciones holonómicas del sistema, quedando así solo dos ejes de velocidades útiles, uno que le indica que tan rápido se debe mover hacia adelante y otro que tan rápido debe girar. Una vez enviado este vector de velocidades, es el firmware de la tarjeta OPENCR1.0 quien se encarga de descomponer estas velocidades lineales en velocidades angulares de ruedas por medio de las ecuaciones de la cinemática inversa.

Como se observa en la Figura 18, el diagrama de bloques incluye un bloque extra el cual contiene la cinemática inversa del robot que convierte de velocidades lineales de la entrada a velocidades angulares de las ruedas, esto con el fin de simular la función de la tarjeta OPENCR1.0, es decir,

conectar la entrada que es una combinación de velocidades lineal y angular con el modelo de cinemática directa que recibe por entradas las velocidades de las ruedas.

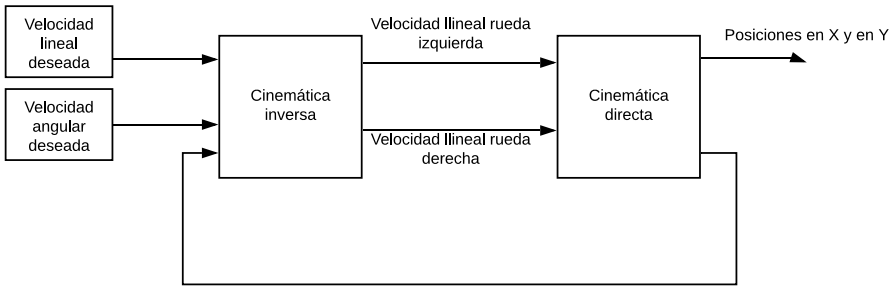

Figura 18. Diagrama cinemática inversa y directa Turtlebot 3. Elaboración propia.

Un ejemplo de movimiento simple sería una entrada de tipo escalón para la velocidad lineal, y una referencia en 0 para la velocidad angular, el móvil presentará el mismo comportamiento que el mostrado en la Figura 16. Comportamiento del sistema para velocidades de ruedas iguales. que corresponde a una línea recta. Con base en lo anterior, se realizaron pruebas de funcionamiento del robot móvil para su inspección y se ejecutaron de manera que, a partir de los resultados, se pueda definir si el robot necesita un control para la corrección de su movimiento o no.

Las pruebas se realizaron de la siguiente manera: se enviaron tres casos de órdenes de velocidades diferentes por intervalos de tiempo diferentes, esto es, 0.10m/s durante 20s, 0.05m/s durante 60s, 0.10m/s durante 40s y 0.10m/s durante 50s para recorrer 2, 3, 4 y 5m respectivamente. Para validar las pruebas se realizaron 5 intentos diferentes de los casos referidos y se obtuvieron los resultados mostrados en Tabla 4, Tabla 5, Tabla 6 y Tabla 7. Para cada prueba se registró el error de posición en los ejes [x], [y] midiendo la distancia real con cinta métrica y con la información proporcionada por los sensores.

Tabla 4. Pruebas para 2 metros V = 10.0 cm/s. Elaboración propia.

Ground Truth		Gold Standard (odom)	
error x[cm] ± 0.1	error y [cm] ± 0.1	error x [cm] ± 0.01	error y [cm] ± 0.01
7.6	1.5	3.46	5.44
14	-1.3	10.8	-2.24

	5.6	1.4	3.482	0.6
	6.9	1.6	3.33	15.86
	13.4	1.5	10.3	2.2
Media	7.6	1.5	3.482	2.2
Des. Estándar	3.493994848	1.121784293	3.494977745	6.257457631

Tabla 5. Pruebas para 3 metros V = 5.0 cm/s. Elaboración propia.

	Ground Truth		Gold standard (odom)	
	error x[cm] ± 0.1	error y [cm] ± 0.1	error x [cm] ± 0.01	error y [cm] ± 0.01
	3.5	3.4	0.9	1.6
	4	2.1	1.4	4.8
	4	1.6	1.4	9.6
	4.9	2.3	2	-9
	3.5	5.2	0.9	8.3
Media	4	2.3	1.4	4.8
Des. Estándar	0.511468474	1.282809417	0.406939799	6.645780616

Tabla 6. Pruebas para 4 metros V = 10.0cm/s. Elaboración propia.

	Ground Truth		Gold standard (odom)	
	error x[cm] ± 0.1	error y [cm] ± 0.1	error x [cm] ± 0.01	error y [cm] ± 0.01
	5.1	3.2	1.5	4.02
	4.3	10.4	3.4	12.94
	5.9	5.3	2.1	9.3
	4.5	3.2	0.9	2
	4.6	3.3	1.9	-0.56
Media	4.6	3.3	1.9	4.02
Des. Estándar	0.574108004	2.778056875	0.828492607	4.918325

Tabla 7. Pruebas para 5 metros V = 10.0cm/s. Elaboración propia.

	Ground Truth		Gold standard (odom)	
	error x[cm] ± 0.1	error y [cm] ± 0.1	error x [cm] ± 0.01	error y [cm] ± 0.01
	3.3	2.1	1.3	9.2
	5.9	4.6		
	6.5	3.2	2.5	3
	6.4	1.1	1.4	-0.5
	5.9	3.1	1.2	7.2
Media	5.9	3.1	1.35	5.1
Des. Estándar	1.176435294	1.172006826	0.524404424	3.755912

Como se puede observar en las pruebas previamente realizadas, en cada una de ellas está presente un error de posición en ambos ejes. Nótese que el error en el eje x oscila entre los mismos valores independientemente del tiempo y de la velocidad, es un error visualmente notable y en el peor de los casos hay un desfase de referencia del 6.7%. En cuanto al error de posición del eje y, además del error causado por desfase de velocidades de las ruedas está el desfase en cuanto al posicionamiento del robot al momento de empezar a hacer la prueba, esto se puede evidenciar en la Figura 19 donde se observa que teóricamente los ejes [y] de la prueba y los ejes [y] del móvil deben ser paralelos, pero por error humano esto no es posible y este error debe ser tenido en cuenta. De igual manera se tiene cuenta que estos errores son indicados también por la odometría del robot lo cual es un indicio de su precisión y sus datos son confiables para la ubicación del robot.

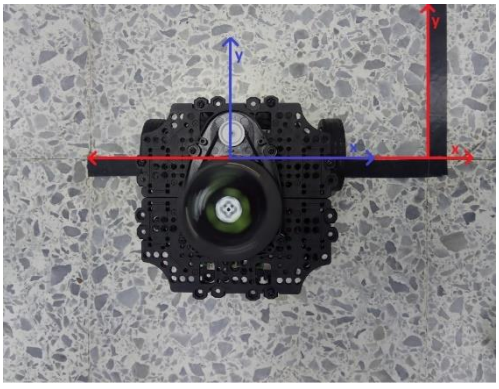

Figura 19. Posición inicial para prueba de funcionamiento turtlebot 3 burger. Elaboración propia.

Es importante que estos errores se traten debido a que serán acumulativos para cada punto de referencia al cual se le indique al robot que llegue. Con un control de bajo nivel o aplicado a los motores para ratificar su velocidad no se asegura que el móvil llegue a la referencia, pues hay otros factores como posible rodadura causada por el estado del suelo, como se puede observar en la Figura 20b las ruedas del móvil quedan con una capa de polvo lo cual facilita los deslizamientos. Para asegurarse de que la velocidad del móvil sí es la que corresponde, se hizo una medición con ayuda de un tacómetro, Figura 20a, donde se envió una referencia de velocidad de 0.1m/s, lo cual corresponde a 0.09921m/s para un radio de 0.033m, esto es menos del 1% lo que es de esperarse ya que los actuadores son servomotores de alto rendimiento [24]. En atención a esto, se procede a hacer un control de un nivel más alto, es decir, un control de posición que asegure que el robot llegue a los puntos indicados, esto implica que el seguimiento de las trayectorias indicado por el método de planeación de trayectorias deberá ser por referencias de posición.

Figura 20. a) Medición de velocidades de ruedas con tacómetro. b) Estado de las ruedas luego de las pruebas de funcionamiento. Elaboración propia.

Como se expuso anteriormente, las órdenes de movimiento del robot Turtlebot 3 Burger deben estar compuestas por una velocidad lineal y otra angular. Con el sistema de odometría del robot del cual se evidenció su confiabilidad, se pueden usar sus datos de posición y orientación para ser realimentados y formar un sistema de control añadiendo una constante de proporcionalidad k_p como se muestra en la Figura 21 y de esta forma mejorar los resultados en cuanto a posicionamiento del robot en comparación con la prueba anterior. En la Figura 22 se evidencia la trayectoria trazada por el móvil para una referencia de 0.6m y 0.3m en los ejes x, y respectivamente.

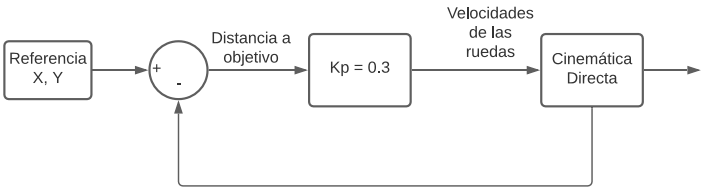

Figura 21. Diagrama de bloques de control PID de posición. ***Elaboración propia.***

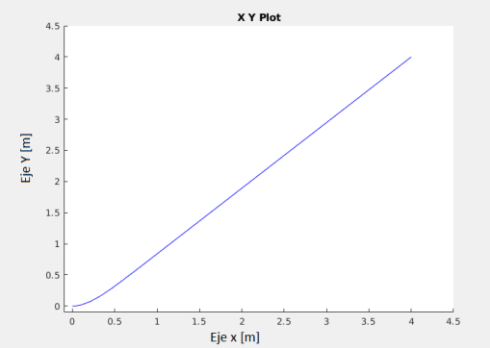

Figura 22. Trayectoria recorrida con control de posición. Elaboración propia.

Este control de velocidad se implementó en el Turtlebot 3 burger obteniendo la trayectoria vista en la Figura 22, se propuso una condición de parada en un error equivalente al 10% de la medida de separación de las ruedas que corresponde a 1.6cm, error que es aceptable ya que es imperceptible y fácilmente de lograr con el control de posición implementado por la precisión de los actuadores del robot [25]. Nótese que los resultados fueron muy similares en simulación (Figura 16) y en la implementación (Figura 22) y cumple con el error propuesto como se ve en la Figura 23, por lo que este control supera el problema del posicionamiento encontrado en las pruebas de funcionamiento anteriores.

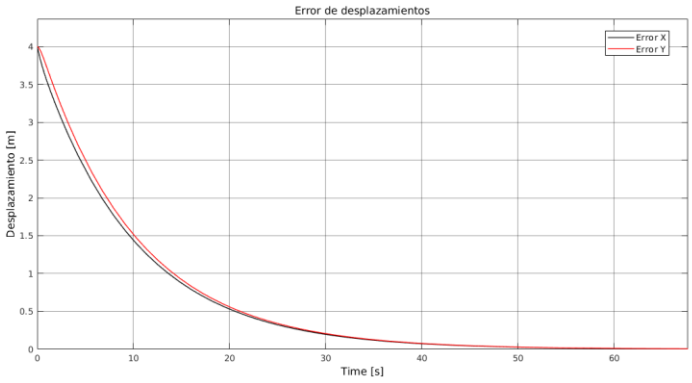

Figura 23. Errores de posición control PID. ***Elaboración propia.***

3 PLANEACIÓN DE TRAYECTORIAS

3.1 SELECCIÓN DEL MÉTODO

Una vez implementado el control de posición del robot móvil, el siguiente paso es la selección de un método de planeación de trayectorias para lo cual fue necesaria una revisión de los métodos existentes. Debido a que esta es un área aún en investigación, existe una cantidad considerable de métodos y algunos de estos tienen variaciones que trabajan sobre alguna de las falencias de sus métodos originales [26]. Antes de entrar en los métodos, se empieza por definir el problema teniendo en cuenta las consideraciones tomadas al momento de hacer el modelo cinemático del robot móvil.

- El problema es el siguiente:

Dada una posición y una orientación del robot móvil en su espacio de trabajo, generar un camino especificando una secuencia continua de posibles configuraciones de posiciones y orientaciones del móvil evadiendo contacto con cualquier tipo de obstáculo, empezando en un punto dado de inicio y terminando en un punto dado final y reportar falla en caso de que tal camino no exista [27].

Cada método que existe se enfoca en resolver este problema desde distintos enfoques, pero se pueden agrupar o clasificar dentro de los 4 niveles propuestos en la Figura 24. En el primer nivel se encuentran las restricciones propias del robot. Un sistema es holonómico cuando es posible controlar todos sus grados de libertad, por el contrario, un sistema no-holonómico es cuando un grado de libertad está presente pero no es posible su control. Un ejemplo de sistema no-holonómico es un automóvil convencional debido al problema del estacionamiento paralelo y, un ejemplo de un algoritmo que tenga en cuenta esta restricción es *Dubins optimal path* [28]. En el nivel 2 se clasifican los métodos dependiendo de su necesidad de un modelo del espacio de trabajo, un ejemplo de un método que necesite modelo del entorno es el conocido A* (A asterisco) que necesita un modelo previo del entorno para encontrar un camino factible, en contraste existe el método de los campos potenciales [29] que no necesita modelo del entorno.

En el tercer nivel los métodos se clasifican dependiendo de su capacidad de re-planeamiento. En general, los algoritmos que entran en la categoría on-line son aquellos que no requieren modelo previo

del entorno, tal como campos potenciales, sin embargo, no siempre es el caso. Por ejemplo, el algoritmo D* [30] que es una variación de A* requiere modelo del entorno igualmente, pero está optimizado para permitir un procesamiento on-line. Por último, se clasifican los métodos en determinísticos y probabilísticos, un ejemplo de método determinístico es Dijkstra [27] y se caracterizan por encontrar la misma solución cada vez que son ejecutados con las mismas condiciones, en contraste con los probabilísticos tales como RRT [31] los cuales al tener un componente aleatorio en su cómputo, sus resultados cambian con cada ejecución.

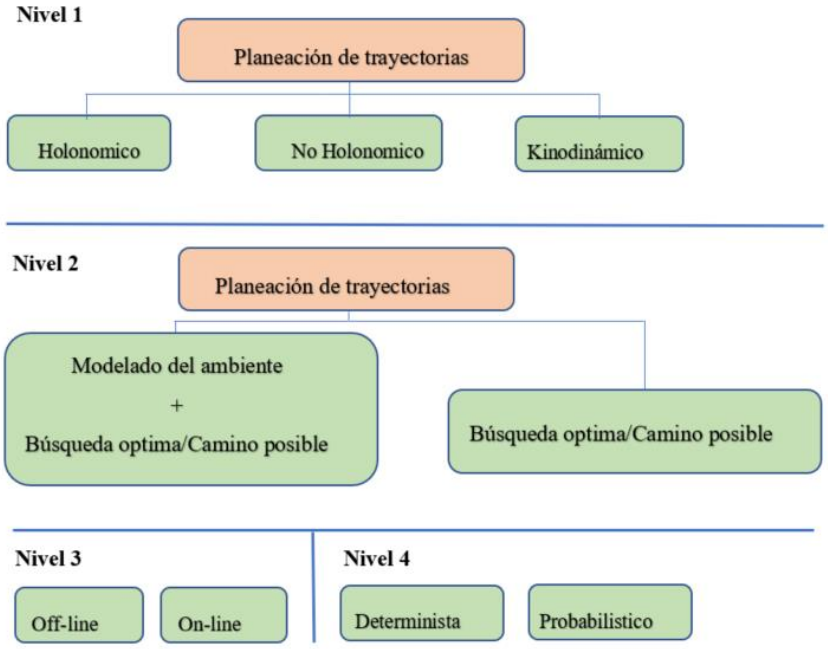

Figura 24. Clasificación de métodos de planeación de trayectorias por niveles [3]. Elaboración propia.

Para la selección del método de planeación de trayectorias se tuvo en cuenta las anteriores clasificaciones pues estas se utilizaron para compararlos entre sí, aludiendo nuevamente al objetivo general del proyecto el cual es integrar un algoritmo de evasión de obstáculos a uno de planeación de trayectorias para la evasión de obstáculos, el método de trayectorias inicialmente puede ser cualquiera ya que al momento de que el móvil se encuentre con un obstáculo no identificado, es trabajo del algoritmo de evasión de obstáculos tomar el control del robot hasta que este sea rebasado, sin embargo, es importante evaluar los métodos para elegir uno.

En el primer nivel de clasificación, se requiere que el método tenga en cuenta que el móvil es no-holonómico y sus indicaciones sean acordes a esto. En el nivel dos, el modelo del ambiente sería una ventaja debido a que esto evitaría estancamientos que es un objetivo clave en el desplazamiento. En cuanto al nivel 3, cualquier clasificación es permitida, lo único indispensable es un sistema de interrupción para conmutar entre algoritmos.

Por último, en el nivel 4 sería notable que no sea probabilístico debido a que ahorraría tiempo de ejecución al asegurar un camino óptimo y al ser siempre el mismo sería una ventaja al momento de la toma de datos. Por lo anterior se realiza una tabla comparativa con algunos de los algoritmos más utilizados con sus clasificaciones acompañadas de pesos para su selección, estos pesos se asignaron de la siguiente manera:

- Al nivel 1 se le asigna un peso de 20 debido a que es preferible que el algoritmo tenga en cuenta que el móvil es no-holonomico
- El nivel 2 tiene el peso más alto, 40 puntos debido a la importancia de la existencia del modelo y que este sea óptimo.
- El nivel 3 tiene un peso de 0 en esta selección debido a que es indiferente si el algoritmo tiene la capacidad de recalcular la trayectoria cuando encuentre obstáculos debido a que esta tarea de evasión es el propósito de la siguiente parte del algoritmo general que se desarrolla.
- El nivel 4 tiene un peso de 20 en virtud de que al ser determinista se facilita la comparación al momento de realizar las pruebas de funcionamiento. Por último, se tiene en cuenta el tiempo estimado para la implementación de cada uno de los algoritmos propuestos, este tiempo tendrá un peso de 20 debido a que es importante seguir con el cronograma propuesto.

Tabla 8. Comparación algoritmos de planeación de trayectorias. Elaboración propia.

Clasificación / Algoritmo	Nivel 1 [20]	Nivel 2 [40]	Nivel 3 [0]	Nivel 4 [20]	Tiempo esperado de realizar [20]	Puntaje total
A*	No-holonomico	Modelo de entorno + búsqueda camino optimo/camino factible.	Off - line	Determinista	1 semana	100
RRT	No-holonomico	Modelo de entorno + búsqueda camino	Off - line	Probabilístico	1 semana	40

		optimo/camino factible.				
Campos potenciales	No-holonomico	Búsqueda camino optimo/camino factible	On-line	Determinista	2 semanas	40

Teniendo en cuenta la tabla anterior, el algoritmo que se escogió fue el A*, se tuvo en cuenta el tiempo esperado a realizar cada uno de estos de acuerdo con su nivel de complejidad según la bibliografía leída [31] y se optó por calificar positivamente (color azul) los más sencillos, esto considerando que es un paso clave en el proyecto mas no es el punto de investigación de este.

3.2 ALGORITMO A*

A*, desarrollado en 1968, es un algoritmo de búsqueda. Es una extensión del algoritmo Dijkstra en el cual se disminuye la exploración por medio de un costo añadido de todo el camino, la heurística. Debido a su naturaleza, entregará el camino más corto dentro del modelo dado [3]. Para la explicación de este método se supondrá un modelo en el que se divide el ambiente en cuadrículas. La forma como funciona se muestra en el Algoritmo 1, se empieza en una cuadrícula o nodo inicial la cual se expande a sus vecinos dándole a cada uno un peso que se llamará f_n. En la Figura 25a se muestra un ejemplo donde en el primer instante el nodo inicial se muestra en azul y está denotado con la letra A. Los nodos a su alrededor se le llaman vecinos, a cada uno se le asigna un peso f_n expuesto dentro del cuadro amarillo (Figura 25a). Inicialmente todos los vecinos son candidatos posibles por eso se marcan de color verde y este conjunto se llama *Openset*. El peso f_n de cada nodo se define entonces:

$$f_n = g_n + h_n \qquad (9)$$

Donde g_n es el costo de ir del nodo actual a un nodo vecino y se muestra en la figura con un círculo de color café; h_n es la heurística que es la distancia de cada nodo al punto final y se muestra en la figura con un círculo de color morado. De todos estos nodos vecinos de *A*, se selecciona el que menor peso tenga, se saca del conjunto *Openset* y ahora se ingresa en el conjunto *Colsedset* y se dibuja de color rojo en la Figura 25b. Este proceso se repite hasta llegar al punto final, llegando al camino final mostrado en la Figura 26 y este camino final se computa como la secuencia de nodos subsecuentes dentro del conjunto *Closedset* que tenga la menor sumatoria de pesos f_n.

Algoritmo 1: A* [3], Elaboración propia		
1	:	Inicializar *Openset.*
2	:	Inicializar Closedset.
3	:	Elegir punto inicial.
4	:	Elegir punto final.
5	:	Añadir el punto inicial a *Openset.*
6	:	**Mientras** (no *se ha alcanzado el nodo de final):*
7	:	*Calcular nodo con menos fn.*
8	:	**Si** (es el punto final): terminar procedimiento
9	:	Añadir nodo con menor fn a *Closedset.*
10	:	Sacar nodo con menor fn de Openset.
11	:	**para** (cada vecino del nodo actual):
12	:	**sí** (el vecino tiene un valor de ng más bajo que el actual **y** está en *Closedset*):
13	:	Reemplace el vecino con el nuevo valor gn más bajo.
14	:	El nodo actual es ahora el padre del vecino.
15	:	**Si no, si** (el valor de g actual es menor **y** este vecino está en Openset):
16	:	Reemplace el vecino con el nuevo valor g más bajo.
17	:	Cambiar el padre del vecino a nuestro nodo actual.
18	:	**Si no, si** (este vecino no está en ambas listas):
19	:	agréguelo a la lista abierta y establezca su gn.

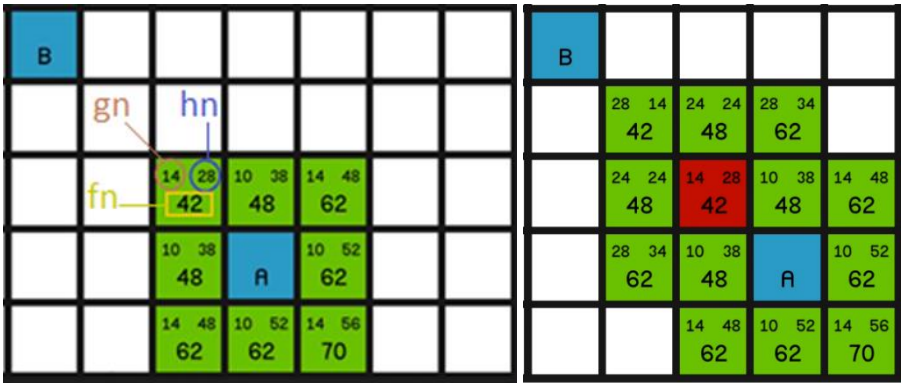

Figura 25. a) Mapa de coste algoritmo A paso 1. b) Mapa de coste algoritmo A* paso 2. Elaboración propia.*

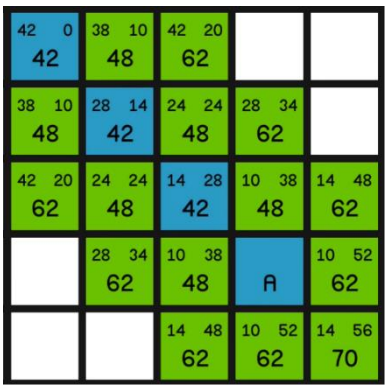

Figura 26. Mapa de coste algoritmo A paso final. Elaboración propia.*

3.3 PRUEBAS Y CORRECCIONES DEL MÉTODO SELECCIONADO.

Para las pruebas del algoritmo de planeación de trayectorias A* es necesario un modelo previo del entorno. Se empezó por la creación de un ambiente virtual similar al laboratorio de robótica de la Universidad Militar Nueva Granada considerando sus mediciones, sin tener en cuenta las mesas, sillas ni demás elementos y quitando el espacio de trabajo del laboratorista. Con el robot en una posición arbitraria, el modelo del espacio se muestra en la Figura 27. Esta simulación se realizó con ayuda del software Gazebo [32] que se conecta con ROS.

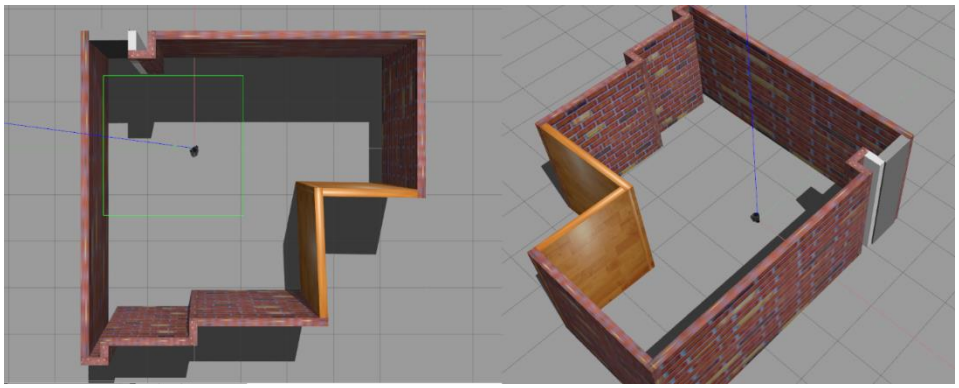

Figura 27. Vista superior (Izquierda) y vista general (derecha) del ambiente virtual. Elaboración propia.

Una vez se tiene el entorno virtual, se prosigue haciendo representación cuadriculada de éste en forma de celdas regulares aproximadas [27] como se muestra en la Figura 28, celdas que serán utilizadas como nodos por el algoritmo A*. Las celdas se eligieron cuadradas para mantener el modelo simple y dada la geometría del Turtlebot 3 Burger dichas celdas se establecieron del tamaño de la parte más ancha del robot, esto es 0.178m o 17.8cm cubrir la mayor cantidad de posiciones posible y el robot

se moverá por los puntos medios de estas celdas considerando el error tolerado por el control implementado. Para términos ilustrativos se agregaron diferentes colores donde los cuadrados de color negro representan el espacio libre y los de color amarillo representan obstáculos o espacio no disponible para la navegación del móvil.

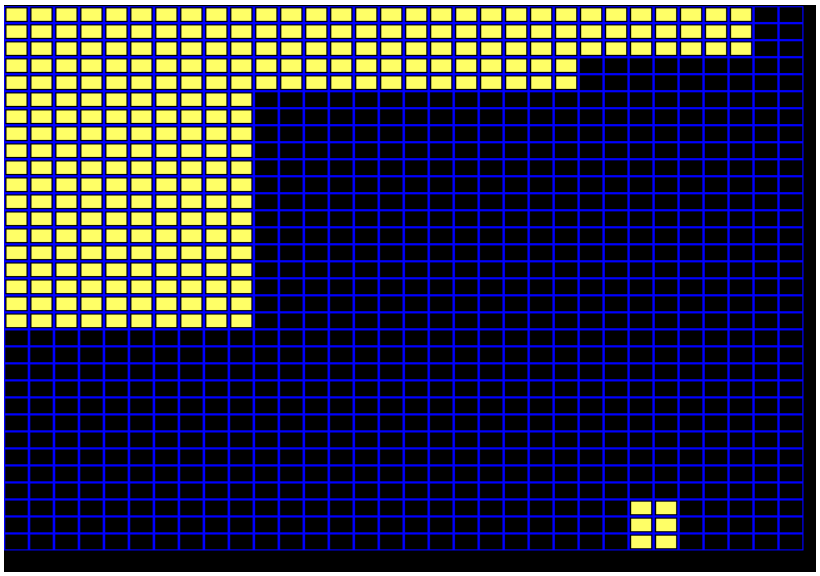

Figura 28. Modelo espacio de trabajo. Elaboración propia.

Teniendo el entorno virtual de trabajo y su modelo, se procede a la realización de pruebas. Éstas fueron de 3 tipos: La primera prueba consiste en ir de un punto inicial a un punto final cualquiera. Teniendo en cuenta la forma en que se modeló el espacio de trabajo, los puntos a los cuales se le puede indicar llegar al robot deben ser en forma de una coordenada que indique el centro de cada uno de los cuadros del espacio libre. Para esta primera prueba el punto inicial se encuentra en (21, 20) y el punto final se indicó en (2, 28). La trayectoria que el algoritmo calculó y el desplazamiento se muestra en la Figura 29, los puntos azules denotan el camino, mientras que los puntos verdes indican los posibles puntos candidatos a camino *Openset*.

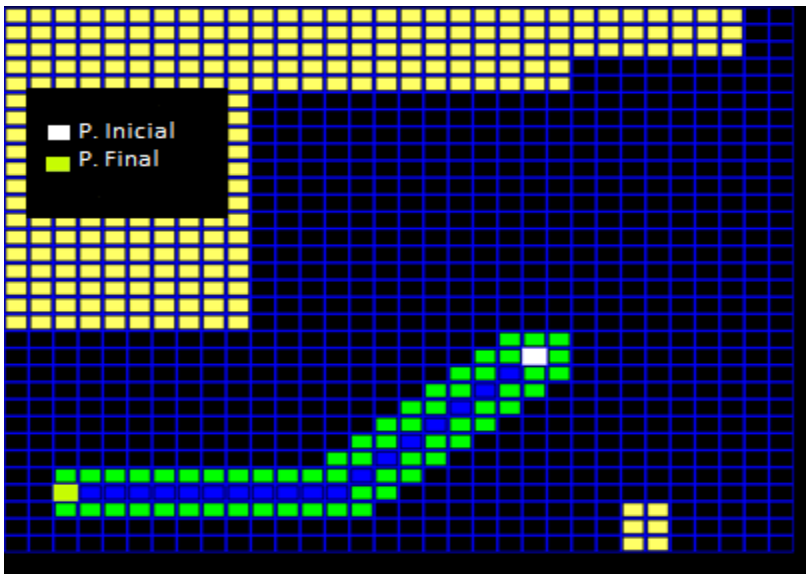

Figura 29. Trayectoria calculada con A, prueba 1. Elaboración propia.*

Dado que el algoritmo A* entrega el camino en forma de coordenadas, basta con enviarlas como referencias de posición y con ayuda del control implementado en la sección 2.6 el móvil las seguirá. Las posiciones seguidas en cada momento de la prueba se muestran en la Figura 30.

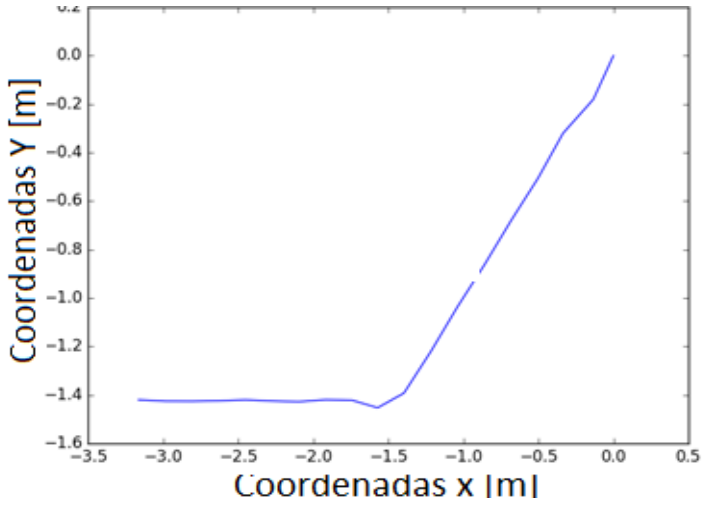

Figura 30. Coordenadas del robot durante la primera prueba de A. Elaboración propia.*

Para la segunda prueba se añade un obstáculo simple en forma de pared, conservando los puntos iniciales y finales. La trayectoria que calculó el algoritmo se muestra en la Figura 31. Nótese que

aparecen unos puntos rojos, estos son los candidatos que el algoritmo revisó, pero rechazó como puntos de trayectoria (*Closedset)*. Su gráfica de desplazamiento se muestra en la Figura 32.

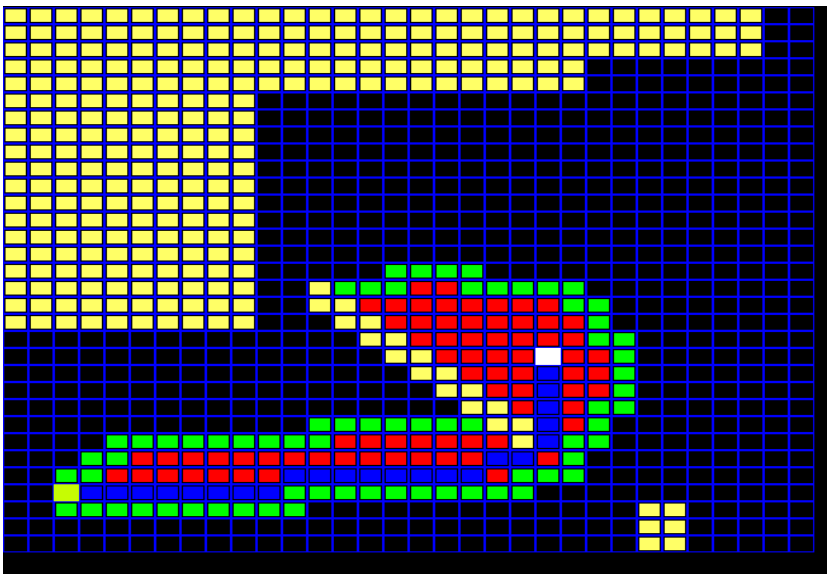

Figura 31. Trayectoria calculada por A, prueba 2. Elaboración propia.*

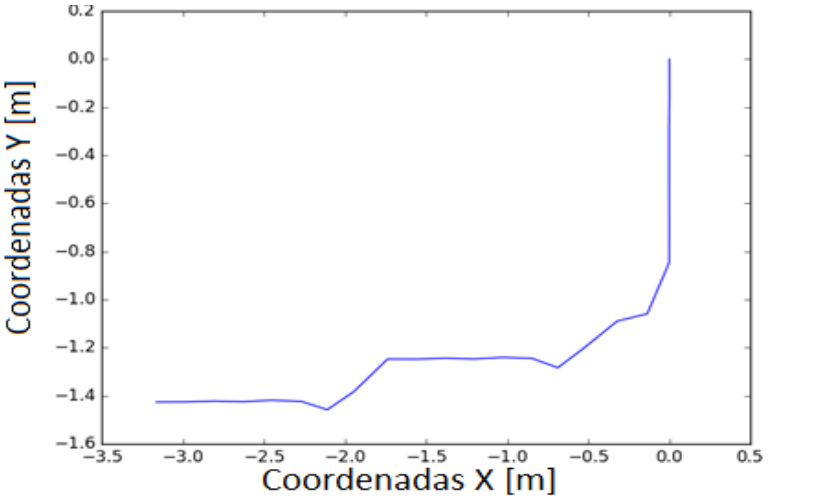

Figura 32. Coordenadas del robot durante la segunda prueba de A. Elaboración propia.*

Para la última prueba se añaden más obstáculos y se prueban varios puntos iniciales y finales, éstos son (28, 28), (2, 28), (13, 8), (26, 10) en orden consecutivo. Esta vez se le indicó al algoritmo que una vez calculara un camino, se dirigiese al siguiente punto y el resultado se muestra en la Figura 33 y Figura 34.

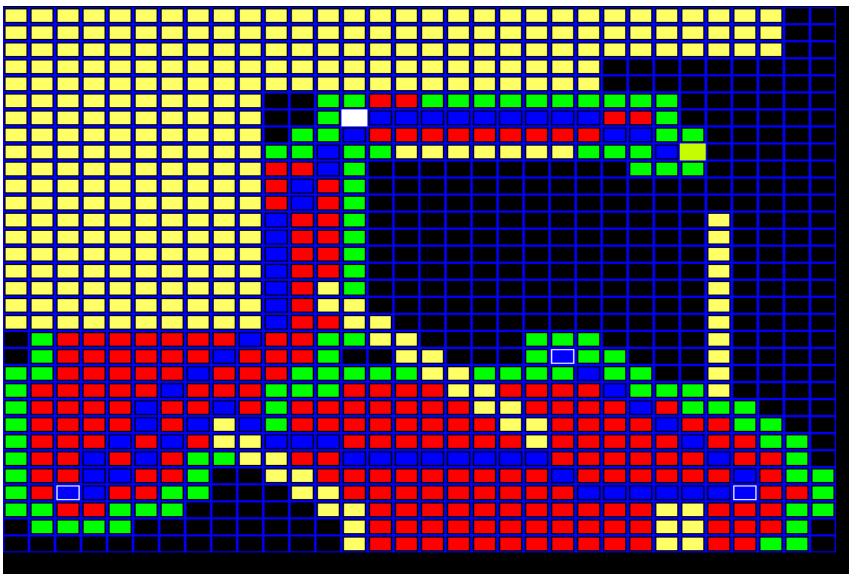

Figura 33. Trayectoria calculada por A, prueba 3. Elaboración propia.*

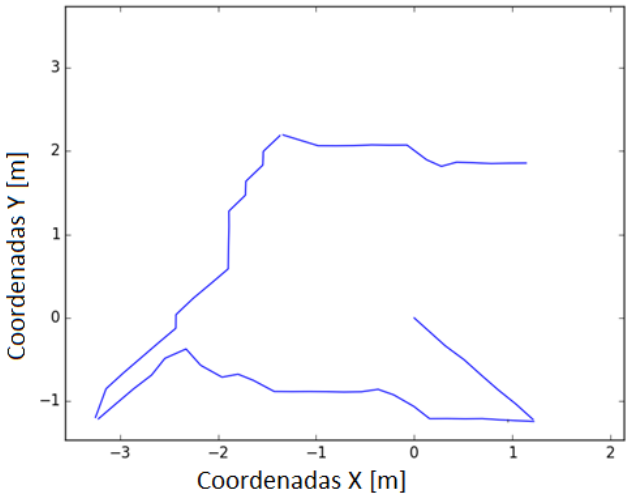

Figura 34. Coordenadas durante la tercera prueba de A. Elaboración propia.*

4 EVASIÓN DE OBSTÁCULOS DINÁMICOS

4.1 INTELIGENCIA ARTIFICIAL, EL APRENDIZAJE AUTOMÁTICO Y LOS ALGORITMOS GENÉTICOS

En esta sección se informará sobre los métodos tenidos en cuenta para implementar la tarea de la evasión de los obstáculos. Dado que la mayoría de estos métodos involucran inteligencia artificial es importante definir los términos relacionados que serán utilizados más adelante.

La inteligencia artificial (IA) es un campo de la informática que se centra en la creación de algoritmos que lleven a comportamientos de las máquinas para que éstas se consideren como inteligentes. Para no entrar en la definición del término de inteligencia, en este trabajo se define que la IA es el procedimiento con el cual se logra que una máquina se comporte como un humano. Con los avances en cuanto a la capacidad de cómputo y más con la llegada de las tarjetas gráficas GPUs, con su procesamiento en paralelo, se han logrado grandes avances en este campo. Uno de los más importantes es el del aprendizaje automático o *Machine Learning* el cual a su vez se ramifica en varios tipos, como el aprendizaje supervisado o el aprendizaje por refuerzo, este campo se puede representar gráficamente por medio de la Figura 35, además de ilustrar cada tipo con ejemplos de sus aplicaciones [33].

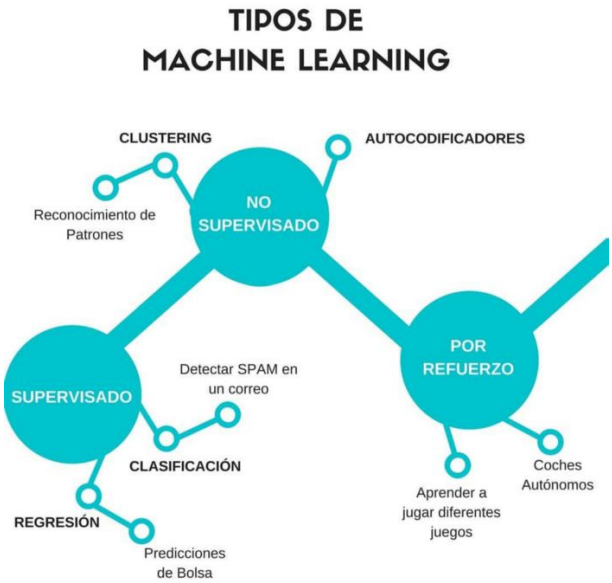

Figura 35. Tipos de aprendizaje de máquina. Tomado de: https://medium.com/

En este punto del aprendizaje automático es donde entran las redes neuronales artificiales que, aunque no son exclusivas del aprendizaje automático si son una técnica bastante utilizada en este tipo de aprendizaje. Las redes neuronales artificiales se consideran aproximadores de funciones y son modelos computacionales inspirados en la forma en la que trabaja nuestro cerebro, una gran cantidad de neuronas que se conectan y transmiten información entre ellas forman lo que se conoce como redes neuronales. Se puede decir entonces que la unidad de procesamiento en una red neuronal artificial es la neurona artificial, esta recibe unas entradas asociadas con unos pesos con los cuales realizará una serie de operaciones para entregar una salida. Para formar una red neuronal artificial, estas neuronas se organizan formando lo que se conoce como capas. En la Figura 36 se muestra la representación de una red neuronal con una capa de entrada (color rojo), una capa intermedia (color azul) y una capa de salida (color verde), una red neuronal artificial puede tener varias capas intermedias que también se conocen con el nombre de capas ocultas. Matemáticamente una neurona se representa como un sumador lineal. [33]

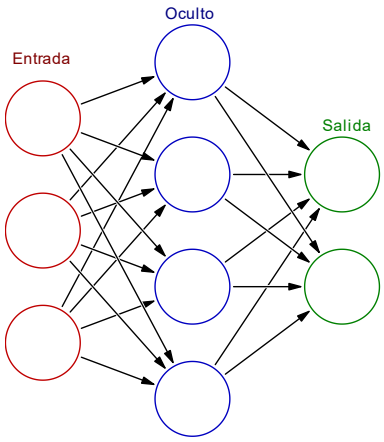

Figura 36. Representación de una red neuronal. Tomado de: www.wikipedia.org/wiki/Red_neuronal_artificial

Las redes neuronales artificiales requieren de un proceso de entrenamiento encargado de ajustar los pesos asociados a las neuronas y así cumplir con la tarea deseada, durante este proceso en el aprendizaje supervisado es necesario suministrarle a la red las salidas correctas para cada entrada. De esta forma, se puede hacer un cálculo de errores el cual se busca minimizar. Una vez una red neuronal está entrenada funcionará como un excelente aproximador tanto para las entradas con las cuales se hizo el entrenamiento como para entradas que sean relacionadas llevando a que las redes neuronales artificiales obtengan resultados sorprendentes como el de clasificar imágenes de perros y gatos, tipos de flores, etc.

Paralelamente, existen otro tipo de algoritmos que conllevan a las máquinas a tener comportamientos inteligentes; esta vez desde un enfoque de optimización se hace referencia a los algoritmos genéticos, inspirados en la teoría de la evolución de Darwin. Estos estos algoritmos permiten la evolución de una población de individuos por medio de condiciones similares para que por medio de estímulos aleatorios estos individuos tengan la oportunidad de evaluarse entre ellos y los más adaptados sobreviven. Este procedimiento se evidencia con más detalles en el Algoritmo 2.

Algoritmo 2: Algoritmo genético. Tomado de: [34]

1	:	**Inicialización** aleatoria de la población
2	:	**Aplicación** de estimulo
3	:	**Evaluación** de cada uno de los individuos con la función idoneidad.
4	:	Aplicar **Selección,** Aplicar **Cruzamiento,** Aplicar **Mutación.**
5	:	**Condición de termino**

Nótese que en el Algoritmo 2 aparecen términos de biología como cruzamiento y mutación, estos términos son llamados así a propósito, debido a que estas son las operaciones genéticas que se aplican para la adaptación de los individuos. El tamaño inicial de la población dependerá de la naturaleza del problema. El estímulo aplicado a cada individuo no necesariamente debe ser el mismo, la evaluación de cada individuo se realiza por medio de una función de idoneidad que tendrá por salida un valor numérico que representa el desempeño del individuo, normalmente se busca que la mayor cantidad de individuos posibles tengan mejores idoneidades. Una vez calculada la idoneidad de cada uno de los individuos se hace una selección de los mejores que serán los que pasarán a la siguiente generación; este proceso también se conoce como elitismo ya que los mejores individuos son seleccionados sin hacerles transformaciones extras, estas transformaciones son la cruza entre individuos y la mutación. La primera consiste en tomar dos individuos que serán los *padres* y tomar características de ambos para formar uno o varios individuos diferentes. Por último, se realiza la mutación, proceso mediante el cual se hacen pequeños cambios sobre los individuos creados por la cruza, esta mutación será aleatoria [35].

Suponga dos individuos de una población que se somete a una evolución por algoritmos genéticos, un ejemplo de un individuo creado por medio de la cruza entre dos individuos padres puede ser el mostrado en la Figura 37. Nótese que los dos padres heredan una parte de sí mismos, por lo que las

partes restantes pueden ser utilizadas para crear nuevos individuos y de igual manera se pueden tener más aún si la partición no se hace a dos partes sino a más partes dependiendo del tamaño de los padres.

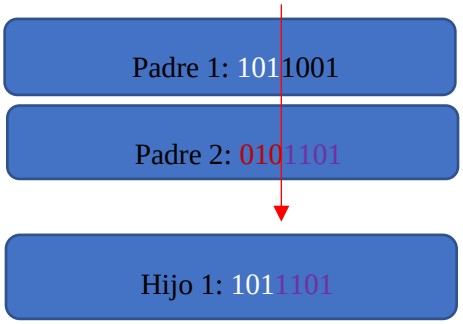

Figura 37. Proceso de cruza entre dos individuos. Elaboración propia.

Este nuevo individuo creado tendrá el mismo tamaño de los padres y debido a que nació a partir de los mismos, se parecerá a ambos tal y como sucede en la biología. El proceso de mutación sobre este individuo creado se puede hacer cambiando uno de sus dígitos, debido a que la representación es binaria es fácil imaginar este proceso y se evidencia en la Figura 38. En este caso la mutación solo hace cambio de un dígito lo que quiere decir que el cambio no es tan drástico.

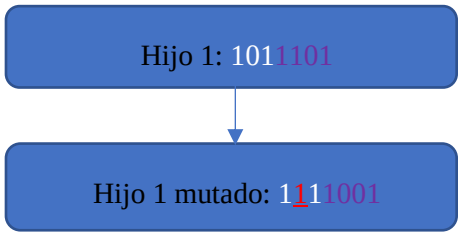

Figura 38. Proceso de mutación. Elaboración propia.

Al final de cada ciclo, la población debe ser de igual tamaño que la inicial por lo que el número de individuos seleccionados por elitismo y generados por cruza deben ser fracciones de la población que sumen su total en partes que pueden ser parámetros cambiados por el usuario [34].

4.2 ALGORITMOS DE INTELIGENCIA ARTIFICIAL APLICADOS EN LA EVASIÓN DE OBSTÁCULOS DINÁMICOS

Por lo que refiere a la evasión de obstáculos existen diversos métodos para una revisión del estado del arte en esta sección se exponen mayoritariamente aquellos trabajos donde se utilice inteligencia artificial debido a que se está tratando con un problema de conducción autónoma y es en la inteligencia artificial donde se han hecho los más grandes avances [36]. Las estadísticas muestran que el 94% de los choques en automóviles son causados por errores del conductor y de este un 10% son causados debido a que el conductor está bajo los efectos de alguna sustancia alucinógena o simplemente distraídos [37]. La conducción autónoma entonces tiene el potencial de reducir estos casos de accidentes, además de las aplicaciones adicionales que esto conlleve con la utilización del tiempo de conducción se tiene más tiempo productivo por parte de los conductores. Para lograr esta tarea, los móviles en cuestión son equipados con sistemas de control robustos que permiten una maniobrabilidad precisa y a esto se añaden algoritmos de planeación de trayectorias las cuales serán seguidas mientras sea posible, pero debido a lo fortuito del mundo real siempre se presentarán situaciones impredecibles, pero sobre las cuales es necesario tomar una decisión y, es aquí donde la inteligencia artificial tiene una ventaja sobre cualquier otro algoritmo imperativo.

Los estimadores a pesar de no ser inteligencia artificial, son una tecnología comúnmente usada para la tarea de la evasión de obstáculos, un ejemplo se ve en [25], donde se utiliza un filtro de Kalman para estimar la poción siguiente de un obstáculo en movimiento y estas dos posiciones, la actual y la predicha, se toman como dos obstáculos estáticos diferentes. Con estos dos se hace un recálculo de la trayectoria y así evadiendo el obstáculo como se muestra en la Figura 39, .

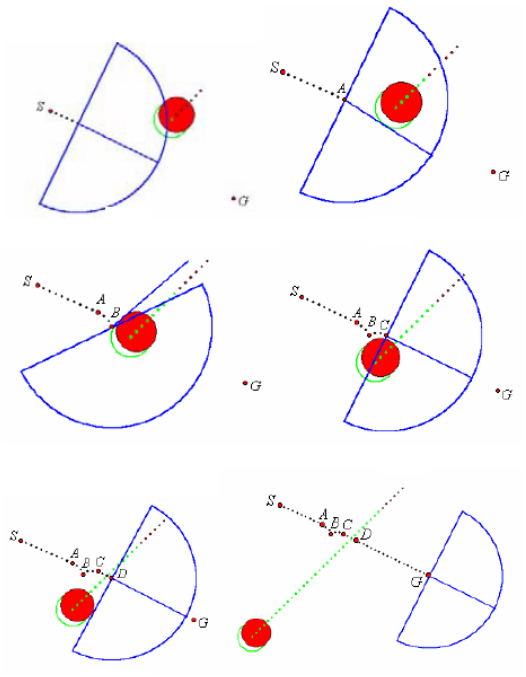

Figura 39. Algoritmo de evasión de obstáculo con filtro predictivo. Tomado de [25].

Paralelamente se ha utilizado frecuentemente lógica difusa, un ejemplo se observa en [38] donde por medio de esta tecnología se consigue un controlador para los actuadores de las ruedas de un robot diferencial, utilizando como entradas de este la distancia al obstáculo, la orientación del obstáculo (ángulo formado por el robot y el obstáculo) y la orientación del punto objetivo (ángulo formado por el robot y el punto donde se desea ir) y las salidas son la velocidad lineal del robot y su velocidad angular.

Un enfoque diferente es el aprendizaje por refuerzo utilizado en [4], el cual es un método de aprendizaje automático que consiste en situarse en un contexto de agente-ambiente, donde el agente puede tomar acciones que estimulen al ambiente para que este le devuelva un estado siguiente y una recompensa, siendo como meta final lograr el mayor acumulado de esta última. Este ciclo se puede representar por medio del diagrama mostrado en la Figura 40.

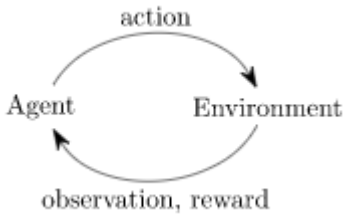

Figura 40. Lazo agente ambiente. Tomado de: [39]

Al agente no se le dice nunca qué acciones tomar, éstas deben ser descubiertas por medio de su exploración y descubrir qué acciones maximizan esta recompensa. En específico, la técnica para implementar este tipo de aprendizaje en [5] es Q-Learning, en el cual se discretizan estos estados del robot obteniendo una tabla *Qs* que representa el ambiente y sus posibles acciones, teniendo entonces por cada estado los llamados valores Q de las acciones, en donde el valor mayor de Q indica que es más probable que tomando esa acción en ese estado se obtenga la mayor recompensa. Por cada acción que se toma, el estado actual de la tabla se actualiza mediante la ecuación de optimalidad basada en la ecuación de Bellman:

$$Q^{new}(s_t, a_t) \leftarrow (1-\alpha) \cdot Q(s_t, a_t) + \alpha \cdot (r_t + \gamma^t \cdot \underbrace{max}_{a} Q(s_{t+1}, a)) \qquad (10)$$

Donde α es la tasa de aprendizaje que a su vez es un hiperparámetro que toma valores entre 0 y 1 y controla cuanto cambia el peso Q en respuesta a la recompensa obtenida. $s_t \; y \; a_t$ son el estado y la acción actual respectivamente. γ es un factor de descuento y r_t es la recompensa obtenida. Esta ecuación actualiza el valor Q de la acción actual que conlleva al estado siguiente con mayor recompensa, priorizando las acciones que logren esto a corto plazo por medio del factor de descuento, un número entre 0.5 y $0.\hat{9}$ que va quitándole valor a la recompensa a medida que las iteraciones pasan. Con este método se han obtenido los resultados mostrados en la Figura 41 donde se evidencia la trayectoria final de un agente en un ambiente distinto al que se entrenó.

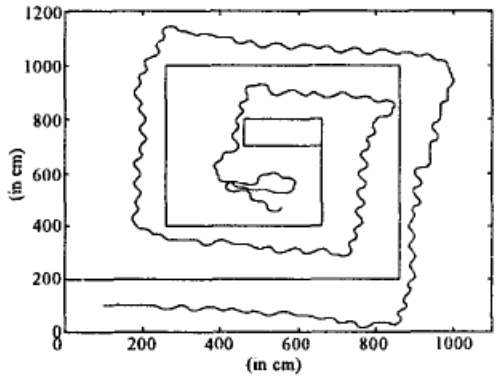

Figura 41. Planeación de trayectorias y evasión de obstáculos con método de aprendizaje por refuerzo. Tomado de [4].

Otro método más reciente es el utilizado en [22], donde se aplican dos conceptos, las redes neuronales y los algoritmos genéticos, en una combinación conocida como NEAT (Neuroevolution of augmenting topologies) el cual consiste en la evolución de redes neuronales por medio de los conceptos de cruza, mutación, elitismo, entre otros, traídos de los algoritmos genéticos. En este algoritmo no solo se cambian los pesos de la red neuronal, sino que también se cambia la topología de la red en cuanto a conexiones y número de neuronas.

Por último, está el método Deep Reinforcement Learning (DRL) en el que se combinan los métodos de aprendizaje por refuerzo y las redes neuronales. Este es incluso un tema en sí mismo por lo que existen varios algoritmos para su implementación mostrados más adelante [40].

Tanto en la selección del robot móvil como en la selección del método de planeación de trayectorias, se enfrenta nuevamente la postura de elegir uno o varios entre muchos métodos. Los dos primeros métodos mencionados, están enfocados en tomar acciones sobre el sistema de control de bajo nivel del robot móvil, esto es, sobre las ruedas, y debido a que el Turtlebot 3 no permite control directamente sobre sus ruedas quedan descartados. En cuanto al método de Q-Learning es necesaria una representación tabular de los estados del robot lo cual en el mejor de los casos supone una tabla con 100^{100} elementos, lo cual es computacionalmente inviable. En este sentido quedan dos métodos, NEAT y DRL. La evaluación se realizará sobre estos dos.

A pesar de no haber seleccionado todavía ningún método específico para implementar DRL, se sabe que las redes neuronales llevan tiempos considerables de entrenamiento al igual que los algoritmos genéticos y ambos dependen de una función modelada por el usuario de su desempeño. En el caso de los algoritmos genéticos se llama función de idoneidad o fitness en inglés y en el caso de DRL se debe modelar la recompensa obtenida por el agente en el ambiente.

Como se ve a lo largo de este documento, se ha venido trabajando sobre un modelo discretizado por celdas exactas del entorno. Con este se realizó la planeación de trayectorias, las pruebas de control y más adelante se muestran pruebas de reconocimiento de obstáculos con el mismo modelo. Este modelo es una representación simplificada del espacio de trabajo seleccionado, el laboratorio de robótica de la Universidad Militar Nueva Granada, considerando sus dimensiones y las del robot, lo que lo hace un candidato a considerar para hacer simulaciones sobre cualquiera de los dos métodos ya que se tienen bajo control todas las variables del entorno tal como las paredes, y los obstáculos. Así mismo, si se utiliza este modelo simplificado, se reduce el costo computacional en el proceso de entrenamiento ya que se ha prescindido del simulador 3D y en cambio se tiene un modelo que si es necesario se puede omitir el apartado gráfico que supone un costo computacional alto.

El último factor por considerar es el tiempo de implementación de cada método, en donde NEAT se destaca por contar con una librería la cual se encarga de los procesos de creación y evolución de las redes neuronales [34], en contraste con DRL, que si bien existen librerías como Tensorflow [41] y el agente sería uno solo el proceso supone un tiempo de implementación más largo teniendo en cuenta además que es necesario seleccionar uno de los métodos de la Figura 45, sin mencionar que existen más métodos de los ahí mostrados.

4.3 NEAT (NEUROEVOLUTION OF AUGMENTING TOPOLOGIES)

La Neuro-evolución de topologías aumentadas trata de la evolución de redes neuronales con algoritmos genéticos [34]. La representación de la red se hace por medio de un grafo llamado fenotipo y este a su vez se representa por una tabla llamada genoma o genotipo. Éste contiene una lista de genes de conexión que a su vez se conforman por dos nodos; cada gen de conexión contiene un nodo de entrada, un nodo de salida, su peso de conexión y si está habilitado o no. Un ejemplo de grafo se

observa en la Figura 42 donde se observan 3 nodos de entradas enumerados por 1, 2 y 3; los nodos 1 y 4 conforman un gen de conexión donde el nodo 1 es la entrada, 4 es el nodo de salida, su peso de conexión puede ser 0.7 y puede estar habilitado. Siguiendo esta metodología, el genoma de la red neuronal mostrada en la Figura 42 se evidencia en la Figura 43.

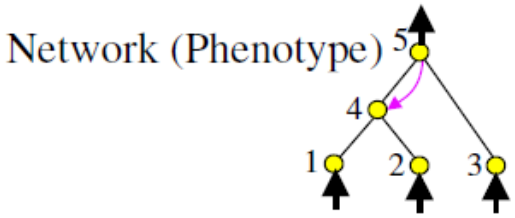

Figura 42. Fenotipo de ejemplo. Tomado de [34]

Genome (Genotype)

Node Genes	Node 1 Sensor Input	Node 2 Sensor Input	Node 3 Sensor Input	Node 4 Hidden Hidden	Node 5 Hidden Output
Connect. Genes	In 1 Out 4 Weight 0.7 Enabled Innov 1	In 2 Out 4 Weight 0.5 Enabled Innov 3	In 2 Out 5 Weight 0.5 DISAB Innov 4	In 3 Out 5 Weight 0.2 Enabled Innov 5	In 4 Out 5 Weight 0.4 Enabled Innov 6	In 5 Out 4 Weight 0.6 Enabled Innov 10

Figura 43. Genoma de red neuronal de la Figura 42. Tomado de [34]

El proceso de mutación puede ser de dos tipos: se añaden conexiones (Figura 44 izquierda) o se añaden nodos de conexión (Figura 44 derecha). Como se ve en la Figura 43 cada gen de conexión también es acompañado por un número llamado número de innovación. Éste es un número que indica el orden de aparición de cada nodo de conexión; por ejemplo, teniendo en cuenta que el número actual de innovación es 6, si las dos mutaciones fueron generadas una después de la otra, la mutación generada por agregar la conexión se le asigna el número de innovación 7, y al añadir un nodo de conexión se añaden dos conexiones. A éstas se le asignan los números 8 y 9, es por esto que en la Figura 43, la salida 5 tiene número de innovación 10. De esta forma se puede realizar la cruza sin importar la topología de la red. Los nodos de conexión con números de innovación iguales se pasan directamente y de los que no, se elige el que tenga una idoneidad más alta.

Figura 44. Tipos de mutación. Tomado de: [38]

4.4 DRL (DEEP REINFORCEMENT LEARNING)

Como se mencionó, el aprendizaje por refuerzo es un tipo de aprendizaje automático, el cual trata con un problema que comúnmente se llama par estado-acción o por su nombre en inglés *state-action pairs*, es por esto que se habla acerca de un ambiente que es el que entrega los estados o de los cuales el agente "ve" los estados (puede "ver" el ambiente completo en cuyo caso se llamaría un ambiente completamente observable o, puede "ver" solo una parte del ambiente y se llamaría un ambiente parcialmente observable) y dentro de este ambiente el agente puede tomar acciones por las cuales recibirá una recompensa. El fin del aprendizaje por refuerzo será siempre maximizar la recompensa total.

Para entender cómo se alcanza la meta del aprendizaje por refuerzo es necesario entonces formalizar los términos que involucran a este. El primero es la política o por su nombre en inglés *Policy* que es la norma usada por el agente la cual le dice que acción a_t tomar, esta puede ser de tipo determinista,

$$a_t = \mu(s_t) \tag{11}$$

Donde s_t es el estado en cierto tiempo t, O puede ser estocástica,

$$a_t \sim \pi(\cdot\,|s_t) \tag{12}$$

La principal diferencia de una con respecto a la otra es el nivel de exploración que alcanzan. Una política determinista tendrá niveles de exploración bajos debido a que siempre realizará las mismas

acciones en un estado determinado y por el contrario una política estocástica tendrá niveles de exploración altos debido a que cuenta con un factor aleatorio en su política. Formalmente, la recompensa total se le llama Retorno o por su nombre en inglés *Return*. Ésta se puede expresar como una función R y es la suma de las recompensas adquiridas a partir de un tiempo t. Este retorno puede ser sin descuento y finito,

$$R(\tau) = \sum_{t=0}^{T} r_t \tag{13}$$

O puede ser de con descuento e infinito,

$$R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t \tag{14}$$

Donde γ es el factor de descuento y puede adquirir valores entre 0 y 1. Entre más lejos al 0 significa que se les da más importancia a las recompensas futuras. De este último término nace una expresión llamada función Q que recibe dos parámetros, un estado y una acción y devuelve la esperanza del retorno con descuento infinito que se puede obtener en el futuro.

$$Q(s, a) = \mathbb{E}[R_t] \tag{15}$$

En cada estado *s*, hay un conjunto de acciones a. Como se desea maximizar el retorno o la recompensa total, normalmente se escoge la acción en ese estado que tenga el mayor valor Q. Nótese que la función Q se puede representar como una tabla donde por cada estado *s*, cada acción a tendrá su valor Q. Estas son las dos maneras de seleccionar una acción en un ambiente, ya sea con la política o con la función Q. Los algoritmos que se enfocan en la política entran en una categoría llamada optimización de política o por su nombre en inglés Policy Optimization y los que se enfocan en la función Q entran en la categoría de *Q-learning O* Value learning.

Para un espacio de estados pequeños en una estrategia basada en la función Q como se mencionó anteriormente, se puede hacer una representación tabular e iterativamente aplicar la ecuación (10) y

así optimizar la función Q, pero el problema está cuando el espacio de estados es muy grande hasta el punto de que se vuelve computacionalmente inviable almacenar todos estos estados y sus respectivos valores Q dentro de la tabla.

Es aquí donde entran en juego las redes neuronales como aproximadores universales de funciones. En la Figura 45 se muestran algunos algoritmos de aprendizaje por refuerzo con redes neuronales separados por diferentes categorías, según si son basados o no en un modelo del entorno donde este modelo del entorno se refiere a una función que prediga las transiciones entre estados y las recompensas esperadas a obtener. En la categoría que no dependen del modelo (Model-Free RL) se desprenden los dos tipos antes mencionados, basados en la optimización de la política y los basados en la función Q. En cuanto a los algoritmos que pertenecen a la categoría basadas en el modelo se dividen en dos tipos, uno donde el agente aprende el modelo y otra donde el modelo está dado.

Actualmente se cuenta con una representación del entorno de trabajo, el laboratorio de robótica de la Universidad Militar Nueva Granada, sin embargo, no se tiene un modelo de este por lo que se procede a presentar algunas características de varios de los algoritmos de la categoría que no dependen de un modelo.

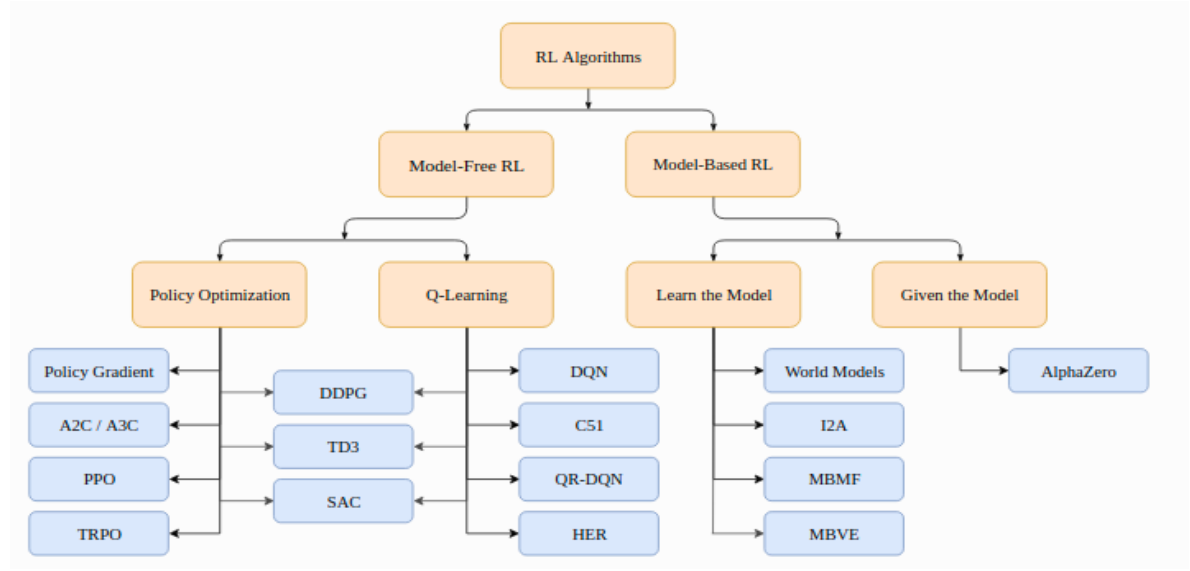

Figura 45. Taxonomía de algoritmos de aprendizaje por refuerzo. Tomado de: [39]

4.5 ACONDICIONAMIENTO DEL MÓVIL PARA LA CONDUCCIÓN AUTÓNOMA

4.5.1 Detección de obstáculos

Antes de proceder con la evasión de los obstáculos dinámicos, es necesario que el robot identifique estos obstáculos. Para esta tarea se utiliza la información proporcionada por el sensor LIDAR, primero se identifica cómo entrega el robot y ROS la información, esto es, por medio de un tópico llamado /scan, y este a su vez envía la información en forma de un mensaje llamado "/LaserScan". Como se observa en la Figura 46, este mensaje nos da toda la información acerca del sensor como los rangos máximos y mínimos, los ángulos máximos y mínimos y lo que en este caso nos concierne, las lecturas de distancia en metros. En términos de programación, el mensaje en su sección de rangos nos entrega un vector de 360º con una medida de distancia para cada ángulo.

sensor_msgs/LaserScan Message

File: sensor_msgs/LaserScan.msg

Raw Message Definition

```
# Single scan from a planar laser range-finder
#
# If you have another ranging device with different behavior (e.g. a sonar
# array), please find or create a different message, since applications
# will make fairly laser-specific assumptions about this data

Header header            # timestamp in the header is the acquisition time of
                         # the first ray in the scan.
                         #
                         # in frame frame_id, angles are measured around
                         # the positive Z axis (counterclockwise, if Z is up)
                         # with zero angle being forward along the x axis

float32 angle_min        # start angle of the scan [rad]
float32 angle_max        # end angle of the scan [rad]
float32 angle_increment  # angular distance between measurements [rad]

float32 time_increment   # time between measurements [seconds] - if your scanner
                         # is moving, this will be used in interpolating position
                         # of 3d points
float32 scan_time        # time between scans [seconds]

float32 range_min        # minimum range value [m]
float32 range_max        # maximum range value [m]

float32[] ranges         # range data [m] (Note: values < range_min or > range_max should be discarded)
float32[] intensities    # intensity data [device-specific units].  If your
                         # device does not provide intensities, please leave
                         # the array empty.
```

Figura 46. Información del mensaje LaserScan.
Tomado de: http://docs.ros.org/melodic/api/sensor_msgs/html/msg/LaserScan.html

Cuando se tienen todas las medidas de distancia, hay que procesarlas de tal manera que nos den una posición en el espacio de trabajo. Para este cálculo se realizó el modelo mostrado en la Figura 47, en el que se muestra una posición arbitraria (x_r, y_r) del robot con una orientación θ, un obstáculo ubicado detectado en (x_o, y_o) a una distancia d del robot.

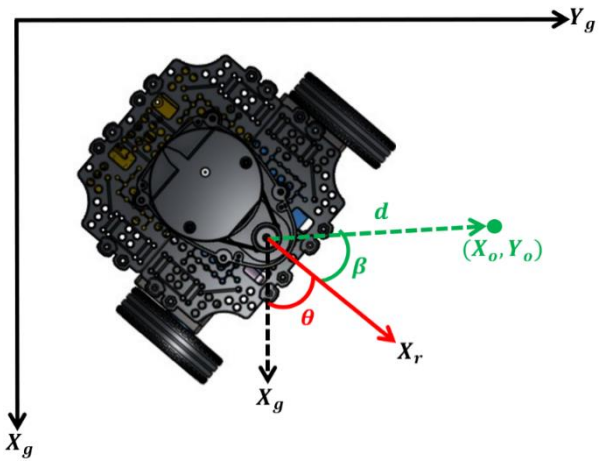

Figura 47. Visualización del estado de un obstáculo arbitrario para su detección. Elaboración propia.

La medida detectada por el robot es la distancia d y el ángulo β que es el ángulo de orientación de la medición, es decir, el ángulo formado entre el eje X_r del robot y el obstáculo. Para calcular las coordenadas del obstáculo con respecto a los ejes globales X_g y Y_g se utilizan las siguientes ecuaciones:

$$X_o = X_r + d * cos(\theta + \beta) \quad (16)$$

$$Y_o = Y_r + d * sin(\theta + \beta) \quad (17)$$

Las ecuaciones (16) y (17) se utilizan para cada medida en los 360° que proporciona el sensor y se obtiene como resultado la Figura 48, nótese los obstáculos se pueden ubicar en cualquier posición del entorno, es decir, que no deben estar ubicados en exactamente en celdas y es tarea del algoritmo de detección ubicarlos en las celdas más cercanas a la ubicación real de estos. Las celdas de color amarillo representan los obstáculos detectados por el algoritmo y las celdas de color azul representan el camino seguido por el robot. Nótese que alrededor del camino se encuentran celdas de color verde y van bajando la tonalidad hasta ser de color blanco, esto es debido a que se añadieron *zonas de riesgo* para clasificar el riesgo que representa un obstáculo que se encuentre alrededor de este, siendo las celdas adyacentes al camino denominadas de alto riesgo, las que se encuentren dos casillas alejadas del camino de riesgo medio y las que se encuentren tres casillas alejadas del camino de riesgo bajo.

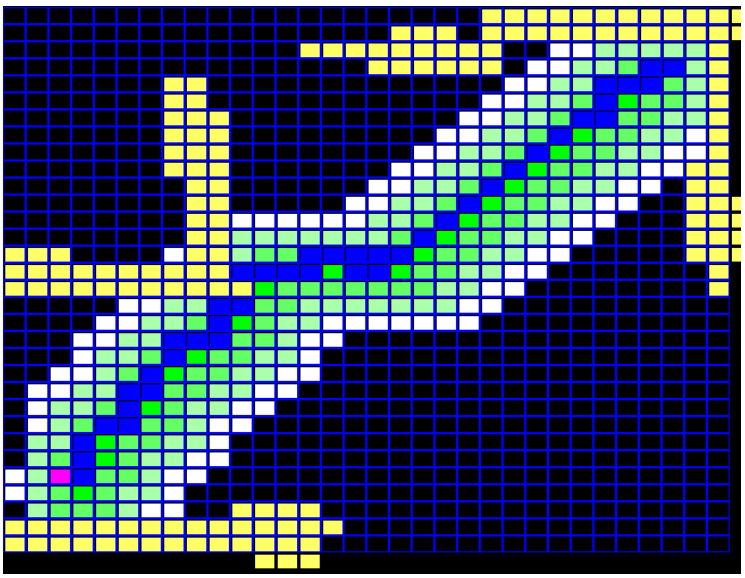

Figura 48. Prueba de detección de obstáculos con el modelo anterior. Elaboración propia.

Como se tienen dos algoritmos de control, uno de planeación de trayectoria inicial y uno de evasión de obstáculos, es necesario tener una conmutación entre ambos, se decidió que el algoritmo de inteligencia artificial sólo tomará el mando cuando el robot detecte que hay un obstáculo en una casilla de alto riesgo tal como se muestra en el diagrama de la Figura 49.

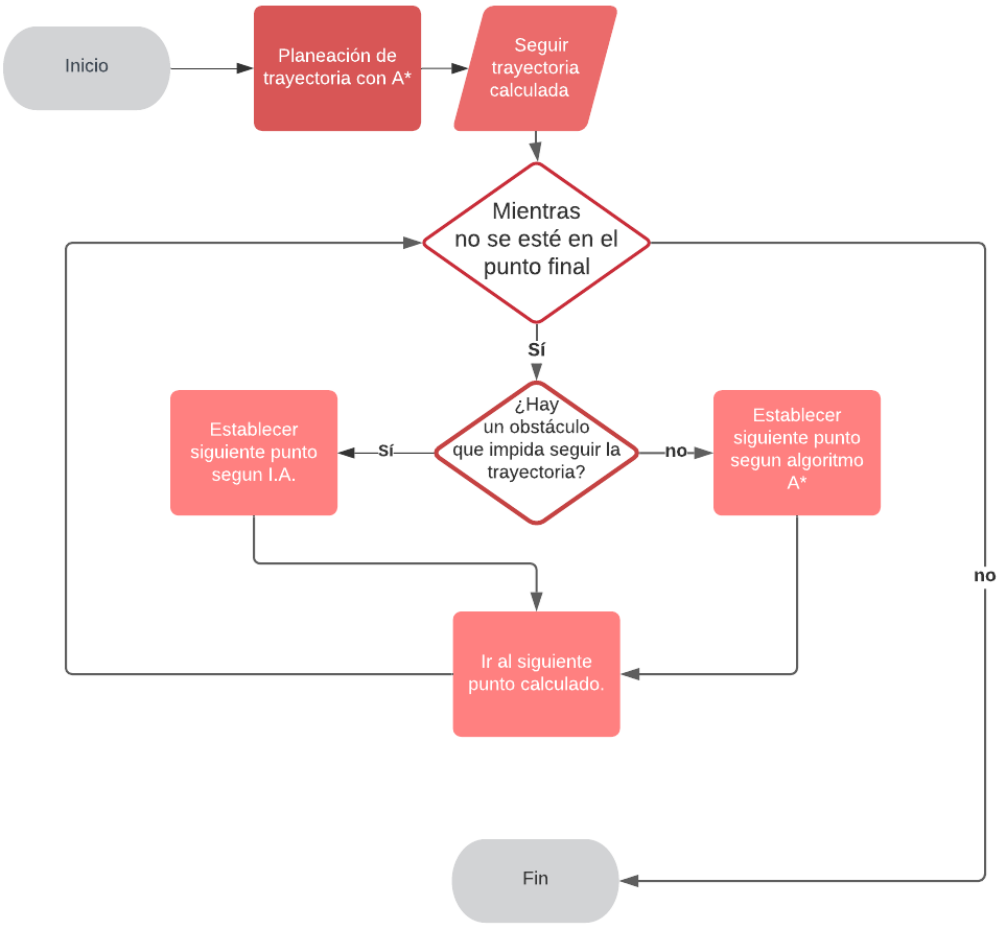

Figura 49. Diagrama de flujo para la conmutación entre algoritmo de I.A. y planeación de trayectorias. Elaboración propia.

4.5.2 Tipos de obstáculos

Antes de la implementación de cualquiera de los métodos es importante la definición de los tipos de obstáculos con los que el móvil se va a tener que enfrentar. Como se definió en los objetivos del proyecto, estos obstáculos serán específicos lo cual quiere decir que tendrán un comportamiento predefinido, es decir, variables como velocidad, tamaño y forma están antes definidas.

El primer tipo se define como obstáculos que no estaban contemplados en la planeación de la trayectoria, pero luego aparecen de manera espontánea en una posición arbitraria e interfieren con la trayectoria planeada, estos tendrán forma de cajas cuadradas de 71.6 cm de largo que es el promedio de la distancia de una zancada de una persona promedio [42], este tipo de obstáculo se evidencia en la Figura 50 y hacen referencia a situaciones impredecibles del entorno en el cual aparecen objetos

espontáneamente como por ejemplo algún objeto que se cayó o algún ser humano que se posicionó estático.

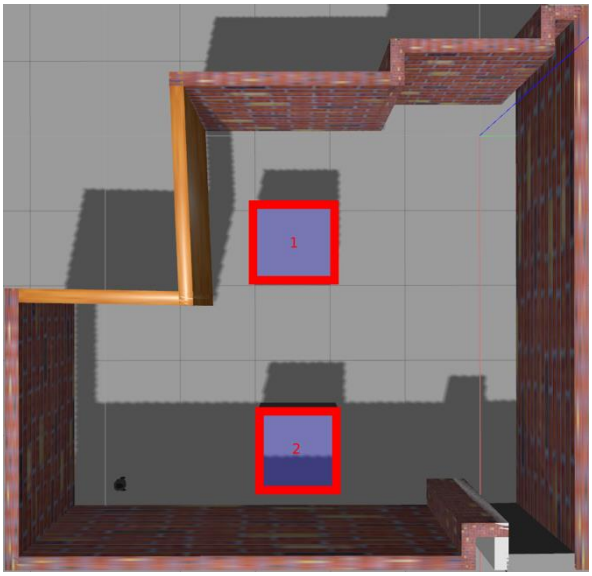

Figura 50. Obstáculo tipo 1. Elaboración propia.

El siguiente tipo se define como un obstáculo de tipo móvil, no está al principio de la planeación de la trayectoria y aparece espontáneamente y se mueve a velocidad constante de 0.06 m/s de tal manera que se atraviesa en la trayectoria inicial del móvil, tal como se observa en la Figura 51 y estos hacen referencia a situaciones impredecibles del entorno como por ejemplo otro robot móvil con trayectoria rectilinea.

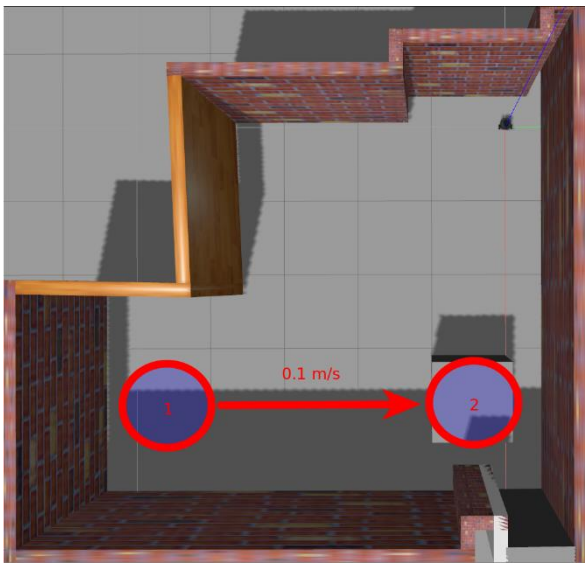

Figura 51. Obstáculo tipo 2. Elaboración propia.

El último tipo se define como un obstáculo que no estaba en la planeación de trayectoria inicial, pero aparece espontáneamente y se mueve a velocidad constante y en un momento arbitrario cambia de dirección y velocidad como se muestra en la Figura 52 y hacen referencia a situaciones impredecibles del entorno como por ejemplo una persona que al caminar cambia su rumbo.

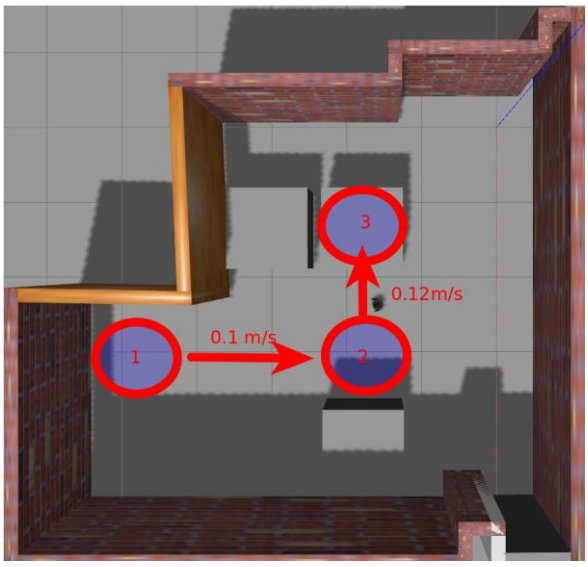

Figura 52. Obstáculo tipo 3. Elaboración propia.

5 DESARROLLO E IMPLEMENTACIÓN DE LOS MÉTODOS DE INTELIGENCIA ARTIFICIAL

5.1 NEAT

Este método se implementó con ayuda de una librería de código abierto llamada Neat-python [34], esta facilita la implementación de los algoritmos genéticos utilizados para el entrenamiento de las redes neuronales. Para su funcionamiento la librería lee un archivo de parámetros modificables entre los cuales se suministró la arquitectura inicial de la red, la función de idoneidad *o fitness* con la que se evaluará los *agentes* que mejor se desempeñen en el ambiente y el criterio de selección de fitness, este último se ingresó de tal manera que el mejor fitness sea el valor más alto.

Para la arquitectura inicial de la red se debe tener nuevamente en cuenta el modelo de celdas aproximadas con el que se ha venido trabajando, los datos de entrada se toman de las celdas alrededor del robot donde se obtienen datos del sensor LIDAR que conforman 361 celdas a la redonda, y para indicarle al robot la posición del obstáculo se agregaron 8 entradas más que corresponden a cada una de las direcciones. Para normalizar estas entradas se ingresó 1 en las casillas donde había obstáculos y 0 en las que no, de igual manera se ingresó 1 en la dirección relativa del obstáculo con respecto al robot y 0 en las demás direcciones, en total quedaron en total 369 entradas. Esto se representa de manera visual en la Figura 53 donde se denota la posición actual del robot con un rectángulo de color fucsia, los obstáculos detectados por el robot con rectángulos de color blanco, el objetivo final con un rectángulo de color amarillo y las celdas vacías con rectángulos de color negro. El vector de entradas se genera en dos partes, la es la detección de obstáculos donde una celda donde se detecte un obstáculo se representa con un 1 y donde no, con un 0 y cada una de las celdas tiene su posición en el vector como se representa en la Figura 53 (izquierda). Hasta este punto se habrían llenado 361 posiciones del vector, las restantes se llenan con la dirección relativa del obstáculo con respecto a la posición del móvil tal que la dirección mas cercana se representa con un 1y las demás como un 0, cada celda tendrá su posición definida en el vector tal como se indica en la Figura 53 (derecha). Al final se tiene el vector de 369 entradas indicado en la Figura 53(abajo).

En cuanto a las salidas, se decidieron 5 tipos de acciones las cuales corresponden a moverse una cuadrícula arriba, abajo, derecha o izquierda y una quinta acción, quedarse en la posición actual.

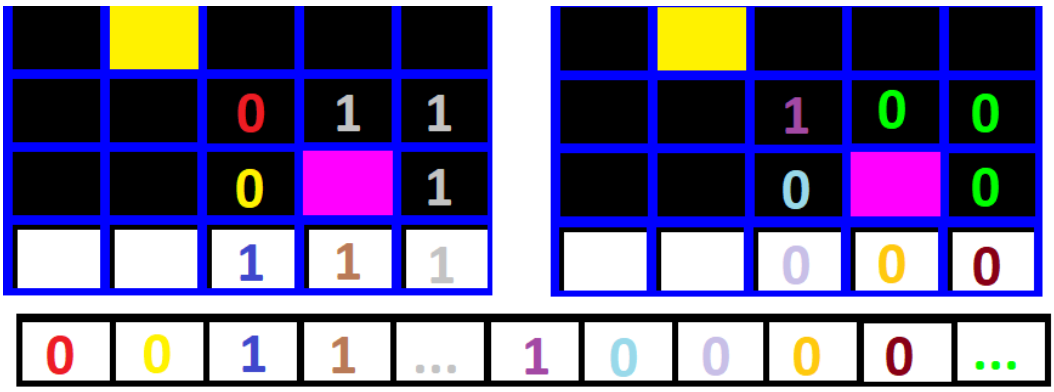

Figura 53. Generación de vector de entradas de red neuronal algoritmo NEAT. Elaboración propia.

Antes de exponer el número de capas ocultas iniciales de la red, es pertinente indicar cuál fue la función de idoneidad. Ésta se enfocó de tal manera que cada agente empiece con un fitness de 200, el cual se actualiza con una función exponencial dependiente de distancia a la cual quedó del obstáculo al final de su recorrido (18) y una penalización de 1 por cada paso tomado.

$$f_{itness}(distancia, pasos) = \frac{200}{\sqrt[3]{distancia}} - pasos \qquad (18)$$

Una vez indicada la función de fitness, se procede a preparar los entrenamientos. El primero se realizó con 7 capas ocultas, todas con función de activación sigmoide, 500 individuos como población inicial y por 50 generaciones, además del obstáculo tipo uno definido en la sección 554.5.2.

Obteniendo un resultado caótico mostrado en la Figura 54 se evidencia un comportamiento que es causado posiblemente por una arquitectura de red inadecuada o insuficientes generaciones, aunque se evidencian unos picos en el fitness que indican que el robot llega al punto final, es muy importante tener estabilidad en el modelo para descartar que sea aleatorio el hecho de que llega al punto final.

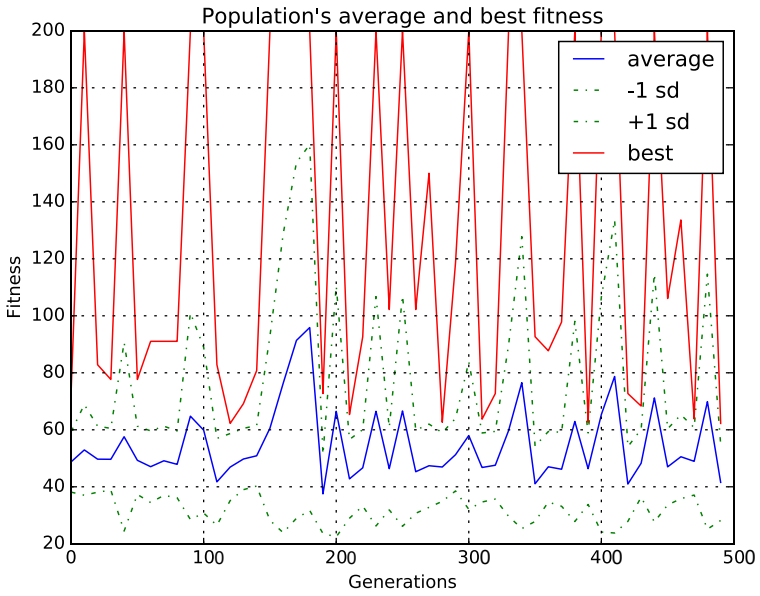

Figura 54. Fitness vs. Generaciones para arquitectura [361, 7, 9]. Elaboración propia.

Para superar este problema se pueden hacer cambios en los parámetros de entrenamientos, pero teniendo en cuenta que el tiempo que llevó entrenar el modelo anterior fue de 4 días utilizando una laptop con un procesador Intel Core I7-5500U, se tomó la alternativa de cambiar el número de capas ocultas iniciales a 12, una población inicial de 200 y la función de activación se cambió por *relu además de aumentar las generaciones a 1000*. En la Figura 55 se observan los resultados a partir de la generación 100. Nótese que, cambiando los parámetros de la red inicial, se afecta de manera directa los resultados finales.

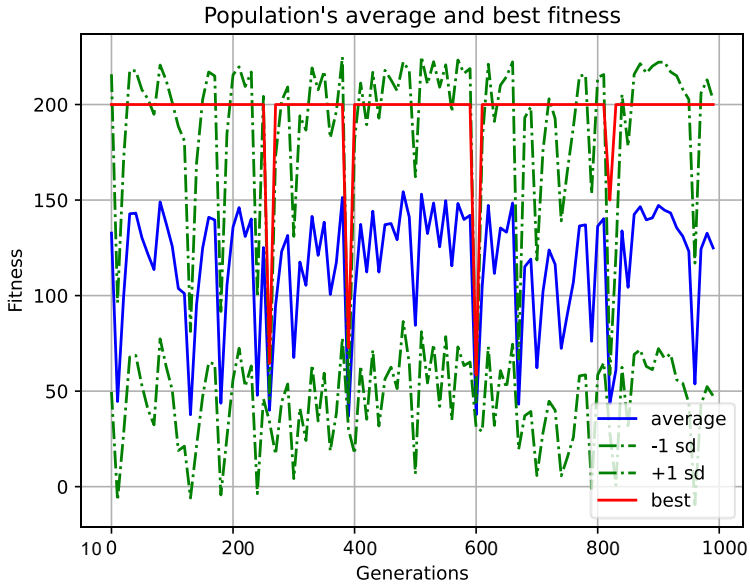

Figura 55. Fitness vs Generación para arquitectura [361, 12, 9]. Elabroación propia.

Para mejorar aún más estos resultados se cambia un poco más los parámetros de entrenamiento, pero esta vez con un concepto llamado *Aprendizaje Curricular (*Curriculum Learning) [43] el cual consiste en aumentar gradualmente la dificultad en los entrenamientos para que, al final del entrenamiento, el modelo tenga un preentrenamiento que facilite que se complete la tarea final de una manera más fluida. Esto se implementó añadiendo 3 *niveles* de dificultad al entorno. En un principio la tarea del robot será la de ir de un punto inicial a otro punto final y cercano. Con esta red pre-entrenada se añaden obstáculos en el camino de manera aleatoria y se distancian más el punto final del inicial, y por último el punto final e inicial estarían lo más alejados posible y los obstáculos serían de todos los tipos definidos. Los resultados finales de estos entrenamientos se evidencian en la Figura 56.

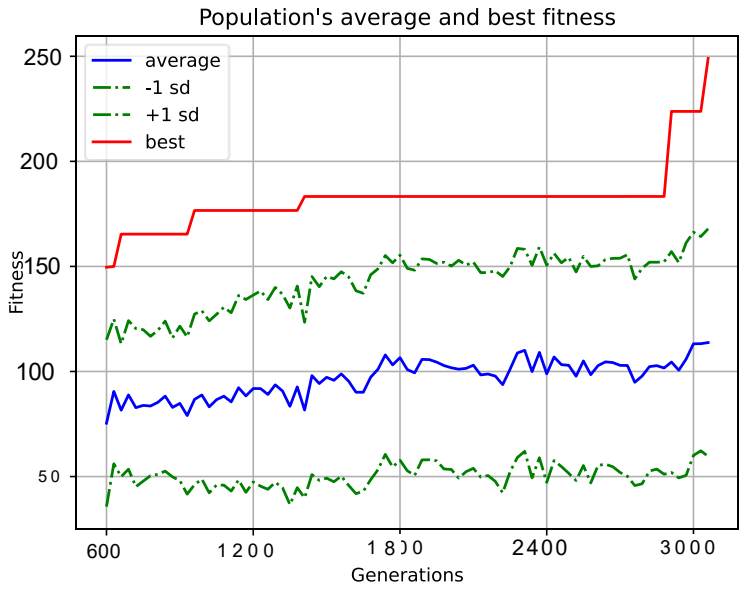

Figura 56. Fitness vs. Generación aplicando aprendizaje curricular. Elaboración propia.

Con el aprendizaje curricular aplicado se evidencia una clara mejora en cuanto a la estabilidad y continuidad del aprendizaje. Aunque se aumentó el tiempo de entrenamiento, fue posible entrenar por más generaciones sin generar estancamientos ni sobre entrenamientos [44].

5.2 DRL

En primer lugar, se definió la librería Tensorflow [41] para la implementación de las redes neuronales a utilizar debido a que ésta aprovecha la aceleración por hardware en las operaciones que se demandan. Para el entrenamiento se requiere un ambiente, por lo que se utilizó el mismo ambiente discretizado que se ha venido trabajando y para el agente se aprovechó el modelo del móvil implementado en el algoritmo anterior. Normalmente, en las aplicaciones con DRL en su red neuronal se implementa una primera capa de tipo convolucional, esto debido a que utilizan imágenes del ambiente como entrada al modelo en este caso se implementó una red neuronal que según la documentación de Tensorflow se denomina Dense (Densa), lo que significa que todas sus neuronas de una capa están conectadas entre sí (Red neuronal convencional), a razón de que se aprovechó la entrada normalizada de 368 pixeles a la redonda del móvil utilizada en el modelo anterior. El tiempo de entrenamiento de este algoritmo fue de 6 días utilizando una laptop con una GPU Nvidia GeForce GTX 1050, un poco mayor a la del modelo anterior.

La topología de la red neuronal que utiliza este modelo es constante con el tiempo. Se optó por implementar 4 capas ocultas de 64 neuronas cada una. En cuanto a los parámetros de actualización de la red (ecuación 10) se muestran en la siguiente tabla:

Tabla 9. Parámetros para DRL.

Parámetro	Valor
α	0.0025
γ	0.99
ε_{decay}	0.99

Nótese que la variable ε_{decay} no se encuentra originalmente en la ecuación 10 y se refiere al decaimiento de la variable ε que se utiliza para la exploración de estados del ambiente. Inicialmente el robot no obedece siempre a la red neuronal que se está entrenando, sino que con probabilidad de ε se escoge entre la acción obtenida por el modelo de red neuronal o se una acción aleatoria, debido a que inicialmente se ε tiene un valor de 1, durante los primeros pasos de entrenamiento el móvil solo tomará acciones aleatorias, fase que se conoce como exploración. A medida que las épocas de entramiento pasan, ε va decayendo a razón de ε_{decay} tal que al final cuando se supone un modelo entrenado si se obedece en su mayoría, esto porque ε tiene un valor mínimo.

6 DEFINICIÓN DE PRUEBAS Y ANÁLISIS DE RESULTADOS

6.1 DEFINICIÓN DE PRUEBAS

Para evaluar el desempeño de las redes neuronales entrenadas en la sección anterior, se diseñó una serie de pasos que el entorno va a seguir para cada uno de los tipos de obstáculos definidos. Para hacer las pruebas comparables se toma el mismo punto de inicio y el mismo punto final. Las pruebas se enfocan en evaluar los 3 componentes del algoritmo diseñado, la planeación de trayectoria, el algoritmo de toma de decisiones o conmutación y el de evasión de obstáculos dinámicos. Para el algoritmo de planeación de trayectorias se puso un obstáculo en forma de cuadrado en la mitad del entorno el cual se deberá ser rodeado; al ser virtual este obstáculo, cualquier otro que se coloque después de que el móvil empieza su ejecución deberá ser evaluado por el algoritmo de conmutación si ese interfiere con la planeación inicial. Por último, se posicionan los tipos de obstáculos 1, 2, 3 según corresponda. Al final de cada ejecución se grafica la trayectoria final y se compara con la planeación inicial además de indicar si la prueba fue exitosa o hubo algún inconveniente como puede ser un choque. Se realizó un total de 10 repeticiones para cada tipo de obstáculo con cada uno de los algoritmos.

6.2 NEAT

La planeación inicial de la trayectoria se evidencia en la Figura 57. Se notan además unos cuadrados rojos agregados a propósito, que corresponden a las paredes del entrono para evitar que el móvil los atraviese.

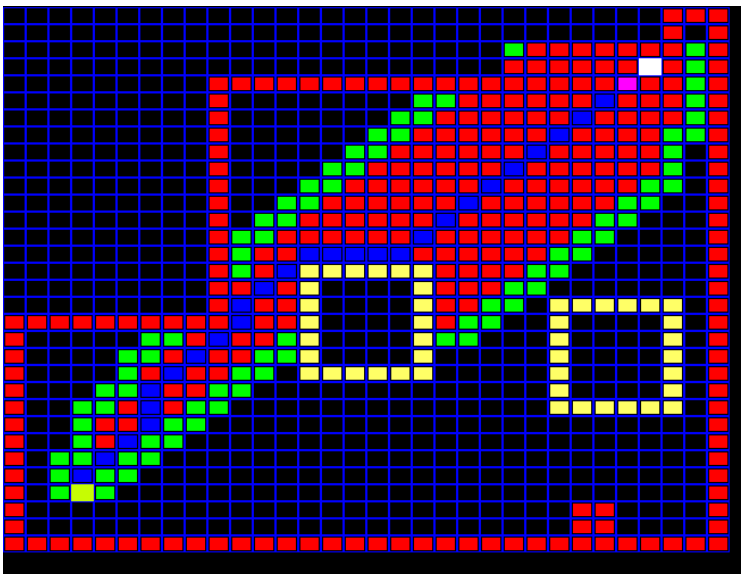

Figura 57. Planeación inicial de trayectoria.

A continuación, se muestra la trayectoria final con el obstáculo de tipo 1, este se representa en la figura con un rectángulo de color rojo.

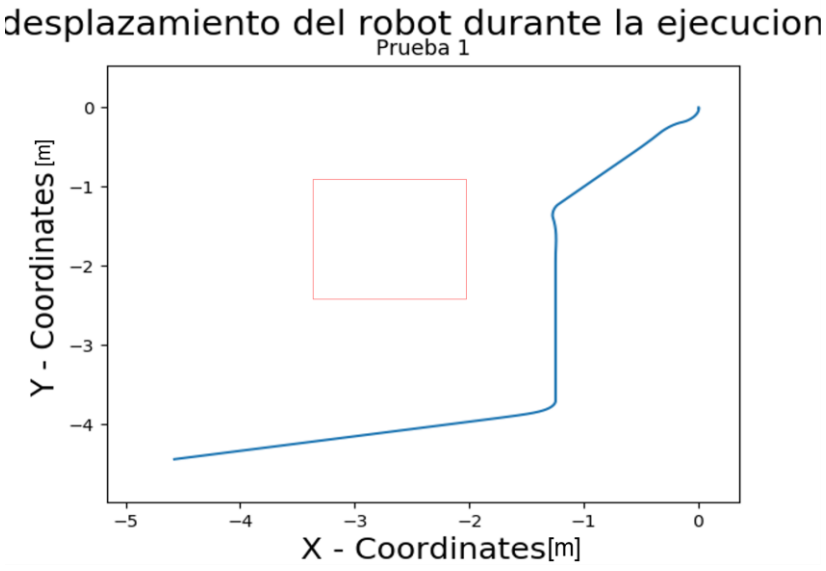

Figura 58. Trayectoria final con conmutación de algoritmos. Elaboración propia.

Nótese cómo el móvil intenta seguir su trayectoria inicial, pero una vez que detecta que un obstáculo no lo permite ese paso (rectángulo rojo en Figura 58) se desvía tomando una ruta alternativa controlada por la inteligencia artificial.

Para los obstáculos móviles este algoritmo no funcionó de la manera esperada dado que falló para las pruebas con los tipos de algoritmos de tipo 2 y 3. En la Figura 59 se observa un ejemplo de una de las pruebas y cómo el móvil se estrella con el obstáculo en movimiento cerca de la coordenada (-1.2, -3.8). Se observa también la similitud en la trayectoria con la de la Figura 58, de lo cual se puede inferir una posible falta de diversidad a la hora de entrenamiento, lo cual lleva a la red a actuar de manera similar en cada ejecución.

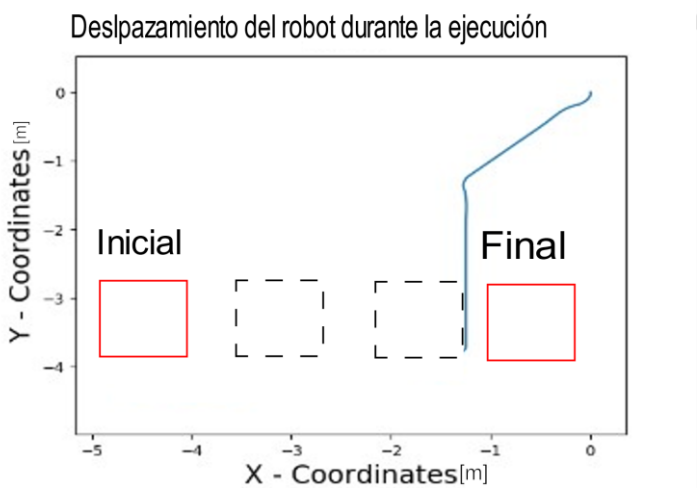

Figura 59 Trayectoria final NEAT. Obstáculo tipo 2. Prueba fallida. Elaboración propia.

6.3 DRL

En las Figura 60, Figura 61 y Figura 62 se muestran las trayectorias finales para los tipos de obstáculos 1, 2 y 3 respectivamente.

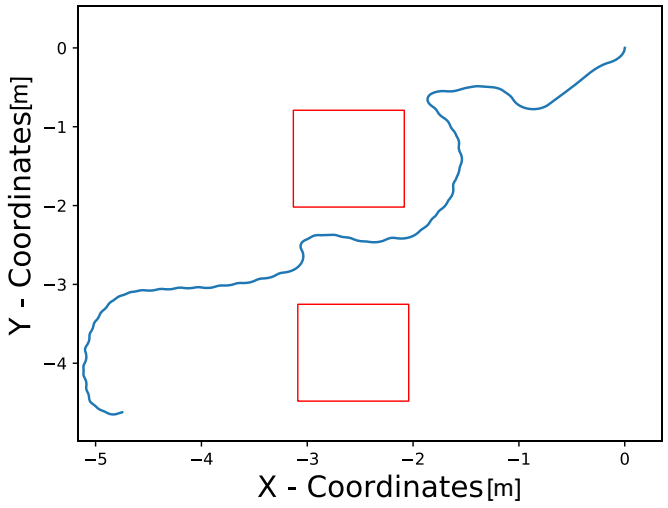

Figura 60. Trazado de ruta realizada con algoritmo DRL para obstáculo tipo 1. Elaboración propia.

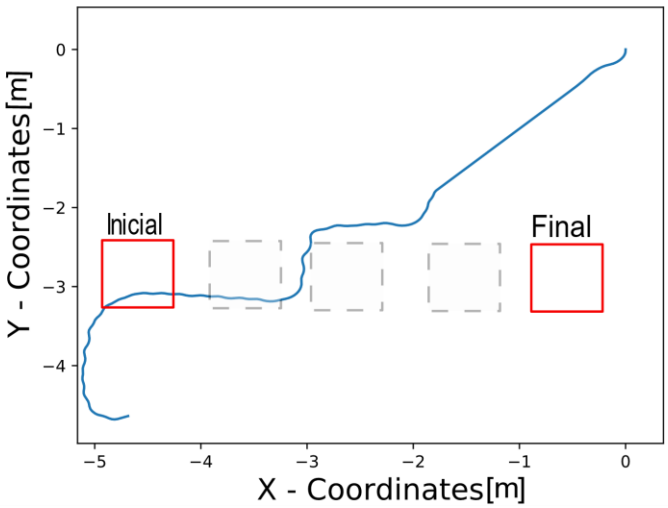

Figura 61. Trazado de ruta realizada con algoritmo DRL para obstáculo tipo 2. Elaboración propia.

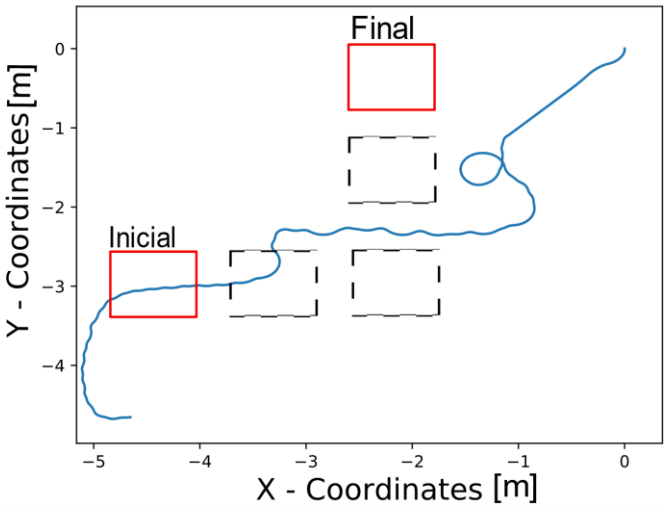

Figura 62. Trazado de ruta realizada con algoritmo DRL para obstáculo tipo 3. Elaboración propia.

Para los obstáculos de tipo 1 y 2 se evidencia que el móvil tiene que desviarse de su trayectoria inicial, pero alcanza el punto final sin ningún tipo de problemas. Nótese que en la Figura 62, para el obstáculo de tipo 3, aunque el móvil alcanza el punto final, este realiza un movimiento circular insólito y no deseado el cual pudo haber sido causado por alguna posición puntual del obstáculo y este tomó la decisión de devolverse, en todo caso esto es uno de los problemas de las redes neuronales, al final funcionan como cajas negras y este tipo de puntos intermedios son indeterminados una manera de evitar estas situaciones es incluir estas situaciones para que sean tomadas en el entrenamiento.

A pesar de que se definen unas características específicas de los obstáculos, una de las bondades de las redes neuronales es su desempeño ante escenarios desconocidos, por lo que se realizaron pequeños cambios en los parámetros de los obstáculos como diferentes posiciones para los de tipo 1 y diferentes velocidades para los tipos 2 y 3. Se realizaron 8 variaciones de cada uno de los tipos de obstáculos obteniendo para cada caso un resultado negativo en el cual el móvil no llegaba a la posición de destino. Un caso en el que el móvil no llega al destino se observa en la Figura 63, en este se modelo con los obstáculos percibidos por el robot como rectángulos de color blanco, la trayectoria hasta el momento recorrida como rectángulos de color verde y la posición actual del móvil como el rectángulo de color rosado, se evidencia como el móvil se encuentra en un estancamiento. Esta prueba fue fallida debido a que el móvil se quedó dando vueltas en el callejón donde se encuentra, se esperaría que tomara la

ruta marcada con el círculo rojo en la Figura 63 pero no fue el caso, debido a que no existe suficiente espacio para el paso.

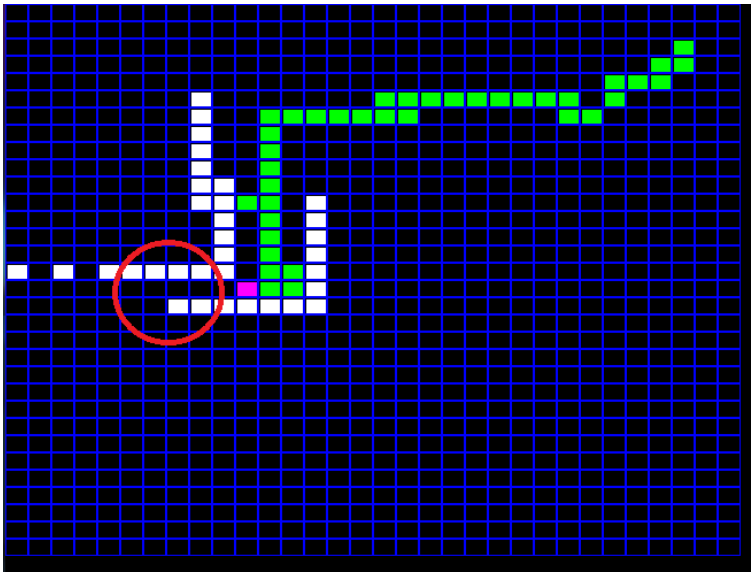

Figura 63. Móvil en ejecución con algoritmo DQL. Prueba fallida. Elaboración propia.

Para el caso de los obstáculos móviles las causales de error fueron las velocidades de los obstáculos que superan a la del móvil por un factor de más de 1.5 por lo que este no tuvo tiempo de evadirlos. Un caso particular se muestra en la Figura 64 en la que se observa que a pesar de que el robot evadió el obstáculo y evitó estrellarse, no llego a la posición final debido a que se quedó dando círculos en la posición mostrada.

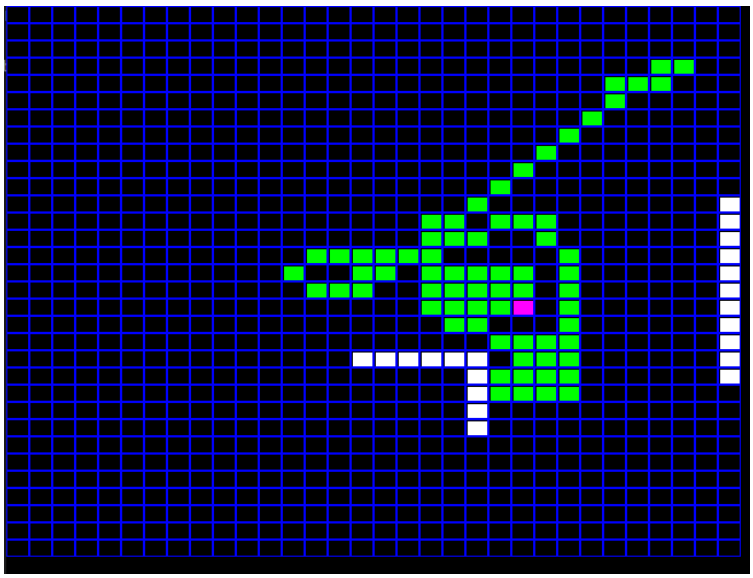

Figura 64. Móvil en ejecución con algoritmo DQL. Obstáculo tipo 2. Prueba fallida. ***Elaboración propia.***

La posible causa de la falla a este último experimento mostrado es la posición relativa del obstáculo con respecto al móvil, debido a que esta misma se replicó llevando a resultados similares, cada vez que el móvil se encuentre sobre la esquina superior derecha del obstáculo con la forma contemplada en la Figura 64, el móvil tiende a quedar dando giros en su posición actual.

En este algoritmo las pruebas fallidas fueron causas de situaciones particulares del entorno que rompieron los parámetros vistos en el entrenamiento, un ejemplo puntual como de la Figura 63 donde visualmente se ve que el robot tiene el espacio para pasar por el espacio reducido entre la pared y el osbtáculo, sin embargo, el algoritmo no toma la decisión de pasar por ahí. Estas situaciones en las pruebas actuales se realizaron con el fin de evaluar cuanto cambio era permitido por el algoritmo de sus condiciones iniciales por lo que aunque este algoritmo falló pueden funcionar como posibles entradas o condiciones en un algoritmo futuro en el que se necesite superar estas situaciones.

7 CONCLUSIONES

El objetivo final del presente trabajo fue la integración de dos algoritmos, uno de planeación de trayectorias con uno de evasión de obstáculos dinámicos específicos con inteligencia artificial, en una estrategia de toma de decisiones e implementarlos en un robot móvil presente en el laboratorio de robótica de la Universidad Militar Nueva Granada. Con esto se buscó realizar un estudio sobre las diferentes técnicas en las diferentes áreas que realizar esta tarea implica, es decir, modelado e implementación de un control de movimiento del robot móvil, planeación de trayectorias para llevarlo de un punto inicial a un punto final teniendo en cuenta obstáculos estáticos y por último técnicas de inteligencia artificial aplicadas a la evasión de obstáculos y a la robótica móvil.

Se pudo observar la importancia del *framework* de robótica ROS con el que fue posible realizar todas las simulaciones y pruebas. Así mismo se hacen todas las operaciones de movimiento y tomas de datos del móvil donde se evidenció la cercanía con la realidad a tal punto que fue posible tomar los datos de la simulación como referencias en el control que se implementó, así mismo fue posible realizar las pruebas de funcionamiento de todos los algoritmos desarrollados sin riesgo de averiar el robot en los posibles choques con paredes, obstáculos, etc. Además, ROS brindó una manera de organizar todo el software desarrollado llevando así a que todo el código escrito para el presente desarrollo se pueda ubicar de manera ágil siguiendo con las prácticas dictadas por ROS.

Durante el estudio de los métodos de planeación de trayectorias se evidenció la gran cantidad de métodos existentes y que, a pesar de eso no se ha llegado a una respuesta universal para la planeación de trayectorias, esto debido a que existen infinitos estados en los que un móvil se puede encontrar y sería una tarea casi imposible preverlos todos. Para superar esto, se compararon algunos de los métodos más sencillos y utilizados debido a que los obstáculos iniciales fueron definidos de tal manera que se evitaran estancamientos o mínimos locales que son los problemas más comunes a los que se enfrentan estos algoritmos de planeación de trayectorias.

Al igual que los métodos de planeación de trayectorias, los métodos de evasión de obstáculos con inteligencia artificial son muy diversos, tanto así que no fue posible realizar una tabla comparativa con los diferentes métodos como si se hizo con el inventario de robots móviles de la Universidad

Militar Nueva Granada y los algoritmos de planeación de trayectorias. En este caso se escogieron dos métodos sobre los cuales se realizó un estudio exhaustivo para su implementación y se observó su desempeño. El algoritmo NEAT presentó problemas con diferentes tipos de obstáculos inmóviles por lo que no se hizo pruebas para obstáculos móviles. Estos problemas surgieron debido al poco control que se tiene durante el entrenamiento incluso utilizando aprendizaje curricular. El modelo empieza con condiciones aleatorias y su comportamiento se define según los estados que le supongan una idoneidad mayor sin importar si al final es un camino óptimo o no. Todos estos escenarios que parecen ser caóticos son los que deben minimizarse con dicha función de idoneidad por lo que el estudio de dicha función puede suponer incluso un trabajo futuro de envergadura similar a la de este mismo.

El algoritmo de DRL mostró resultados satisfactorios incluso para tipos de obstáculos no definidos durante el entrenamiento, los parámetros de la red y sus variables de actualización, aunque fueron más sencillos, fueron estrictamente elegidos debido los tiempos de entrenamiento que este requería teniendo en cuenta que al principio se pretendía implementar y entrenar las redes neuronales sin ningún tipo de aceleración por hardware. Nótese que al agregar tipos de obstáculos se forma implícitamente un tipo de aprendizaje curricular, por lo que se infiere que si se agregan más épocas de entrenamiento y más tipos de obstáculos esta técnica puede ser utilizada para propósitos más generales como el del transporte de utensilios en una oficina o tareas similares e incluso superar las pruebas fallidas realizadas.

8 BIBLIOGRAFÍA

[1] E. Masehian y D. Sedighizadeh, «Classic and Heuristic Approaches in Robot Motion Planning – A Chronological Review,» *International Journal of Mechanical, Industrial Science and Engineering*, vol. 1, nº 5, p. 7, 2007.

[2] Waymo, «Waymo,» Waymo LLC, 2019-2020. [En línea]. Available: https://waymo.com. [Último acceso: 19 10 2020].

[3] O. SOUISSI, R. BENATITALLAH, D. DUVIVIER, A. ARTIBA, N. BELANGER y P. FEYZEAU, «Path Planning: A 2013 Survey,» *IEEE-IESM*, p. 8, 2013.

[4] K. MaEek, I. Petrovic y N. Peric, «A reinforcement learning approach to obstacle avoidance of mobile robots,» *IEEE*, p. 5, 2020.

[5] «Robotis.,» 2020. [En línea]. Available: http://emanual.robotis.com/docs/en/edu/bioloid/premium/.

[6] «Génération Robots,» 2020. [En línea]. Available: https://www.generationrobots.com/en/.

[7] «Arduino,» Arduino, 2020. [En línea]. Available: https://store.arduino.cc/usa/arduino-robot. [Último acceso: 02 03 2020].

[8] «Lego,» LEGO System A/S, DK-7190 Billund, Dinamarca., 2020. [En línea]. Available: https://www.lego.com/es-es. [Último acceso: 20 02 2020].

[9] «FESTO,» Festo Didactic SE, 2020. [En línea]. Available: https://ip.festo-didactic.com/InfoPortal/Robotino/Overview/EN/index.html. [Último acceso: 20 02 2020].

[10] «iClebo Kobuki,» Kobuki, 2015. [En línea]. Available: http://kobuki.yujinrobot.com/about2/. [Último acceso: 21 02 2020].

[11] «ROS Components,» ROS Components, 2016. [En línea]. Available: https://www.roscomponents.com/es/robots-moviles/9-turtlebot-2.html#/3d_sensor-no/controlador-no/estacion_de_carga-no/tipo_de_cable-europa_cee_7_16/bateria_adicional-no/disco_extra-no/montaje-no/brazo_robotico-no/cursos-no/brazo_robotico_montaje_y_configur. [Último acceso: 2020 02 21].

[12] «Robotis e-manual,» Jekyll & Minimal Mistakes., 2020. [En línea]. Available: http://emanual.robotis.com/docs/en/platform/turtlebot3/features/#worlds-most-popular-ros-platform. [Último acceso: 2020 02 21].

[13] «ASUS,» ASUSTeK Computer Inc., 2020. [En línea]. Available: https://www.asus.com/es/3D-Sensor/Xtion_PRO_LIVE/. [Último acceso: 22 02 2020].

[14] «RASPBERRY PI,» RASPBERRY PI FOUNDATION, 2020. [En línea]. Available: https://www.raspberrypi.org/documentation/hardware/camera/. [Último acceso: 2020 02 22].

[15] «Intel RealSense,» Intel Corporation, 2020. [En línea]. Available: https://www.intelrealsense.com/lidar-camera-l515/. [Último acceso: 2020 02 22].

[16] Y. Pyo, C. Hancheol, L. Jung y D. Lim, ROS Robot Programming, 2017, p. 487.

[17] R. Smith, «Open Dynamic Engine,» Slashdot Media, [En línea]. Available: http://www.ode.org. [Último acceso: 29 06 2020].

[18] «Bullet Real-Time Physics Simulation,» WordPress, [En línea]. Available: https://pybullet.org/wordpress/. [Último acceso: 28 07 2020].

[19] M. Sherman y P. Eastman, «Simbody: Multibody Physics API,» Viewfarm, [En línea]. Available: https://simtk.org/projects/simbody. [Último acceso: 28 06 2020].

[20] «Dynamic Animation and Robotics Toolkit,» Georgia Tech and Carnegie Mellon University, 29 05 2020. [En línea]. Available: http://dartsim.github.io. [Último acceso: 29 06 2020].

[21] A. Agrawal, B. Aashay, A. Rohitkumar, A. T. Lima, J. Shuvrangshu y G. Debasish, «Accurate Prediction and Estimation of 3D-Repetitive-Trajectories,» *Indian institute of Science,* vol. 1, p. 12, 2020.

[22] C. Cáceres, J. M. Rosário y D. Amaya, «Approach of Kinematic Control for a Nonholonomic Wheeled Robot using Artificial Neural Networks and Genetic Algorithms,» *IEEE,* p. 6, 2017.

[23] «ROS documentation API,» Apache Server at docs.ros.org Port 80, 2020. [En línea]. Available: http://docs.ros.org/melodic/api/. [Último acceso: 2020 03 3].

[24] «Robotis e-manual,» Jekyll & Minimal Mistakes., 2020. [En línea]. Available: http://emanual.robotis.com/docs/en/dxl/ax/ax-12a. [Último acceso: 2020 03 02].

[25] H.-z. Zhuang, H. Li y S.-x. Du, «Real-tie Path Plannig of Mobile Robots in Dynamic Uncertain Environment,» *Proceedings of the 6th World Congress on Intelligent Control,* p. 6, 2006.

[26] G. Chen y j. Liu, «Hindawi,» 2019. [En línea]. Available: https://doi.org/10.1155/2019/1932812. [Último acceso: 25 03 2020].

[27] J.-C. Latombe y J. Barraquand, «Robot Motion Planning: A Distributed Representation,» *The International Journal of Robotics Research,* p. 24, 1991.

[28] J.-D. De Boissonat, A. Cérézo y J. Leblond, «Shortest paths of bounded curvature in the plane,» *Journal of Intelligent and Robotic Systems,* p. 11, 1994.

[29] H. B. LID Gang, G. J. Wang Hong y N. N. LI Yan, «A Route Planning Method Based on Improved Artificial Potential Field Algorithm,» *IEEE,* p. 5, 2011.

[30] D. Ferguson y A. Stentz, «Field D*: An Interpolation-based Path Planner and Replanner,» Carnegie Mellon University.

[31] I. Noreen, A. Khan y Z. Habib, «A Comparison of RRT, RRT* and RRT*-Smart Path Planning Algorithms,» *International Journal of Computer Science and Network Security*, vol. 16, nº 10, p. 8, 2016.

[32] University of Southern California, «Gazebo,» Open Source Robotics Foundation, 2014. [En línea]. Available: http://gazebosim.org. [Último acceso: 26 03 2020].

[33] A. Suwandi Ahmad y A. D. Wahyudi Sumari , «Cognitive Artificial Intelligence: Brain-Inspired Intelligent Computation in Artificial Intelligence,» *IEEE*, p. 7, 2017.

[34] CodeReclaimers, «NEAT-Python,» 2020. [En línea]. Available: https://neat-python.readthedocs.io/en/latest/. [Último acceso: 04 15 2020].

[35] K. O. S. a. R. Miikkulainen, «Evolving Neural Networks Through Augmenting Topologies.,» *IEEE*, vol. 10, nº 3, 2002.

[36] S. Grigorescu, B. Trasnea, T. Cocias y G. Macesanu, «A Survey of Deep Learning Techniques for Autonomous Driving,» *Journal of Field Robotics*, p. 39, 2019.

[37] B. Paden, M. ˇC´ap, S. Z. Yong, D. Yershov y E. Frazzoli, «A Survey of Motion Planning and Control Techniques for Self-Driving Urban Vehicles,» *IEEE*, vol. 1, nº 1, p. 23, 2016.

[38] G. Mester, «Obstacle Avoidance of Mobile Robots in Unknown Environments,» *5th International Symposium on Intelligent Systems and Informatics*, p. 5, 2007.

[39] OpenAI, «OpenAI,» 2020. [En línea]. Available: https://gym.openai.com/docs/.

[40] M. Volodymyr, K. Koray, D. Silver, A. Graves, I. Antonoglou, Wierstra y D. RIedmiller, «Playing Atari with Deep Reinforcement Learning.,» p. 9, 2013.

[41] Google, «TensorFlow,» 2020. [En línea]. Available: https://www.tensorflow.org. [Último acceso: 14 04 2020].

[42] B. d. podologiadeportiva, «Podologia Deportiva,» 2018. [En línea].

[43] Y. Bengio, J. Louradour, R. Collobert y J. Weston, «Curriculum Learning,» *Association for Computing Machinery*, p. 8, 2009.

[44] L. Haidong, L. Jiongcheng, G. Xiaoming , L. Binghao, L. Yuting y L. Xinglong, «Research on overfitting of deep learning,» *IEEE*, p. 4, 2019.