Reconocimiento de Espacios con Optimización de Trayectorias Utilizando el Robot "Turtlebot 3 Burger"

David Leonardo Martínez Herrera

Universidad Santo Tomás

División de Ingenierías y Arquitectura

Facultas de Ingeniería Electrónica

Tunja

2019

Reconocimiento de Espacios con Optimización de Trayectorias Utilizando el Robot "Turtlebot 3 Burger"

David Leonardo Martínez Herrera

Trabajo de grado para optar al título de

Ingeniero Electrónico

Director

Ing. Adolfo avila

Universidad Santo Tomás

División de Ingenierías y Arquitectura

Facultad de Ingeniería Electrónica

Tunja

2019

Nota de Aceptación:

_____

_____

_____

_____

_____

_____

_____

Firma jurada 1

_____

Firma jurada 2

Tunja, mayo 2019

Las ideas expresadas en este libro son responsabilidad exclusiva de los autores, por lo cual no comprometen a la Universidad Santo Tomas ni a la Facultad de Ingeniería Electrónica.

Dedicatoria

Agradezco a mi familia por apoyarme en cada instante de mi formación como profesional y como persona; también agradezco a compañeros de ingeniería con quienes con su ánimo me ayudaron a culminar este proyecto.

David L. Martínez H.

Agradecimientos

El autor expresa sus agradecimientos a:

Al Ingeniero Adolfo Ávila, por su colaboración, dedicación y compromiso en el desarrollo de este trabajo de grado.

Contenido

	Pág.
Glosario	16
Resumen	17
Introducción	19
1. Descripción del Problema	20
2. Justificación	22
3. Objetivo	24
3.1 General	24
3.2 Específicos	24
4. Marco de Referencial	25
4.1 Marco Teórico	25
4.1.1 Heurictic	33
4.2 Marco Conceptual	43
4.2.1 La raspberry pi 3 tipo b	49
4.2.2 Características de la raspberry pi 3 tipo b	50
4.3 Marco Histórico	52
4.3.1 Turtlebot1	54
4.3.2 Turtlebot2	54
4.3.3 Turtlebot3	55
4.4 Estado del Arte	56
4.4.1 Internacionales	56
4.4.2 Locales	57
5. Metodología	58
6. Resultados	60
6.1 Dimensiones del área del funcionamiento del robot	60

6.2 Conexiones del robot 62
6.3 Diagrama de flujo y nodos 65
6.4 Configuración y programación del robot 73
7. Prueba de Resultados y Discusión 85
7.1 Pruebas de funcionamiento 85
7.1.1 Prueba 1 85
7.1.2 Prueba 2 86
7.1.3 Prueba 3 88
7.1.4 Prueba 4 89
7.1.5 Prueba 5 91
7.1.6. Prueba 6 93
7.1.7 Prueba 7 95
8. Costos 100
Conclusiones 101
Recomendaciones 102
Referencias Bibliográficas 104
Anexos 107

Lista de Figuras

Figura 1. Logo de ROS. Fuente. Ros. (Works on Arm, s.f.) 25

Figura 2. Grafo del Algoritmo dijkstra. Fuente. Dijkstra's shortest path algorithm | Greedy Algo-7. (GeeksforGeeks, s.f.). 26

Figura 3.Menor ruta mediante el uso del algoritmo dijkstra. Fuente. Algorithms. (Introduction to A*, 1997). 27

Figura 4. Ruta generada por el algoritmo Greedy Best-First-Search. Fuente. Algorithms. (Introduction to A*, 1997). 28

Figura 5. Trazo de una línea utilizando el algoritmo Bresenhall. Fuente. Local planning for a mobile robot (Koz´ak, 2013). 29

Figura 6. Análisis de los valore binarios en un mapa. (Parungao, Hein, & Lim, 2018). 30

Figura 7. Ejemplo del Algoritmo A*. m. Dijkstra's shortest path algorithm | Greedy Algo-7. (GeeksforGeeks, s.f.). 31

Figura 8. Trazo de una ruta con un obstáculo. Fuente. Algorithms. (Introduction to A*, 1997). 31

Figura 9.Trazo de trayectoria con el algoritmo A-star(A*). Fuente. Algorithms. (Introduction to A*, 1997). 32

Figura 10.Ejemplo aplicando el algoritmo A* con el método Manhattan. (Brilliant, 2019) 34

Figura 11. Ruta mediante el método A* manhattan. Fuente. Algorithms. (Introduction to A*, 1997). 35

Figura 12. Ruta mediante el metodo A* Euclidean. Fuente. Algorithms. (Introduction to A*, 1997). 35

Figura 13. Ruta mediante el método A* Diagonal Distance. (Introduction to A*, 1997). 36

Figura 14. Ruta mediante el método A* Euclidean and breaking-ties. Fuente. Algorithms. (Introduction to A*, 1997). 37

Figura 15. Ruta mediante el método A* "Euclidean" and "breaking-ties" con obstáculos. Fuente. Algorithms. (Introduction to A*, 1997). 37

Figura 16.Trazo de línea usando producto cruz y método "breaking ties" para obtener una línea recta entre dos puntos. Fuente. Algorithms. (Introduction to A*, 1997). 39

Figura 17. Generación de nodos aproximadamente 2000 nodos (derecha) y generación de 5 nodos(izquierda). Fuente. Rapidly Exploring Random Tree (RRT) and RRT* (Huang, 2018). 40

Figura 18. Diferentes rutas al variar el valor de velocidad lineal y angular. Fuente. Local planning for a mobile robot (Koz´ak, 2013). 42

Figura 19. Sistema operativo Ubuntu MATE 16.04.2 LTS para raspberry pi. Fuente. How to setup and install ubuntu mate 16.04.2 lts on raspberry pi 3 or 2. (Frost, 2017). 43

Figura 20. Imagen de inicio después de instalarse el sistema operativo en la raspberry pi. (Raspberry Pi, 2013) 44

Figura 21. Robot turtlebot3 burger. Tomado de emanual robotics. Fuente. TurtleBot 3 Burger. (Robots Components, 2016) 46

Figura 22. Circuito de comunicación Half dúplex Uart de un motor dynamixel. Fuente. Manual Vl 29.00 (Robotis, 2010) 47

Figura 23. Sensor de distancia laser LDS-01(LIDAR). Fuente. 360 Laser Distance Sensor LDS-01 (LIDAR). Fuente. 360 Laser Distance Sensor LDS-01 (LIDAR) (Robotis, 2019) 47

Figura 24.Tarjeta embebida OpenCR. Fuente. Manual Vl 31.30.00 (Robotis, 2010) 48

Figura 25. Botones normalmente abiertos para la reactivación del frameware o aplicaciones. Fuente. Manual Vl 31.30.00 (Robotis, 2010) 49

Figura 26. Logo de la compañía Willian Garage. Fuente. Willow Garage. (willow Garage, 2015) 52

Figura 27. Logo de la versión Kinetic Kame ROS para utilizarse con el robot turtlebot3 burger. Fuente. Willow Garage. (willow Garage, 2015). 54

Figura 28. iRobot y Turtlebot1 elaborado en Willow Garage. Fuente. Willow Garage. (willow Garage, 2015) 54

Figura 29 Turtlebot2 (izquierda) y Sensor Kinect(derecha). Fuente. TurtleBot 2 (Generation Robots, 2015) 55

Figura 30. Turtlebot 3 Burger(Izquierda) , Turtlebot 3 Wiffle(medio) y Turtlebot 3 Wiffle Pi (derecha). Fuente. TurtleBot 2 (Generation Robots, 2015) 55

Figura 31. Área definida del robot de forma hexagonal. Fuente. Autor 60

Figura 32. Montaje del área hexagonal. Fuente. Autor 61

Figura 33. SLAM utilizando el robot Turtlebot3 burger. Fuente. Tomado de Rviz. 61

Figura 34. Conexiones físicas del robot turtlebot3 burger. Fuente. Universidad Santo Tomás, Tunja. 62

Figura 35. Batería recargable y cargador de batera del robot "turtlebot3 burger". Fuente. Universidad Santo Tomás, Tunja. 63

Figura 36. Diagrama de conexión del robot "turtlebot3 Burger" con un computador de forma remota. Fuente. Autor, basado en Manual de Turtlebot3 Burger. 64

Figura 37. Diagrama de flujo para funcionamiento del robot ¨Turtlebot3 Burger¨. Fuente. Autor. 65

Figura 38. Gráfica de Nodo. Fuente .Tomado de ROS. 67

Figura 39. Nodo /amcl. Fuente: Tomado de Ubuntu 16.4. Fuente. Autor. 68

Figura 40. Nodo /move_base. Fuente: Tomado de Ubuntu 16.4. Fuente. Autor. 69

Figura 41. Nodo /turtlebot3_slam_gmapping. Fuente. Autor. 70

Figura 42. Nodo /explore_server. Fuente. Autor. 70

Figura 43. Nodo /turtlebot3_core. Fuente. Autor. 71

Figura 44. Archivo frontierexploration.launch. Fuente. Autor. 72

Figura 45. Archivo Launch perteneciente a la programación en el robot turtlebot3. Fuente. Autor. 76

Figura 46. Programación de la comunicación con la tarjeta openCR y la tarjeta raspberrypi 3. 77

Figura 47. Programación en la rasberry pi para unir el sensor lds con el puerto USB. Tomado de Ubuntu Fuente. Autor. 77

Figura 48. Nodo de unión entre la Raspberry pi y el sensor lds junto con un ordenador. 79

Figura 49. Cargar y programar mediante el ejecutable de Ubuntu. Tomado de Ubuntu 16.4. 80

Figura 50. Lectura de los paquetes creados y relacionados con la raspberry pi utilizando comandos de ROS en el ejecutable de Ubuntu. Fuente. Autor. 81

Figura 51. Resultado aplicando el algoritmo A-star(A*) junto con dijktras. Fuente: Autor. 85

Figura 52. El robot turtlebot3 llega al obstáculo en la posición final. Fuente. Tomada de Ubuntu 16.4. 86

Figura 53. Resultado de la prueba 2. Realizando SLAM (izquierda) y en pista física (derecha). Fuente. Autor. 86

Figura 54. Trayectoria roja que sigue el robot turtlebot3 burger. Fuente. Autor. 87

Figura 55. Resultado de la prueba 3. Pista física (izquierda) y realizando SLAM(derecha).Fuente. Autor. 88

Figura 56. El robot turtlebot3 llega al obstáculo siguiendo la trayectoria más corta. Fuente. Autor. 89

Figura 57. Resultado de la prueba 4. Ruta corta de color rojo para llegar a la cruz (izquierda) y ruta más corta con SLAM (derecha).Fuente: Tomado de Ubuntu 16.4. 90

Figura 58. El robot turtlebot3 llega a la cruz roja siguiendo la trayectoria más corta. Fuente. Autor. 90

Figura 59. Resultado de la prueba 5. Obstáculo en la ruta más corta. Fuente. Autor. 91

Figura 60. Recorrido del robot turtlebot3 a través segunda ruta más corta utilizando SLAM. Fuente. Autor. 92

Figura 61. Resultado de la prueba 5 aplicando el algoritmo A-star(A*) junto con dijktras con obstáculos. Fuente. Autor. 93

Figura 62. El robot realiza una rotación para cambiar su sentido. Fuente. Autor. 94

Figura 63. El robot turtlebot3 llega a la cruz roja siguiendo la trayectoria más corta en zig-zag. Fuente. Autor. 95

Figura 64. Vista en tres dimensiones del terreno hexagonal con obstáculos (derecha) y la pista en físico(izquierda). Fuente. Autor. 96

Figura 65. Tipos de vista que permite el programa rviz. Fuente. Autor. 96

Figura 66. Modelo entre un análisis teoría (izquierda) y practico utilizando el algoritmo A* (derecha). Fuente. Autor. 97

Figura 67. Trazo de ruta con una vista en 3D. Fuente. Autor. 98

Figura 68. Trayectoria del robot turttlebot3 burger para la prueba 7. Fuente. Autor. 99

Lista de Tablas

pág.

Tabla 1. *Actividades realizadas para definir el proceso metodológico del proyecto.* 59

Tabla 2. *Costos generales del proyecto*. 100

Lista de Anexos

	Pág.
Anexo A. *Código de Plano Local*	107
Anexo B. *Código Rviz*	110

Glosario

Actuador: Dispositivo electromecánico que actúa sobre el medio exterior, convierten una magnitud eléctrica en otra de otro tipo (mecánica, térmica, etc.), mediante un proceso inverso al de los sensores. Los actuadores pueden mantener niveles de salida continuos o discretos. Entre el controlador y los actuadores están las interfaces, que no son otra cosa que acondicionadores que adaptan la señal a la entrada del actuador. (Romero, Vázques, & de Castro, 2010)

Algoritmo: Es un método para resolver un problema, el cual tiene una serie de procesos gráficos, numéricos o metodológicos que se enfoca en manejar variables tanto lógicas como físicas. Estos procesos tienen un orden lógico que permite dividir el resultado final en una serie de pequeños resultados o pasos que llevan a cumplir con el objetivo planteado.

Quaternio: En robótica quaternio es definido como la rotación sobre alguno de los ejes x, y o z en coordenadas cartesianas. En el caso del robot con dos ruedas, quaternio es la rotación en el eje z para permitir tener orientación del robot hacia un objetivo final.

Robot: Robot es definido como dispositivo con unidades eléctricas, electrónicas y mecánicas el cual permite ser programado y auto-controlado para realizar acciones autónomas o por medio de un usuario.

Sensor: Dispositivo capaz de transformar magnitudes físicas o químicas (llamadas variables de instrumentación) en magnitudes eléctricas. Las variables de instrumentación pueden ser, por ejemplo: temperatura, intensidad lumínica, distancia, aceleración, inclinación, desplazamiento, presión, fuerza, torsión, humedad, PH, etc. Una magnitud eléctrica puede ser una resistencia eléctrica (como en una RTD), una capacidad eléctrica (como en un sensor de humedad), una tensión eléctrica (como en un termopar), una corriente eléctrica (como en un fototransistor), etc. (Quintana, 2016)

Resumen

En la actualidad existen nuevos prototipos que permiten ser utilizados para adquirir un amplio conocimiento en el campo de la electrónica y la robótica. Uno de estos prototipos es llamado "TURTLEBOT", el cual es posible realizar investigación sobre el reconocimiento y optimización de trayectorias. Este robot es caracterizado por ser un robot en dos ruedas, diseñado para realizar navegaciones, desplazamientos y rutas sobre áreas planas.

Se planteó programar un robot (robot para superficie en dos dimensiones) TURTLEBOT3 BURGER con el objetivo de aplicarse un algoritmo que optimice la ruta, con el fin de adquirir y aplicar conocimiento sobre el uso de esta plataforma en virtud de nuevos y futuros desarrollos. En base se define un área hexagonal adecuada de recorrido para el robot, colocando varios obstáculos dentro de este terreno que permitiera comprobar la programación y ejecución del algoritmo programado sobre la plataforma "TURTLEBOT3 BURGER".

En base a los objetivos planteados se analizó e investigo sobre el funcionamiento, características y previos proyectos realizados con la plataforma "TURTLEBOT3 BURGER". Además, se investigó el manejo y uso de cada uno de los componentes que conforma la plataforma "TURTLEBOT3 BURGER" necesarios para su adecuado funcionamiento. Una vez hecho esto se realizó la comprobación del funcionamiento de cada componente del robot.

Programando el robot TURTLEBOT3 BURGER se realizaron siete pruebas de funcionamiento dentro de una pista de forma hexagonal. Cada prueba tenía un nivel de complejidad cada vez mayor lo cual permitía adquirir resultados y realizarse mejoras en la programación del robot "TURTLEBOT3 BURGER".

En los resultados se abordan modos de conexión, uso de robot turtlebot3 burger, modo programación junto con una serie de pruebas que se enfocan en aplicar un algoritmo llamado "dijkstra y Astar" que permita optimizar la rutas con un robot de dos rudas y visualizarse en un

programa llamado “Rviz”

Finalmente se comprobó la parte teórica con la parte práctica adquiriendo resultados y cumpliendo los objetivos planteados inicialmente utilizando el robot “TURTLEBOT3 BURGER”.

Introducción

La importancia de estudiar la optimización de trayectorias permite avanzar en investigaciones sobre decisiones que tomaría un robot de forma autónoma para trasladarse de un terreno con muchos obstáculos y finalmente llegar hacia su objetivo final. Además, permite conocer resultados en la utilización de algoritmos que se asocien al continuo funcionamiento de robots que solucione la optimización de trayectorias que puedan cumplir su objetivo en el menor tiempo posible.

En la actualidad se está estudiando automóviles eléctricos autónomos que tomen decisiones dentro de una ciudad con obstáculos. Estos automóviles eléctricos realizan un continuo reconocimiento del lugar donde transita, analizando los obstáculos durante su trayecto. Para su funcionamiento define su destino y la ruta de menor longitud y mayor facilidad de recorrido.

En la actualidad existen brazos robóticos pertenecientes a la compañía ABB que realizan tareas autónomas y repetitivas, en las cuales calculan su recorrido de desplazamiento para tener adecuada movilidad en la toma de decisiones que requiere mayor precisión y menor tiempo de funcionamiento. Este robot calcula su ubicación inicial junto con el recorrido que debe realizar para llegar a su objetivo final.

1. Descripción del Problema

La robótica es un tema que ha estado en desarrollo y permite su amplio estudio y análisis, sin embargo, los robots requieren ser programados con algoritmos que permita definir una trayectoria dentro de un espacio.

El manejo de paquetes del sistema de operación robótica (ROS)permite mejorar la efectividad en el desarrollo de proyectos en robótica; sin embargo, los estudios y aplicaciones de esta plataforma en robótica aun están en desarrollo y requiere de programación y conocimiento de esta plataforma en robótica para la generación de paquetes que cumplan los objetivos planteados.

El conocimiento de la plataforma ROS puede permitir muchas aplicaciones para diferentes tipos de robots que pueden ser ensamblados y programados en un plano en 2D o 3D.

La optimización de trayectorias es un tema de investigación en el campo de la ingeniería electrónica, mecatrónica y mecánica que ha venido en desarrollo durante los últimos años para aplicaciones automovilísticas e industriales; sin embargo, estas investigaciones y conocimientos sobre previos avances y nuevos desarrollos siguen en curso para futuros desarrollos y nuevos robots.

Como proyecto de investigación podemos aplicar la siguiente pregunta:

¿Cómo aplicar un algoritmo a un robot que permita optimizar una trayectoria dentro de un terreno plano con obstáculos?

Adicionalmente podemos plantear unas preguntas específicas:

¿Cómo realizar un mapeo y navegación de un terreno hexagonal utilizando el robot "TURTLEBOT 3 BURGER" para obtener su ubicación y localización continua?

¿Qué ventajas permite el manejo de los paquetes de ROS para el manejo y desarrollo de trayectorias utilizando dos motores Dynamixel XL430-W250-T?

¿Qué técnicas de optimización pueden ser utilizadas para detectar obstáculos y generar trayectorias hacia al punto final definido utilizando el robot “Turtlebot3”?

¿Qué ventajas permitiría utilizar un sensor de distancia (LDS) con movimiento a 360 grados en comparación con sensores ultrasónicos tradicionales?

2. Justificación

La robótica a través de los últimos años se ha convertido en un tema de investigación que ha permitido tener participación en aplicaciones industriales, ambientales, sociales y académicos. Dentro de las cuales existe diferentes clases de plataformas, circuitos eléctricos, códigos electrónicos entre otros nuevos desarrollos eléctricos y mecánicos han sido creados como avances tecnológico para ser utilizados por personas con deseos de aprendizaje y control de estos dispositivos; este proyecto busca incorporar conocimientos en el área de la robótica con capacidad de entender las bases que confirma la lógica que se vinculan plataformas avanzadas creadas en los últimos años basándonos en algoritmos para la realización de tareas específicas.

Un campo importante de aplicación de la robótica en el sector de la automatización industrial. En este campo la manipulación robótica implicara que se desplacen piezas y herramientas en el espacio mediante algún tipo de mecanismo. Las aplicaciones de la manipulación robótica y definición de trayectorias son ampliamente utilizadas en robots industriales para ensamblajes o manipulador mecánico. Por esta razón el desarrollo del movimiento a través del espacio y generación de trayectorias cartesianas son parte importante dentro del estudio de la robótica y sus aplicaciones industriales.

Es importante el conocimiento de la robótica porque permite plantear nuevos proyectos a niveles académicos, sociales e industriales utilizando robots; estos proyectos pueden ser realizados partiendo de experiencias adquiridas con plataformas robóticas, mediante el desarrollo de códigos de programación que sean entendidos ejecutados en sistemas embebidos para alcanzar objetivos diseñados y plateados.

El robot “TURTLEBOT 3 BURGER” es unas plataforma robótica con capacidades para tener aplicaciones con paquetes del software (ROS) y alcanzar objetivos en dos dimensiones

porque cuenta con tarjetas tales como: Raspberry py 3 modelo B que permite ser configurado con un sistema operativo UBUNTU Mate , tarjeta de control robótico(OpenCR) con procesador ARM Cortex- M7 con puertos TTL además de dos motores Dynamixel de ultimo desarrollo en la robótica, los cuales pueden ser utilizados y programados para alcanzar objetivos planteados en el campo de la robótica.

3. Objetivo

3.1 General

Aplicar sobre sobre la plataforma “TURTLEBOT 3 BURGER” los algoritmos de Dijkstra y A-star(A*) para optimizar trayectorias de menor recorrido dentro de una plataforma hexagonal plana y con obstáculos.

3.2 Específicos

– Determinar las características de hardware del robot” TURTLEBOT 3 BURGER”, sensores, tarjeta de procesamiento y actuadores.

– Reconocer las características que permite ROS como herramienta para el desarrollo de proyectos con robots.

– Comparar métodos para conducir un robot desde un punto inicial hasta un punto final reconociendo su trayectoria mediante un continuo mapeo y monitoreo del robot.

– Utilizar el método de “aproximaciones dinámicas de Windows” sobre la plataforma “TURTLEBOT3 Burger” que permita guiar el robot sobre una ruta a la cual se acerque a la trayectoria optimizada por los algoritmos dijstra y A-star.

– Realizar la programación sobre las tarjetas de procesamiento del robot con los algoritmos Dijkstra y A-star.

4. Marco de Referencial

4.1 Marco Teórico

En este capítulo se pretende mostrar algunos de los referentes teóricos más relevantes para el desarrollo del presente proyecto, tal como ROS que se define como un sistema operativo flexible con aplicaciones en el control y manejo de robots a nivel académico e industrial. ROS actúa como puente entre desarrollo de aplicaciones en la robótica y el manejo dentro de un sistema operativo para ser emulado como máquina virtual. (Trapi, 2017).

Figura 1. Logo de ROS. Fuente. Ros. *(Works on Arm, s.f.)*

ROS promueve librerías y herramientas para desarrolladores de software para crear aplicaciones en robots. ROS es una plataforma completamente libre para cambiar, utilizar y comercializar alrededor del mundo. El objetivo es primordial es permitir a desarrolladores de software construir más aplicaciones de robots rápidamente y fácilmente sobre una plataforma común. (Garage, 2019)

Los algoritmos de generación de ruta se conciben principalmente como herramientas matemáticas que permiten ser aplicadas en video juegos, transmisión de paquetes o en la realización de rutas con GPS.

Los sistemas de operación para robot ROS contiene métodos de parametrización e interpolación que son: “Standard behavior”,”grid path”,”simple potencial calculation”,”A*

Path",Dijkstra's, "A*", "Old Navfn Behavior". (Open Source Robotics Foundation). Cada método podría ser aplicado en distintos tipos de robots según sea su función y diseño.

El sistema operativo ROS permite la gestión y el control de cualquier tipo de robot de manera manual o remota gracias a un protocolo de comunicación http que comunica el robot con un computador de manera remota.

Los algoritmos A* search y Dijkstra son capaces de detectar caminos más cortos y optimizados entre nodos o celdas el cual es posible desplazase. Los algoritmos pueden aplicarse en plataformas roboticas que requiere un desplazamiento en dos o en tres dimensiones. (Brillian Math and science, 2018)

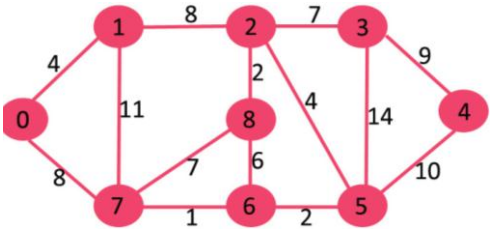

Figura 2. Grafo del Algoritmo dijkstra. Fuente. Dijkstra's shortest path algorithm | Greedy Algo-7. *(GeeksforGeeks, s.f.).*

El algoritmo de Dijkstra busca el camino más corto y permite calcular caminos de longitud mínima entre dos o más vértices. El algoritmo analiza el valor de los nodos o celdas y de la distancia de pequeñas trayectorias que requiere para llegar a la ruta final. El grafo determina dentro de un mapa la cantidad de nodos o celdas posibles que puede necesitar para conectarse y llegar a su objetivo final.

El algoritmo permite determinar el menor esfuerzo o el costo más bajo entre un punto de inicio a un punto final. Este algoritmo usa nodos o celdas como punto final para realizar el cálculo

de distancia y finalmente calcular el menor esfuerzo que tomaría el robot para llegar a su destino.

Cuando el robot no ha realizado una exploración del terreno toma este valor como infinito permitiendo avanzar por donde el robot conoce la ruta hasta conocer si existe posibilidad de avanzar o no.

El algoritmo define una trayectoria a través de los nodos o celdas en el cual el robot tomara una decisión para avanzar por el mejor trayecto.

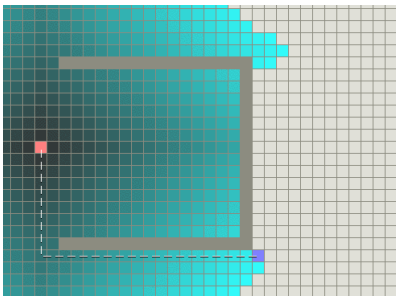

*Figura 3.*Menor ruta mediante el uso del algoritmo dijkstra. Fuente. Algorithms. *(Introduction to A*, 1997).*

En la figura 3 vemos que la celda de color rojo es el punto de partida desde donde se va a trazar una trayectoria y la celda de color azul es el punto final de la trayectoria. Las celdas de color gris ósculo define los obstáculos y la línea descontinua define el trayecto aplicándose el algoritmo Dijkstra. El color negro oscuro define la zona cercana al punto de partida y las celdas de color azul turquesa definen las zonas cercanas al punto de llegada.

El algoritmo Greedy Best-First-Search en comparación con el algoritmo dijkstr generara una ruta que acercaría al punto final pero no podrá seguir por el obstáculo cóncavo que impediría su generación de ruta.

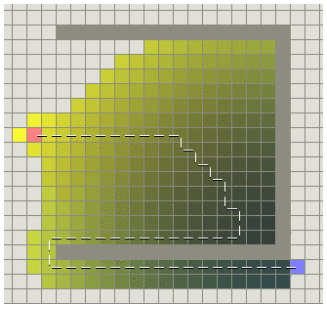

Figura 4. Ruta generada por el algoritmo Greedy Best-First-Search. Fuente. Algorithms. *(Introduction to A*, 1997).*

En la figura 4 vemos que la celda de color naranja es el punto de partida del trazo de una trayectoria y la celda de color azul es el punto de llegada de la trayectoria. Las celdas de color gris definen el obstáculo y las líneas descontinuas define la trayectoria aplicando el algoritmo "Greedy Best-first-Search". Las celdas de color amarillo definen el espacio cercano al punto anaranjado y las celdas de color negro definen el espacio cercano a la celda de color azul. El cambio de color de las celdas define que tan cercano o tan lejos esta del objetivo.

Cuando el robot ocupa una celda dentro del mapa cuando está en movimiento, el valor de esas celdas será tomado como transitada por el robot, haciendo que esta celda no permanezca constante dentro de la ruta trazada para el robot. (Amits Thoughts on pathfinding, 2019)

El trazo de las líneas entre dos puntos será generado mediante el algoritmo "Bresenhal"; el cual permite tener una posibilidad de generar una línea recta de manera digital entre dos puntos calculando su pendiente mediante las coordenadas de los puntos. Cuando se establecen una serie de puntos en una trayectoria que parte de un punto inicial a un punto final, este algoritmo une cada punto para finalmente tener el trazo total de la ruta que el robot seguirá. (Kozak, 2013).

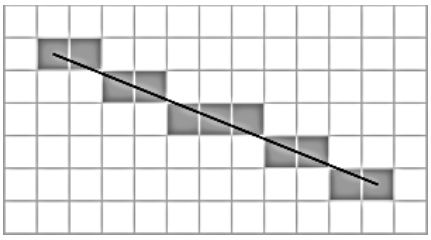

Figura 5. Trazo de una línea utilizando el algoritmo Bresenhall. Fuente. Local planning for a mobile robot *(Koz´ak, 2013)*.

En la figura 5 vemos celdas de color blanco y de color gris. Las celdas de color gris son las celdas que es posible ocupar para trazar una línea aplicando el algoritmo" Bresenhall". Este algoritmo realiza trayectos ocupando celdas dentro de un espacio, la cual después de calcular la mejor ruta utilizando el algoritmo A-star y Disjstra , realiza la trayectoria aplicando el algoritmo "Bresenhall".

El algoritmo dijkstra tendrá que tomar solo el valor g (n) donde n es el valor del costo de la celda o vértice lo cual el valor heuristic será cero (h(n)=0) para este algoritmo. Esto permitirá realizar la suma de las distancias entre nodos o vértices, permitiendo realizar la suma de las posibles rutas y tomar la que tenga el menor valor final desde el punto inicial al punto final.

En la rejilla la matriz en dos dimensiones toma el valor de cero para la celda ocupada y uno para las celdas ocupadas. (Parungao, Hein, & Lim, DIJKSTRA ALGORITHM BASED INTELLIGENT PATH PLANNING WITH TOPOLOGICAL MAP AND WIRELESS COMMUNICATION, 2018).

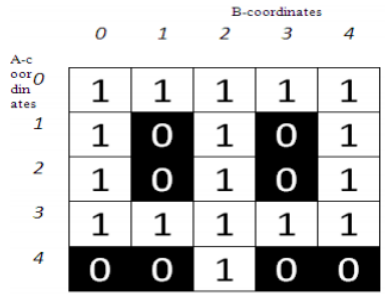

Figura 6. Análisis de los valore binarios en un mapa. *(Parungao, Hein, & Lim, 2018).*

En la figura 6 vemos celdas de color negro y color blanco. Las celdas de color negro son definidas como obstáculos asignándole un valor binario de cero a cada una de estas. Mientras que las celdas de valor binario unos representan los espacios por donde el robot puede transitar.

Este algoritmo tiene como base el análisis del mapa generado y visualizado por el programa Rviz a partir de los datos obtenidos con el robot como punto central.

El robot buscara seguir la ruta que es trazada por el algoritmo Dijkstra haciendo una rotación sobre el eje z, mediante el cual una variable w que permitirá decidir cuanta rotación necesitara el robot para buscar una posición nueva que mejore el desplazamiento por el mejor camino trazado.

El algoritmo Dijkstra permite analizar el costo del nodo o celda según la posición se encuentre hasta con respecto al punto final. Con esto no solo sería la ruta más corta si no que busca llegar el punto final con mayor rapidez a través de nodos o celdas que pueda ser ocupada y que se dirijan hacia su objetivo.

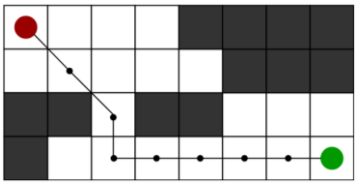

Figura 7. Ejemplo del Algoritmo A*. m. Dijkstra's shortest path algorithm | Greedy Algo-7. *(GeeksforGeeks, s.f.).*

En la figura 7 vemos como el algoritmo A* realiza el menor trayecto posible desde el punto de partida hasta el punto de llegada.

El algoritmo A* traza una línea diagonal entre celdas y llegar a otras celdas buscando ocupar celdas por las que pueda seguir trazando la ruta. (Patel, 2019)

El algoritmo A* analiza los espacios no ocupados, la posición del objetivo final y la posición inicial desde donde se trazará la ruta. El algoritmo busca llegar a su objetivo final trazando la ruta por donde las celdas no estén ocupadas. El conocimiento del terreno ayuda a que el algoritmo pueda trazar una trayectoria óptima lo cual sin esto se generarían errores en la generación de la ruta.

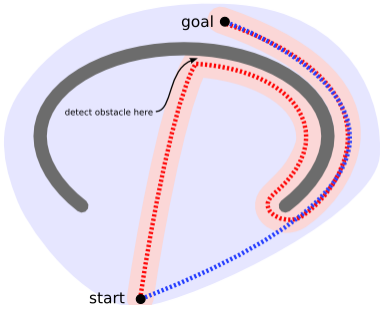

Figura 8. Trazo de una ruta con un obstáculo. Fuente. Algorithms. *(Introduction to A*, 1997).*

En la figura 8 vemos dos posibles trayectos. En el trayecto rojo representa el recorrido que

haría el robot si no se aplica el algoritmo A*, mientras que en el trayecto azul se puede ver como un robot realiza un menor recorrido respecto al trayecto rojo para llegar a la meta.

La ruta roja es generada porque no es conocido el terreno por completo. La ruta azul es el trayecto óptimo y más corto que es realizado después de tener conocimiento por completo del terreno. (Amits Thoughts on pathfinding, 2019)

El algoritmo A* es como el algoritmo dijkstra porque calcula el camino más cercano. También el algoritmo A* es como el algoritmo Greedy best-serch porque puede manejar heuristic para guiarse, es decir buscando acercarse a su punto final como su mayor prioridad sin analizar la distancia en su recorrido. Además el algoritmo A* search es un algoritmo con características del algoritmo breadth-first search (BFS) el cual analiza el nodo inicial, seguido de los nodos adyacentes o vértices permitiendo ver los nodos o celdas visitadas y mostrada dentro de un árbol de nodos o celdas los nodos a su alrededor que más tiene alcance que los nodos cercanos. A diferencia del algoritmo dijkstra tomar la ruta más rápida para llegar a la meta.

El algoritmo A* une el algoritmo dijkstra(favoreciendo el recorrido cercano de los vértices más cercanos al punto de partida) y el algoritmo Greedy best-search (favoreciendo los vértices más cercanos al punto final). (Amits Thoughts on pathfinding, 2019)

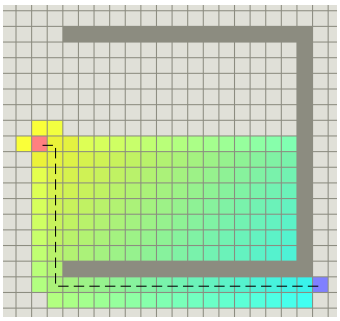

*Figura 9.*Trazo de trayectoria con el algoritmo A-star(A*). Fuente. Algorithms. *(Introduction to A*,*

1997).

En la figura 9 podemos ver la trayectoria utilizando el algoritmo A*, el cual analiza y detecta el límite del obstáculo, de tal forma que realiza un recorrido de forma vertical para luego desplazarse de modo horizontal hacia su meta.

El algoritmo A-star analiza los *costos exactos* como un valor g(n) de los vértices cercanos al punto de inicio a algún vértice n(n es definido como vértice de alguna celda), y el valor de h(n) como el *costo estimado* "heurictic" desde un vértice n hasta el punto final. h son los vértices del color(amarillos) cercano al punto de inicio y lejos del punto final y g son los vértices del color azul claro cercano al punto final y lejos del punto de inicio. El algoritmo A-star hace un balance dándole un valor o costo a los vértices de cada celda del mapa el cual varía dependiendo de qué tan cerca o lejos este del punto de inicio o final. La función total es la adición f(n) = g(n) + h(n) , equivalente al balance final del costo de los vértices. Cuando la celda es tomada por un obstáculo el valor no va tener un resultado balanceado lo que tendría que generar la toma de otros vértices de las demás celdas para ser corto su recorrido. (Patel, 2019)

El uso de "heuristic" dentro del algoritmo A* es importante porque permite analizar el valor h(n) y generar la trayectoria más corta. El valor h(n) debe ser menor o igual al valor del costo del vértice n en la celda. Si el valor de h(n) es mayor a g(n) aplicaría más el algoritmo Greedy best-search lo cual no encontraría la ruta más corta.

4.1.1 "Heuristic"

"Heuristic" es definido como una función h(n) que permite adquirir un costo estimado entre un vértice n de una celda o nodo de una rejilla de dos dimensiones y el punto final al que llegara el robot. El costo dependerá de la cercanía y posición del vértice que tenga una celda con respecto al

punto final. Cuando un obstáculo ocupa una celda o nodo el costo será muy grande lo que genera abortar la posibilidad de trazar una ruta por esos nodos(s) o celda(s).

Existe tres distintas clases de heuristic para el trazo de ruta. Estos son: *"the manhattan distance heuristic*", "*the Euclidean Distance Heuristic*" y "*diagonal distanc*e".

El método "manhattan distance heuristic" es utilizado en inteligencia artificial como un método computarizado capaz de calcular el número de celdas que se encuentran tanto vertical como horizontal hasta llegar a la celda de destino. Esto analiza desde la celda de inicio y luego las celdas adyacentes siguiendo la cual el valor de la función f sea menor.

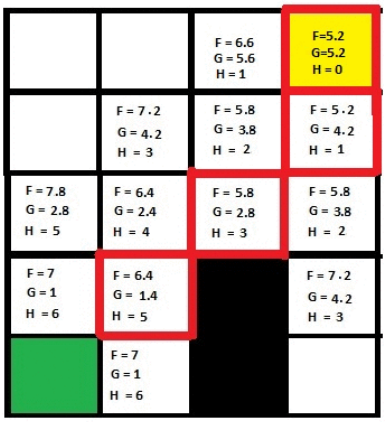

Figura 10.Ejemplo aplicando el algoritmo A con el método Manhattan. (Brilliant, 2019)*

En la figura 10 vemos cada celda de color blanco con el valor de g(n), h(n) y f(n).El valor de g(n) es el valor que va aumentando en las celdas a medida que va alejándose del punto de partida. El valor h el valor de distancia heurístico manhattan que decremento a medida va acercándose a la celda final. El cálculo de h para el método manhattan es dado en la ecuación 12. (Brillian Math and science, 2018)

$$h = |Xstar - Xdestination| + |Ystar - Ydestination| \quad (12)$$

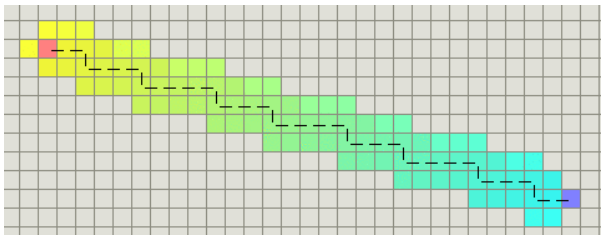

Figura 11. Ruta mediante el método A* manhattan. Fuente. Algorithms. *(Introduction to A*, 1997).*

En la figura 11 vemos que se traza una ruta aplicando el algoritmo A* mediante el método "manhattan", el cual lee los valores de g y h de las celdas alrededor de la celda de color naranja y la celda de color azul.

The "Euclidean Distance Heuristic" es un método más exacto que el método manhattan. Este método calcula la distancia entre dos puntos lo cual es lento por el cálculo para largas rutas. El cálculo de h para el método Euclidiano es dado en la ecuación 13.

$$h = \sqrt{(Xstar - Xdestination)^2 + (Ystar - Ydestination)^2} \quad (13)$$

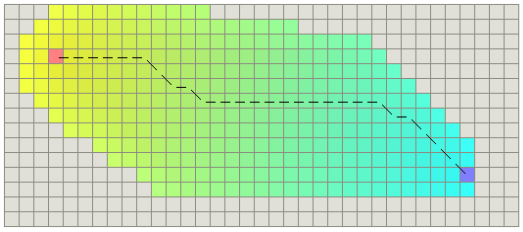

Figura 12. Ruta mediante el metodo A* Euclidean. Fuente. Algorithms. *(Introduction to A*, 1997).*

En la figura 12 vemos dos puntos en la cual se traza una trayectoria utilizando el método A*

"Euclidean Distance Heuristic". La trayectoria se realiza de forma horizontal con acercamientos de forma diagonal.

El método "diagonal distance" es parecido al método "manhattan", pero suma el producto del minio valor para calcular la diagonal entre la celda y la diferencia entre una distancia D1 y D2 que permite escalar el costo final para alcanzar la celda. El cálculo matemático es definido en la ecuación 14. (Amits Thoughts on pathfinding, 2019)

$$h = |Xstar - Xdestination| + |Ystar - Ydestination| + (D2 - 2 * D) * \min(dx, dy) \quad (14)$$

Donde el valor de D1y D2 son los costos en el análisis del paso entre las celdas, y min (dx,dy) como el valor de los pasos entre celdas.

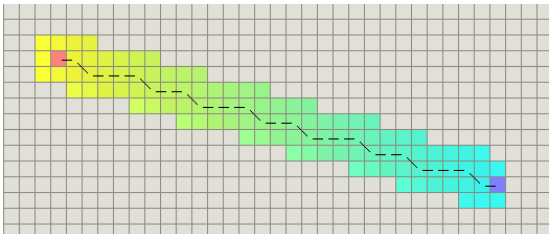

Figura 13. Ruta mediante el método A* Diagonal Distance. *(Introduction to A*, 1997).*

En la figura 13 se puede observar el trazo de ruta al aplicarse el algoritmo A* con el método "Diagonal Distance", el cual se aprecia un trayecto que contiene muchas variaciones, atravesando las celdas de forma diagonal.

Existe además una estrategia que al realizar el cálculo para variar el valor de f , variando h a medida que avanza desde punto inicial al su punto final. Para esto se adiciona 0.1% en el valor h alrededor del punto final. Este método es llamado "breking ties".

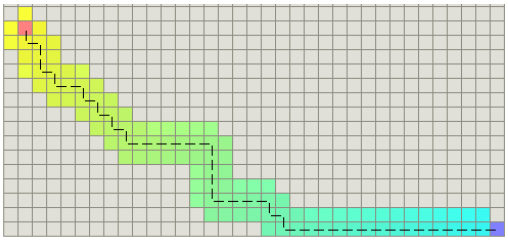

Figura 14. Ruta mediante el método A* Euclidean and breaking-ties. Fuente. Algorithms. *(Introduction to A*, 1997).*

En la figura 14 vemos el trazo de una ruta aplicando el algoritmo A* con el método "breaking-ties", el cual traza una trayectoria inicialmente diagonalmente hasta llegar a la fila donde se encuentra el punto final y continuar de forma diagonal.

La ruta toma celdas que tienden a acercarse con mayor rapidez al punto final en línea recta horizontalmente.

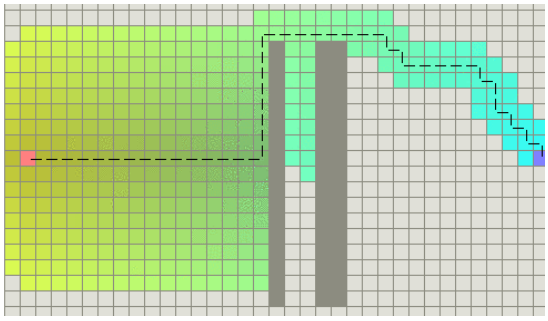

Figura 15. Ruta mediante el método A* "Euclidean" and "breaking-ties" con obstáculos. Fuente. Algorithms. *(Introduction to A*, 1997).*

En la figura 15 la ruta definida aplicándose el algoritmo A* con el método "breaking-ties" el cual la trayectoria va hacia el punto final de destino hasta donde encuentra los obstáculos o celdas de color gris oscuro, el cual la trayectoria va sobre los limites del obstáculos hasta seguir la ruta de

forma horizontal hasta finalmente alcanzar su meta.

El método “breaking-ties” trata de acercándose al punto final antes y después de esquivar el obstáculo médiate incrementar en un valor muy pequeño al valor de h para tener mayor acercamiento al punto final .Este valor es un valor de 0.1% que va adicionado a un factor *p* el cual es tomado como 1/1000 para tener mejor exploración y trazo de su ruta. (Amits Thoughts on pathfinding, 2019).

Un cálculo para manejar el análisis de las trayectorias en mediante el cálculo del producto cruz entre el vector de inicio al punto de llegada y el vector del punto presente y el punto final del vector. El código en computador seria:

```
dx1 = presente.x – final.x
dy1 = presente.y – final.y
dx2 = inicio.x – final.x
dy2 = inicio.y – final.y
cruz =abs(dx1*dy2 – dx2*dy1)
h += cruz*0.001
```

Este cálculo es utilizado mediante tomando la posición inicial del robot y la posición final, así como la posición presente después de su desplazamiento y el punto final de recorrido. El código dará mayor preferencia al camino que tome una línea recta cuando no exista obstáculos.

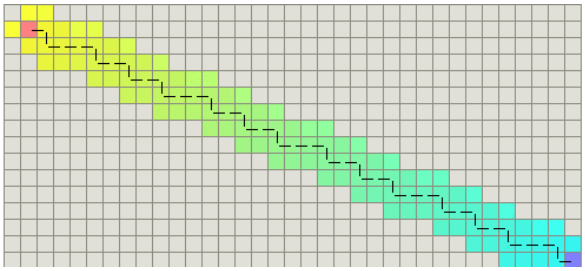

Figura 16.Trazo de línea usando producto cruz y método "breaking ties" para obtener una línea recta entre dos puntos. Fuente. Algorithms. *(Introduction to A*, 1997).*

En la figura 16 vemos un punto de partida de color rojo y una meta o punto de llegada de color azul. En esta imagen se traza una trayectoria aplicando el algoritmo A* con el método "breaking ties" junto con el producto cruz, el cual la línea descontinua se asemeja a una trayectoria reacta.

Las librerías tomadas que permitiría trazar la trayectoria del robot son /base-move, para el manejo de la adquisición estos datos y permitir hacer los cálculos y aplicar el algoritmo A* junto con dijkstra.

La planeación de una ruta para un robot estará formada por dos asuntos. El primer asunto es mediante el análisis de un plano global el cual es enfocado sobre el análisis total o parcial terreno que se encuentra el robot para decide que ruta tomar y llegar a su objetivo o punto final. El segundo asunto es el análisis de un plano local, el cual se encarga de realizar una retroalimentación en tiempo real del terreno alrededor del robot con el propósito de mantener el robot en posición a través del camino escogido por el plano global y no permitir colisiones con el robot.

Un sensor laser estará enviando datos mediante un continuo muestreo dentro de un plano global y local. El plano global analizara la posición del objetivo final y enviara estos datos al plano local. Dentro del plano local analizara la velocidad adecuada por la trayectoria escogida y enviar

la velocidad escogida al robot.

El plano global generalmente realizara un mapa del espacio alrededor del terreno en donde se encuentra el robot con sus parámetros. El plano global generara una serie de puntos en la cual el primer punto es llamado puto de inicio o A y el último punto es llamado punto final o B. El propósito de estos puntos es generar y adicionar muchos puntos (P1,P2…..Pn) que permitan formar la trayectoria entre el punto inicial A y el punto final B. Es posible que la generación de nodos o puntos puedan formar una ruta en la cual el robot busque seguir a través de la unión de los puntos. Los puntos son llamados nodos o celdas los cuales entre más nodos o celdas sean generados después de realizarse una navegación del terreno, permita generar la trayectoria más optima aplicando el algoritmo A* junto con el algoritmo dijkstra. (Open source Robotics Foundation, 2019).

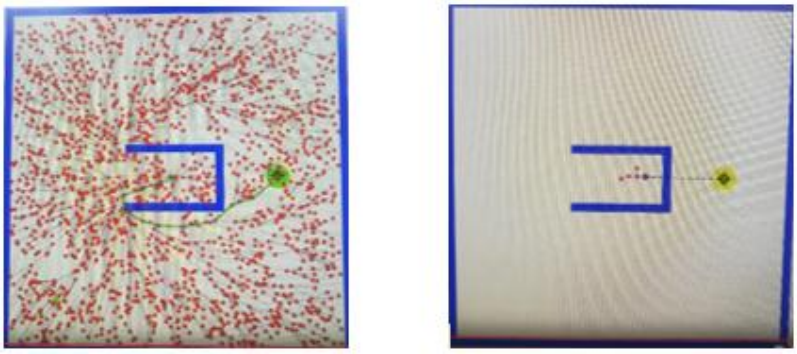

Figura 17. Generación de nodos aproximadamente 2000 nodos (derecha) y generación de 5 nodos(izquierda). Fuente. Rapidly Exploring Random Tree (RRT) and RRT* *(Huang, 2018)*.

En la figura 17 podemos ver que en cuadro de la izquierda doscientos nodos de color rojo y en el cuadro de la derecha cinco nodos de color rojo que nos permite conectar desde punto de partida con los nodos que permitan realizar la trayectoria hasta el punto final.

La generación de muchos nodos forma una cadena de uniones en forma de árbol, la cual al ser computarizada las distancias entre nodos será muy corta lo cual la trayectoria podrá tomar caminos de forma curva y más definida. (Wolfram demostration project, 2018)

El plano local podrá determinar la mejor trayectoria basándose en los datos tomados o adquiridos por el plano global y el sensor laser del robot" turtlebot3 Burger". El plano local también analiza el obstáculo que se encuentra alrededor del robot realizando una marcación con un color permitiendo darles una forma gráfica continua. (Kozak, 2013)

La búsqueda del espacio y manejo de velocidades del robot dentro del plano local estará asociado al método de aproximaciones dinámicas de Windows (DWA). Este método permite generar una trayectoria al robot buscando seguir la mejor ruta trazada por medio de la unión de los puntos conectados basado en el algoritmo A*.

El método de aproximaciones dinámicas de Windows (DWA) buscara mediante la detección de los obstáculos variar los valores de velocidad y escoger una ruta que el robot pueda desplazarse sin tener mucha detección dentro de su translación. Este método es directamente asociado con la dinámica del robot, lo cual buscara un equilibrio entre su velocidad máxima y mínima tanto transnacional como rotacional. Variando la velocidad del robot permitirá trasladarse el robot a posiciones donde no pueda sufrir colisiones el robot y tomar caminos en los cuales el robot pueda seguir y se acerque a la trayectoria optima trazada.

De acuerdo a los valores de velocidad tanto angular como lineal el robot accederá a tomar diferentes tipos de trayectorias. El método buscara variar los valores de velocidad para buscar su mejor orientación y poder llegar a obtener un mejor posicionamiento final que le permita seguir su recorrido con más facilidad para llegar al punto final.

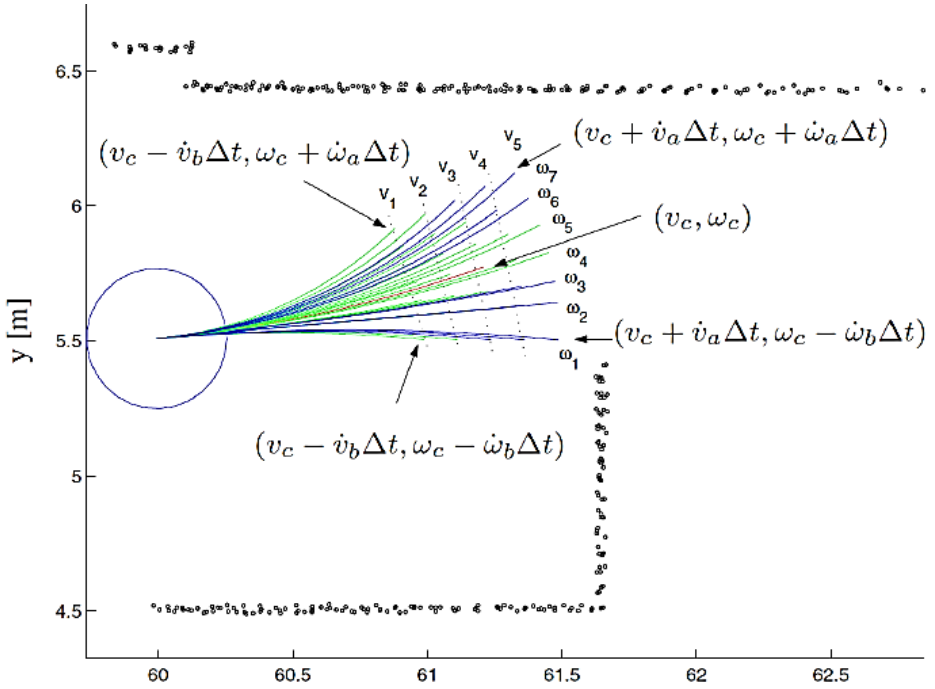

Figura 18. Diferentes rutas al variar el valor de velocidad lineal y angular. Fuente. Local planning for a mobile robot *(Koz´ak, 2013)*.

En la figura 18 podemos ver los diferentes trajecitos que podría tomar un robot móvil al combinar su velocidad lineal(v) y angular(w).Cuando la velocidad angular del robot móvil es cero, el robot móvil se trasladara con una velocidad w1 de forma lineal. A medida que la velocidad angular aumenta la trayectoria se asemeja a un arco.

La trayectoria de color azul describe los distintos cominos erróneos que podría tomar el robot si la velocidad del robot no tiene concordancia. Si la velocidad lineal es alta y la velocidad angular es baja el robot se desplazará en línea recta. Si la velocidad angular es alta y la velocidad angular es baja el robot rotara con poca translación. Cuando la velocidad lineal (v) y angular (w)

es equilibrada, el robot seguirá una trayectoria como se muestra de color rojo. Las trayectorias de color verde podrían ser los caminos que el robot podría tomar para llegar a colisionar con algún obstáculo cercano. (Fox, Burgard, & Thrun, 2017)

El radio de la curva estará dado entre en cociente entre la velocidad lineal y la velocidad angular.

Para tener conocimiento del terreno es necesario hacerse un mapeo por GPS o con sensores laser el cual es parte de un robot "turtlebot3". Este robot permite hacer un continuo conocimiento de un terreno porque contiene un sensor laser (lds) capaz de realizar este trabajo

4.2 Marco Conceptual

Ubuntu MATE 16.04.2LTS

El sistema operativo ideal para este proyecto es llamado "Ubuntu mate", el cual es basado en el sistema operativo linux para placa computadora (SBC). Este sistema operativo permite descargar los paquetes de ROS para Linux y ser compatible con el robot turtlebot3 burger.

Figura 19. Sistema operativo Ubuntu MATE 16.04.2 LTS para raspberry pi. Fuente. How to setup and install ubuntu mate 16.04.2 lts on raspberry pi 3 or 2. *(Frost, 2017).*

La instalación de este sistema operativo además de descargar y manejar paquetes de ROS

para el control y lectura de archivos tipo .py y .ccp , permite construir espacios de trabajo con archivos para recibir y enviar ordenes entre los componentes asociados al robot turtlebot3 burger y ROS. Estos archivos son de tipo .py y. ccp asociados a archivos .xml y .launch

Los archivos son ejecutados desde el ejecutor del sistema operativo para obtener un funcionamiento en la adquisición y trasmisión de datos por los puertos USB.

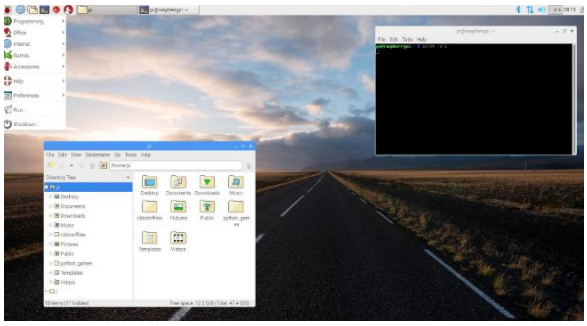

Figura 20. Imagen de inicio después de instalarse el sistema operativo en la raspberry pi. *(Raspberry Pi, 2013)*

Al igual que un computador la rasberry permite conectar un teclado, un mouse, una placa openCR, un sensor LDS entre muchos más componentes que permita la conexión USB y puerto serial.

La imagen del sistema operativo puede verse en un computador portátil mediante conexión VSS via WEB o internet habilitando la conexión remota, o por medio de cable Ethernet mediante el acceso del nombre de la placa y la clave asociada a esta. Además, existen programas en los computadores portátiles con Windows que mediante la dirección IP permite entrar al sistema operativo.

El robot turtlebot3

El robot “turtlebot3” es el último tipo de robot diseñado por la compañía Willian Garage desde el año 2010. Las últimas mejoras de sus prototipos turtlebot3, fue en el año 2017. El robot de menor dimensión y mayor facilidad de uso es llamado “turtlebot3 burger”. (Ackerman, 2013)

Este diseño contiene las últimas tecnologías en el mercado las cuales se encuentra el uso de raspberry pi como computador central del robot; motores dynamxel que contiene memoria RAM, EEPROM, protocolo de comunicación half dúplex UART, para la realización de control de velocidad y aceleración; un sensor laser que permite hacer un mapeo o detección y adquisición de datos con un Angulo de 360° a su alrededor, y por último una tarjeta embebida OpenCR la cual tiene comunicación half dúplex para controlar y comunicarse con los motores dynamixel y la tarjeta raspberry pi tipo B

El turtlebot3 consiste en cuatro platos o bases unidos en forma de hamburguesa. En cada uno de los espacios esta una tarjeta o dispositivos necesarios para su completo funcionamiento. Entre las dos primeras bases o platos inferiores está conectado dos motores dynamixel XL430-W250-T, esenciales para el manejo del robot.

Entre la segunda y tercera base o plato del robot turtlebot3 es colocada la placa OpenCR, importante para la conexión, comunicación y manejo de los dos motores Dynamixel.

Entre la tercera y cuarta base o plato está colocada una placa SBC(computador de una sola placa) o Rasberry Pi modelo B , importante para la comunicación, conexión y trama de datos con la placa OpenCR, el sensor LDS y el computado remoto PC llamado comúnmente como “master”.

Entre la cuarta base o plato es colocado un sensor LDS (Laser Distance Sensor), esencial para la realización del mapeo, localización y navegación. Este sensor es conectado directamente hacia la tarjeta Rasberry Pi o SBC, la cual es encargada de recibir los datos del sensor laser y generar una simultanea localización y mapeo (SLAM) del lugar a ser recorrido.

Figura 21. Robot turtlebot3 burger. Tomado de emanual robotics. Fuente. TurtleBot 3 Burger. *(Robots Components, 2016)*

Este diseño contiene las últimas tecnologías en el mercado las cuales se encuentra el uso de raspberry pi como computador central del robot, dos motores dynamxel que contiene memoria RAM, EEPROM y protocolo de comunicación half dúplex UART, para la realización de control de velocidad y aceleración. Los motores Dynamixel XL430-250-T son una nueva line de actuadores con alto desarrollo tecnológico y conectividad. Estos motores son utilizados en robots como el robot "turtlebot3 burger" porque internamente permiten ser configurados y utilizados para el control PID de posición, velocidad o aceleración, mediante el uso de un sensor encoder(AMS). La velocidad de movimiento de un motor dynamixel XL430-250 es de 47rpm a 61rpm (revoluciones por minuto) entre voltajes de 9 a 12 voltios a una resolución de 0 – 1023.

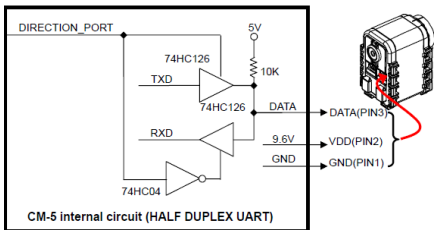

Figura 22. Circuito de comunicación Half dúplex Uart de un motor dynamixel. Fuente. Manual Vl 29.00 *(Robotis, 2010)*

Adicionalmente el robot turtlebot3 contiene un sensor laser(lds) que permite hacer un mapeo o detección y adquisición de datos con un Angulo de 360° a su alrededor,

Figura 23. Sensor de distancia laser LDS-01(LIDAR). Fuente. 360 Laser Distance Sensor LDS-01 (LIDAR). Fuente. 360 Laser Distance Sensor LDS-01 (LIDAR) *(Robotis, 2019)*

Una tarjeta embebida OpenCR la cual tiene comunicación half dúplex para controlar y comunicarse con los motores dynamixel y la tarjeta raspberry pi tipo B.

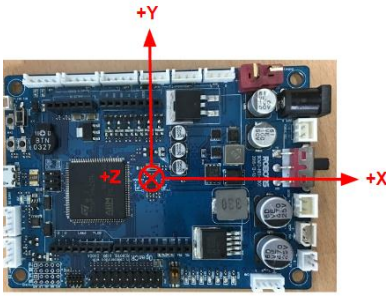

Figura 24.Tarjeta embebida OpenCR. Fuente. Manual Vl 31.30.00 *(Robotis, 2010)*

La tarjeta embebida cuenta con acelerómetro el cual mediante ejes puede tener un análisis de su desplazamiento mediante la movilidad del sensor. La tarjeta tiene salidas y entradas de comunicación tales como: CAN, UART , TTL y RS-485, herramientas que pueden ser programadas desde software de Arduino también. Esta tiene la ventaja de poder ser conectada con tarjetas SBC(single board computer). Esta tiene la capacidad de programarse y relacionarse con los paquetes de ROS, como principal ventaja de la tarjeta OpenCR.

La tarjeta OpenCR tiene la capacidad de ser conectada con una batería externa tipo LI-PO 11.1v, corriente de 800mAh o con cargador externo de variaría de 12V a 5A con interruptor de encendido. Además entradas y salidas PWM, analógicas, digitales, pines JTAG,GPIO ,salida de voltajes de 12v,5v y 3.3v para puerto RTC(Real Time Clock). Además la placa tiene cuatro interruptores como: RESET,BOOT0 (DFU),USB,PUSH SW1 y PUSH SW2.

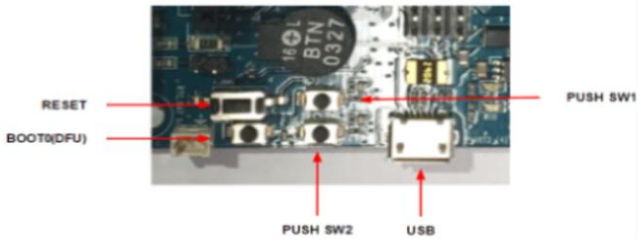

Figura 25. Botones normalmente abiertos para la reactivación del frameware o aplicaciones. Fuente. Manual Vl 31.30.00 *(Robotis, 2010)*

Cuando el programa no puede ser cargado desde el IDE de Arduino es sugerido mantener presionado “Push SW2” y presiona “Reset”. Cuando esto sucede el led de Arduino empieza a parpadear. En este instante es posible cargar el programa compilado en la placa” OpenCR” porque el “fireware” es actualizado.

El “robot tutlebot3 burger” es un dispositivo didáctico para estudiante que quieran conocer de las funciones y características del sistema operativo flexible ROS en la robótica. Este robot permite hacerse diferentes ejercicios o pruebas desde conceptos básico para luego incrementar las habilidades de su uso del sistema flexible ROS, proponiendo nuevos retos o desarrollos con nuevo robot.

4.2.1 La raspberry pi 3 tipo b

Raspberry PI es una placa computadora (SBC) de bajo coste, se podría decir que es un ordenador de tamaño reducido, del orden de una tarjeta de crédito, desarrollado en el Reino Unido por la Fundación Raspberry PI (Universidad de Cambridge) en 2011, con el objetivo de estimular la enseñanza de la informática en las escuelas, aunque no empezó su comercialización hasta el año 2012. (Contreras, 2018)

El concepto es el de un ordenador desnudo de todos los accesorios que se pueden eliminar

sin que afecte al funcionamiento básico. Está formada por una placa que soporta varios componentes necesarios en un ordenador común y es capaz de comportarse como tal.

A la raspberry Pi la han definido como una maravilla en miniatura, que guarda en su interior un importante poder de cómputo en un tamaño muy reducido. Es capaz de realizar cosas extraordinarias. (Contreras, 2018)

4.2.2 Características de la raspberry pi 3 tipo b

Un Chipset Broadcom BCM2835, que contiene un procesador central (CPU) ARM1176JZF-S a 700 MHz (el firmware incluye unos modos Turbo para que el usuario pueda hacerle overclock de hasta 1 GHz sin perder la garantía),

–Un procesador gráfico (GPU) VideoCore IV

–Un módulo de 512 MB de memoria RAM (aunque originalmente al ser lanzado eran 256 MB).

–Un conector de RJ45 conectado a un integrado lan9512 -jzx de SMSC que nos proporciona conectividad a 10/100 Mbps

-2 buses USB 2.0

–Una Salida analógica de audio estéreo por Jack de 3.5 mm.

–Salida digital de video + audio HDMI

–Salida analógica de video RCA

–Pines de entrada y salida de propósito general

–Conector de alimentación microUSB

–Lector de tarjetas SD

–4 puertos USB

Este proyecto fue ideado en 2006 pero no fue lanzado al mercado febrero de 2012. Ha sido desarrollado por un grupo de la Universidad de Cambridge y su misión es fomentar la enseñanza

de las ciencias de la computación los niños. La Raspberry Pi es una excelente herramienta para aprender electrónica y programación. (contreras, 2013)

Los primeros diseños de Raspberry Pi se basaban en el microcontrolador Atmel ATmega644.

En mayo de 2009, la *Fundación Raspberry Pi* fue fundada en Caldecote, South Cambridgeshire, Reino Unido como una asociación caritativa que es regulada por la Comisión de Caridad de Inglaterra y Gales.

La fundación Raspberry Pi surge con un objetivo en mente: Desarrollar el uso y entendimiento de los ordenadores en los niños. La idea es conseguir ordenadores portables y muy baratos que permitan a los niños usarlos sin miedo, abriendo su mentalidad y educándolos en la ética del "ábrelo y mira cómo funciona". El ideólogo del proyecto, David Braven, un antiguo desarrollador de videojuegos, afirma que su objetivo es que los niños puedan llegar a entender el funcionamiento básico del ordenador de forma divertida, y sean ellos mismos los que desarrollen y amplíen sus dispositivos. El co-fundador de la fundación es Eben Upton, un antiguo trabajador de la empresa Broadcom, el cual es el responsable de la arquitectura de software y hardware de la raspberry pi. (contreras, 2013)

Eben Upton, se puso en contacto con un grupo de profesores, académicos y entusiastas de la informática para crear un ordenador con la intención de animar a los niños a aprender informática como lo hizo en 1981 el ordenador Acorn BBC Micro.

La fundación da soporte para las descargas de las distribuciones para arquitectura ARM, Raspbian (derivada de Debian), RISC OS y Arch Linux; y promueve principalmente el aprendizaje del lenguaje de programación Python, y otros lenguajes como Tiny BASIC, C y Perl.(contreras ,2013)

4.3 Marco Histórico

Willow Garage desarrollo oficialmente el hardware y el software para aplicaciones personales en robótica el primero de noviembre de 2007.Willow Garage empezó sus proyectos personales en robótica y ROS se convirtió en una entidad formal. (Garage, 2015)

Tiempo más tarde el grupo ROS llamado como "churn" se asoció con Stanford, permitiendo la construcción de nuevos diseños e implementaciones. Seguidamente a su desarrollo empezó la construcción de su nuevo framework llamado ROS 0.4.

Con la construcción del nuevo framework, el primer robot que permitió la implementación de los paquetes de ROS tiene el nombre de "Milestone 2 Reached". Este robot navego en ocho lugares ingresando por ocho puertas conectando su enchufe en nueve tomas de conexión.

Figura 26. Logo de la compañía Willian Garage. Fuente. Willow Garage. *(willow Garage, 2015)*

En la actualidad han existido doce paquetes o base de datos con repositorios que han permitido de forma libre y flexible la instalación de ROS en sistemas operativos tales como Ubuntu 16.4, Ubuntu Artful,Ubuntu Bionic,Debian Stretch entre otros sistemas operativos compatibles con ROS.

Los doce distribuidores de ROS generadores de archivos y paquetes de ROS, desde el más antigua hasta el más reciente están:

ROS box turtle ,	marzo 2,2010
ROS C turtle ,	agosto 2, 2010
ROS Diamondback ,	marzo 2,2010
ROS Electric Emys ,	agosto 30,2011
ROS Fuerte Turtle ,	abril 23,2012
ROS Groovy Galapagos ,	diciembre 31,2012
ROS Hydro Medusa	septiembre 4,2013
ROS Indigo igloo	julio 22,2014
ROS Jade Turtle	mayo 23,2015.
ROS kinetic Kame	mayo 23,2016.
ROS Lunar Loggerhead	mayo 23,2017
ROS Melodic Morenia	mayo 23,2018

ROS kinetic kame es la versión de ROS seleccionada para la instalación en el sistema operativo Ubuntu 16.4 del computador remoto, además de su instalación en el robot turtlebot 3. Esta versión de ROS nació el 23 de mayo de 2016 (Foote, 2016) y es considerada una versión con características adecuadas para el adecuado funcionamiento del robot Turtelbot 3. La versión es generada para el sistema operativo Ubuntu 16.4 de Linux.

Figura 27. Logo de la versión Kinetic Kame ROS para utilizarse con el robot turtlebot3 burger. Fuente. Willow Garage. *(willow Garage, 2015).*

Antes de la creación del robot turtlebot3 existieron varios modelos de robots creados por Willian Garage. Los primeros robots turtlebot utilizaron la tecnología más reciente para esa época proponiendo alternativas de desarrollo utilizando ROS.

4.3.1 Turtlebot1

Turtlebot1 consiste en una base de iRobot,un packete batería de 3000mAh, una base para conectar con giro, un sensor Kinect ,un portátil Asus 1215N con procesador dual Core y un kit para montaje y arreglo para futuros sensores. El robot fue c creado por Willow Garage , Melonee Wise y Tully foote en Noviembre ,2010. (Ackerman, 2013).

Figura 28. iRobot y Turtlebot1 elaborado en Willow Garage. Fuente. Willow Garage. *(willow Garage, 2015)*

4.3.2 Turtlebot2

Turtlebot2 consiste en una plataforma Yujin Kobuki, una batería de 2200mAh de rápida carga, un sensor Kinect, un portátil Asus 1215N con procesador dual Core ,un conector para cargador y un kit para armado y montaje con disponibilidad para conexión en futuros sensores. Este robot fue creado en octubre del año 2012 (Ackerman, 2013).

Figura 29 Turtlebot2 (izquierda) y Sensor Kinect(derecha). Fuente. TurtleBot 2 *(Generation Robots, 2015)*

4.3.3 Turtlebot3

Turtlebot3 consiste en una idea de varios módulos de platos que usuarios puede escoger a partir de su forma. El turltebot3 viene en tres tipos: Burger, waffle y waffle Pi. Cada uno tiene una forma distinta en tamaño y en funcionalidad.

El turtlebot3 consiste en una base, dos motores dynamixel, una batería de 1800mA, un sensor lidar de 360 grados de giro, una cámara (cámara "Real Sense" para wiffle y cámara rasberry pi para waffle pi), una SBC (computador de una sola placa: rasberry pi 3 tipo B y Intel Joule 570X) y un kit para montaje y construcción para adición de más sensores. Los tipos de robot turtlebot3 fue creado en mayo del año 2017(Ackerman, 2013).

Figura 30. Turtlebot 3 Burger(Izquierda) , Turtlebot 3 Wiffle(medio) y Turtlebot 3 Wiffle Pi (derecha). Fuente. TurtleBot 2 *(Generation Robots, 2015)*

4.4 Estado del Arte

4.4.1 Internacionales

En este artículo resuelve un reto de trazar trayectorias visualizando obstáculos mediante un sistema autónomo con varios automóviles. En ese artículo empieza analizando el costo o ruta fácil de acceso sobrepasando los agentes u obstáculos mediante la modificación del algoritmo Dijstra. En una segunda etapa analiza la detección y predicción de varios automóviles atreves del camino teniendo en cuenta la selección del camino. Por último, permite hacer una visión y localización de los puntos de llegada y automóviles sobre la pista que se encuentra dentro de todo el terreno previamente definido. Con este artículo permite analizar la aplicación del algoritmo Dijkstra para la generación de trayectorias detectando automóviles y obstáculos para no permitir tener un choque. (Mahdavi & Carvalho, 2018)

En este artículo se estudia la planeación y la trayectoria de un robot por medio de varias generaciones de trayectorias médiate algoritmos. Este estudio genera la más eficiente ruta a través de un ambiente selectivo. Dentro de estudio aplican el algoritmo Dijkstra y hace algunas modificaciones para encontrar un apropiado camino y alcanzar su destino. El robot desarrollado fue desarrollado mediante un simulador V-REP de fácil y abierto uso, donde se aplica el algoritmo Dijkstra analizando la trayectoria para no tener colisiones y seleccionar una trayectoria óptima. (Julius Fusic, Ramkumar, & Hariharan, 2018)

En este artículo define la generación de trayectorias para un robot aéreo en el cual transita por zonas seguras y detecta obstáculos en el cual no permite su choque con estos obstáculos. Analiza matemáticamente mediante matrices en tres dimensiones y simulación con Matlab de las posibles variaciones que puede tomar el robot aéreo y no permitir que este se pueda perder el control y choque en alguna área con obstáculos. En este artículo permite analizar de manera vectorial su desplazamiento y rotación mediante un modelo no lineal de control. El modelamiento

es determinado un mínimo valor entre el obstaculillo y el robot dentro de un radio definido. Para modelar el sistema de control no lineal se implementó un modelo espectral “Radau-pseudo”. (Kosari & Teshnizi , 2018).

4.4.2 Locales

En este artículo presenta un método para la generación de trayectoria y control de frecuencia entre varios circuitos de un robot dentro una competición de bajo nivel, el cual predice su próximo movimiento mediante la aplicación de “B-spline” y el filtro “Kalman”. El filtro “Ksdealma” puede predecir su próximo movimiento dentro de un tiempo de procesamiento y “B-spline” controla la optimización de los puntos de la trayectoria y suavizar la toma de curva esquivando obstáculos. Este artículo permite exponer la curva que puede tomar un robot para llegar desde un punto inicial a un punto final. (Restrepo, Villegas, Arias, Serna, & madrigal, 2012)

5. Metodología

Este proyecto se describe bajo un tipo de investigación descriptiva exploratoria, ya que se planea dar una visión general de un reconocimiento del espacio a navegar para optimizar su trayectoria utilizando un robot “TURTLEBOT3 BURGER”, el cual se inicia realizando un estado del arte sobre los proyectos realizados, luego se estudia el sistema de operación para robots, el cual permite escoger los algoritmos que permita la optimización de trayectorias y finalmente realizar pruebas de funcionamiento y comunicación.

El método planteado en la tabla 1, parte del análisis del funcionamiento del robot “TURTLEBOT3 BURGER” dentro de un espacio de trabajo específico de manera hexagonal, verificando la programación de algoritmos “Astar” y “Dijstra” en el robot mediante pruebas de funcionamiento que permita evaluar la menor trayectoria por el cual el robot avance ,con el fin de tomar mejores decisiones en su trayecto y demorar menos tiempo en alcanzar su objetivo o meta.

Tabla 1.

Actividades realizadas para definir el proceso metodológico del proyecto.

Actividad	Tareas	Respuestas
Establecer variables	Con el robot "TURTLEBOT3 BURGER" ¿Qué variables se pueden medir?	distancia, velocidad, tiempo
Compra de materiales	• Verificar características • Verificar disponibilidad	Establecer medidas para el diseño del terreno en donde se desplazara el robot de forma hexagonal.
Ensamble del Robot y base de funcionamiento	Considerar la altura del robot y realizar las barreras donde acote las dimensiones del espacio de trabajo para el desplazamiento del robot "TURTLEBOT3 BURGER"	Realización del montaje del robot y de la base del terreno donde el robot se desplazará.
Pruebas de funcionamiento.	Verifica el correcto funcionamiento del robot "TURTLEBOT3 BURGER" analizando su optimización de trayectorias utilizando obstáculos dentro del terreno	Verificar la ruta que toma el robot para obtener resultados y comprobar la aplicación del algoritmo que permite la optimización de trayectorias usando el robot "TURTLEBOT3 BURGER"

Fuente. Autor basado Metodología de la Investigación. (Hernández, Fernández, & Baptista, 2014)

6. Resultados

6.1 Dimensiones del área del funcionamiento del robot

Para la realización del proyecto el robot “turtlebot3 burger” estará moviéndose dentro de un área de forma hexagonal. Las longitudes de las paredes y la totalidad del área permitirán que el robot pueda trasladarse y rotar con o sin obstáculos.

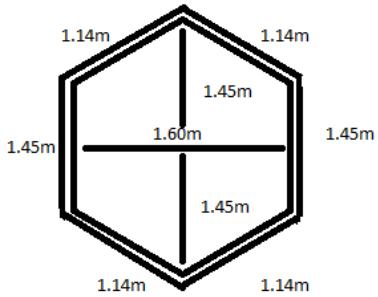

Figura 31. Área definida del robot de forma hexagonal. Fuente. Autor

La altura del área hexagonales es de veintitrés (23cm). Esta altura supera el sensor lds que se encuentre en la parte superior del robot “turtlebot3 burger”. Esto permite realizar el mapa dentro del hexágono.

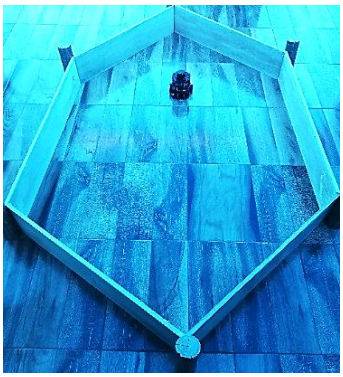

Figura 32. Montaje del área hexagonal. Fuente. Autor

En la figura 32 vemos el área hexagonal limitada para las pruebas de funcionamiento, en donde se encuentra el robot “Turtlebot3 Burger” dentro del área para hacer un escaneo y mapeo del terreno.

El robot “turtlebot3 burger” tiene un sensor laser que permite hacer un simultaneo mapeo y localización del terreno. Esto detecta los obstáculos o barreras cercanas o lejanas que estén al alcance del sensor. Luego permite hacerse un mapa para visualizarse en el software libre “Rviz”.

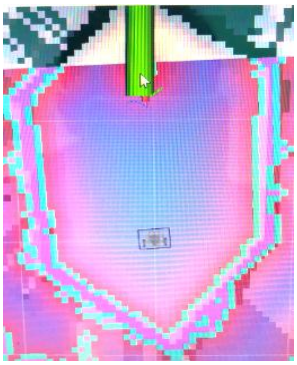

Figura 33. SLAM utilizando el robot Turtlebot3 burger. Fuente. Tomado de Rviz.

En la figura 33 vemos el mapeo del área hexagonal donde el cuadro azul define el robot “Turtlebot3 burger” y las líneas negras como las barreras o limitaciones del terreno

La realización de “SLAM” es visualizado y ejecutado desde el programa Rviz en el computador remoto. El programa permite realizar acciones las cuales sean especificadas dentro de un archivo rviz. El archivo rviz permite visualizar el robot, el mapa del terreno y la trayectoria por donde el robot transitará.

Para comprobar la programación del algoritmo “A-star(A*)” junto a “dijkstrar” sobre el robot, realizamos varios trayectos, los cuales posicionamos el robot dentro del área hexagonal y especificamos una posición final con diferentes obstáculos. El programa rviz junto con los nodos asociados al proyecto trazara una ruta optima según el algoritmo dijkstra o A-star(A*).

6.2 Conexiones del robot

El sensor “lds” en la parte superior es conectado mediante un cable tipo A macho a un puerto USB de la raspberry pi tipo B. De la misma forma es conectada la tarjeta openCR con la raspberry pi tipo B. Los dos motores dynamixel son conectados a la tarjeta OpenCR a los puertos TTL (tres pines tipo macho) los cuales permite manejar voltajes de 11.1v y comunicación entre la tarjeta y los motores.

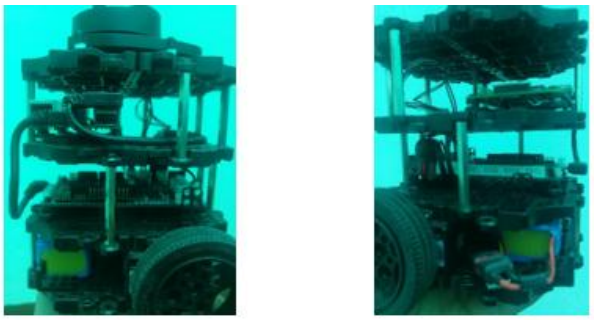

Figura 34. Conexiones físicas del robot turtlebot3 burger. Fuente. Universidad Santo Tomás, Tunja.

En la figura 34 vemos el robot con sus conexiones, el cual la batería de litio de 1000 mA a un voltaje de 11.1v recargable de color azul se conecta a la tarjeta “openCR”, la “tarjeta openCR” se conecta a la tarjeta “Raspberry pi” por cable de datos usb-otg, el sensor “LDS” se conecta a la tarjeta “raspberry pi” por medio de cable USB, la tarjeta “openCR”. Además los pines de alimentación tanto de la tarjeta “raspberry pi” como de la tarjeta “openCR” van conetados. Los dos motores del robot se conectan a los puertos TTL de la

la tarjeta openCR.

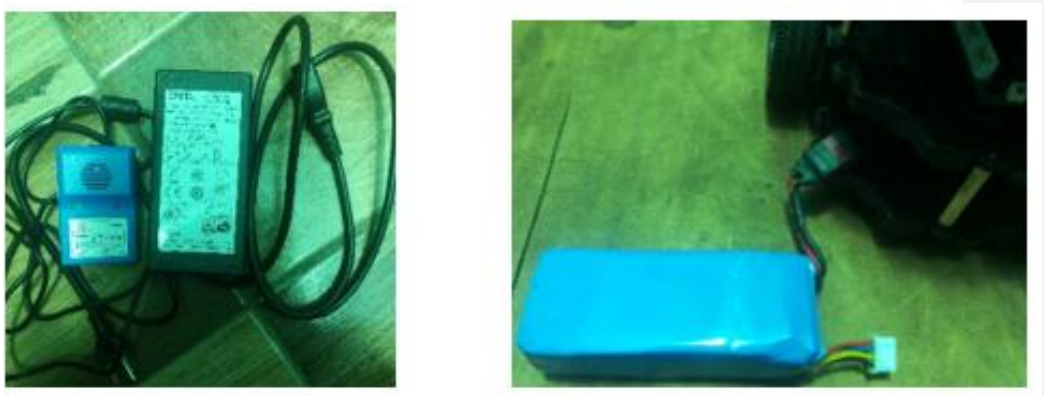

Figura 35. Batería recargable y cargador de batera del robot “turtlebot3 burger”. Fuente. Universidad Santo Tomás, Tunja.

En la figura 35 vemos el cargador y la batería de litio del robot. Al conectar la batería de litio al cargador un bombillo de color rojo se encenderá. Esto indica que la batería empezó a cargarse. Cuando la batería está cargada un bombillo verde se encenderá y el bombillo de color rojo deberá apagarse. Si los dos bombillos están prendidos o ninguno de los dos bombillos se prende la batería estará dañada. Con la batería cargada podrá ser conectada a la placa “openCR” y activarse todos los elementos pertenecientes al robot utilizando el interruptor de activación de la placa openCR.

La tarjeta OpenCR cuenta con un interruptor que permite prender o apagar todos los elementos del robot. Sin la batería el robot puede prender todos sus componentes conectando el cargador sin la bateria utilizando el puerto SMPS de la tarjeta OpenCR

La conexión entre el remoto pc y la raspberry PC empieza desde la conexión de ambos dispositivos dentro de una misma red local. Cuando ambos están conectados a una misma red local es posible configurado el remoto pc como “master” y la rasberry del robot “turtlebot3 burger” como

hostname.

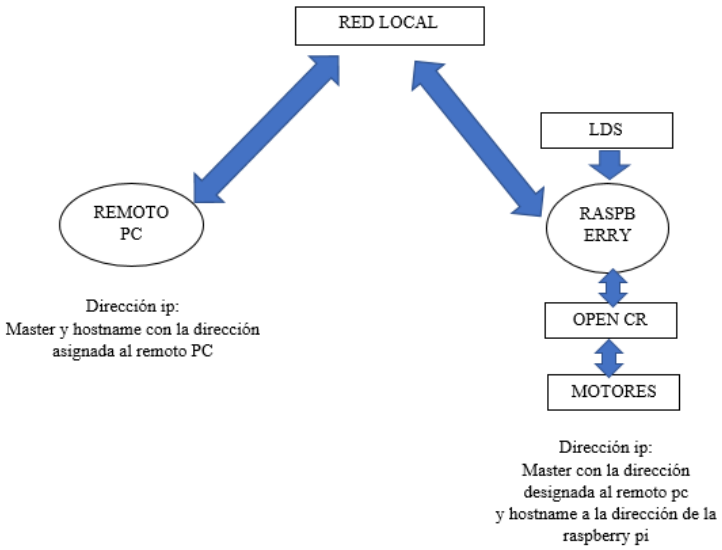

Figura 36. Diagrama de conexión del robot "turtlebot3 Burger" con un computador de forma remota. Fuente. Autor, basado en Manual de Turtlebot3 Burger.

La comunicación dentro de una red local permite enviar y recibir datos del remoto pc al robot y del robot a la raspberry pi al mismo tiempo. Esta comunicación permite que el remoto pc pueda graficar los datos obtenidos por el sensor LDS en el programa "Rviz". Al mismo tiempo el remoto pc puede enviar órdenes al robot para activar los motores de robot y avanzar hacia alguna dirección.

6.3 Diagrama de flujo del funcionamiento del robot"Turtlebot3 burger"

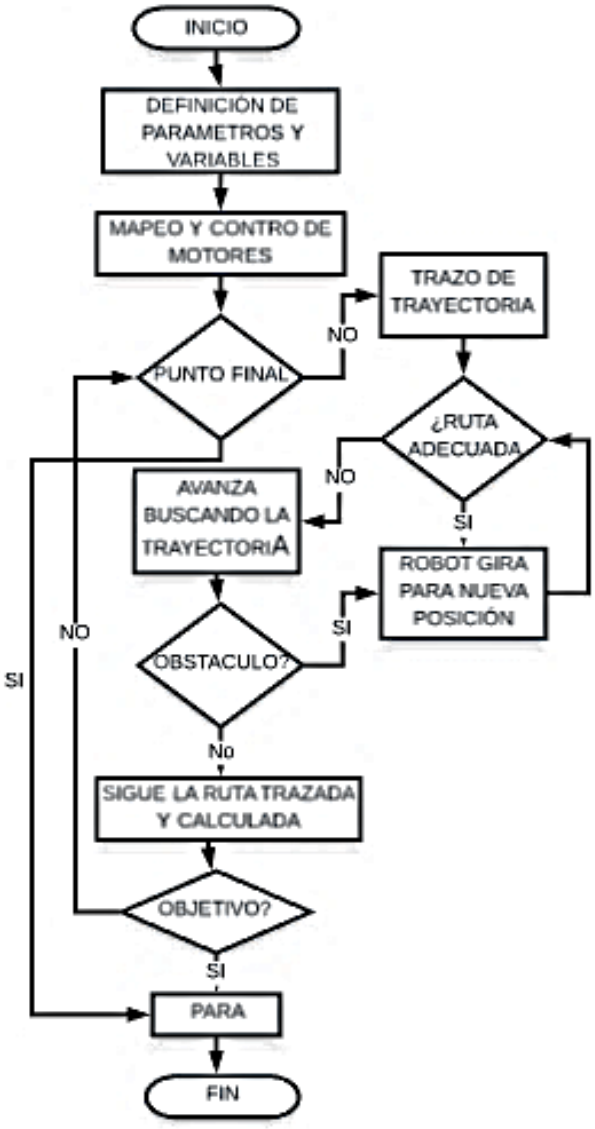

Figura 37. Diagrama de flujo para funcionamiento del robot ¨Turtlebot3 Burger¨. Fuente. Autor.

Comentado [L1]: No se entiende esta figura y tampoco esta llamada o vinculada en el texto

En la figura 37 muestra la lógica de programación del robot “turtlebot3 burger”. El cual primero define parámetros y variables, luego se controla los motores para realizar el reconocimiento de espacios, después define si se especifica un punto final o meta para luego seguir la trayectoria optimizada hasta alcanzar su objetivo.

El método de dinámica de aproximaciones de Windows (DWA) describe los casos como el robot opera para llegar desde un punto inicial hacia un punto final detectando y esquivando obstáculos. En principio al ejecutarse el programa contendrá todos los nodos de conexión entre el robot y el computador junto con los parámetros y las variables definidas para su funcionamiento.

El robot enviara los datos del sensor laser y serán visualizados y mapeados simultáneamente utilizando el programa Rviz.

El algoritmo de funcionamiento general permite tener una constante lectura del robot con respecto a la generación de trayectorias trazadas en el programa “Rviz”. El funcionamiento del robot es seguido gráficamente mediante una sucesión de nodos que realizan funciones independientes. Cada función es construida y ejecutada desde un directorio y un archivo que define los parámetros y variables que serán necesarias.

Los nodos asociados a la programación del ordenador y el robot “TURTLEBOT3 BURGER” permite la comunicación con ROS mediante mensajes teniendo incluso conexión con elementos externos (Trapi, 2017). Un ejemplo es obtener las coordenadas del robot y ser utilizadas para conducir y orientar el robot. Otro ejemplo es realizar el simultaneo mapeo alrededor del robot usando “SLAM” y visualizar la velocidad lineal y angular del robot “turtlebot3 Burger” en el ejecutable del sistema operativo.

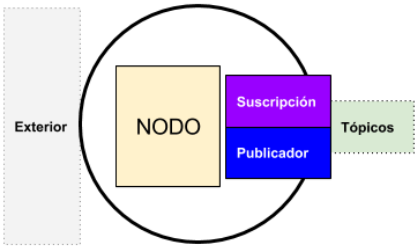

Figura 38. Gráfica de Nodo. Fuente .Tomado de ROS.

Los tópicos son acciones más específicas que el sistema pueda acceder en cada nodo. Cada nodo puede tener más de un tópico. Cuando uno nodo está en funcionamiento se asocia a un tópico que permitirá realizar una tarea de manera específica a través de ordenes por medio de mensajes.

Los mensajes son asociados a "suscripción y publicador" en ROS. La suscripción es el encargado de etiquetan a un tópico con el nodo y acceder a información entre nodos y tópicos. El publicador es el encargado de permitir ver la información asociada a un nodo o tópico para visualizarse o realizar alguna tarea.

Gráficamente los nodos son escritos con una circunferencia y los tópicos con un rectángulo, construidos a partir de la ejecución de archivos c++ o Python.

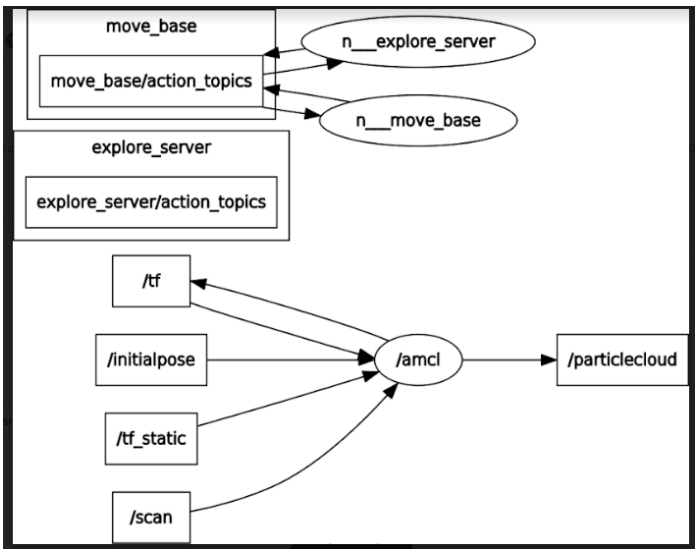

Figura 39. Nodo /amcl. Fuente: Tomado de Ubuntu 16.4. Fuente. Autor.

El nodo /amcl permite leer la posición inicial del robot, las coordenadas y el continuo escaneo en la variación de su posición para ser publicada en el tópico /particlecloud el cual permite analizar la variación de sus coordenadas desde la posición (0, 0, 0) y comunicarse con el computador remotamente.

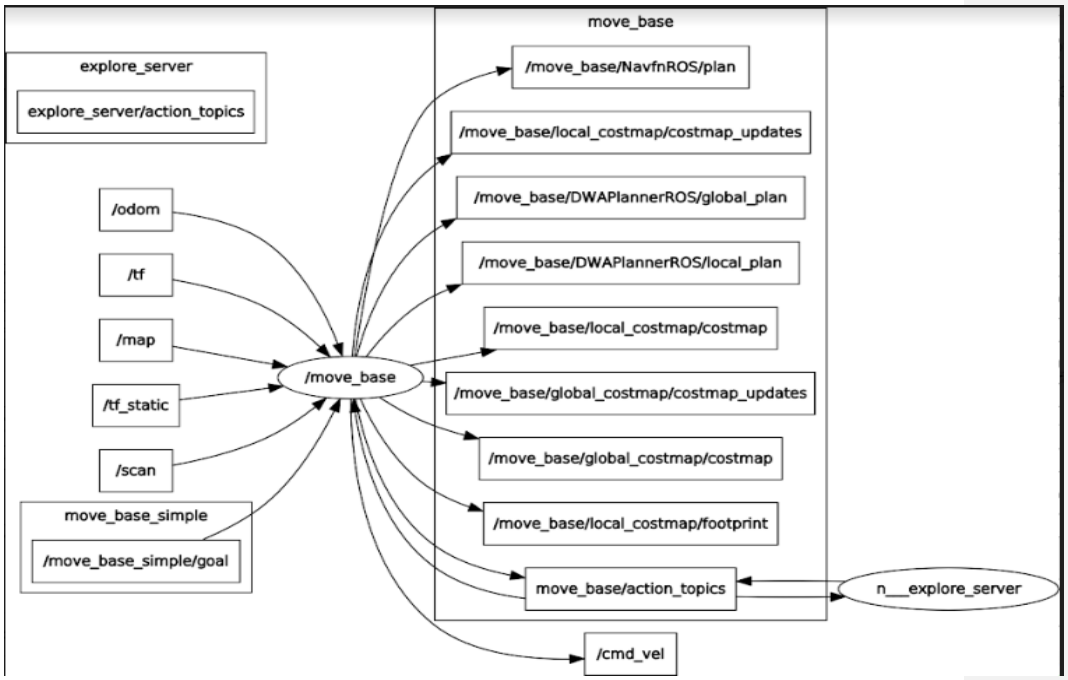

Figura 40. Nodo /move_base. Fuente: Tomado de Ubuntu 16.4. Fuente. Autor.

En la figura 40 vemos el nodo “Move_base” necesario para el funcionamiento del robot, en donde se encarga de mover el robot hacia una meta o punto final. En este nodo se asocian “topicos” o tares que permite realizar el funcionamiento total del robot.

Este nodo recibe los datos tomados del sensor LDS unido con los datos de posicionamiento del robot para construir la trayectoria y variar la velocidad lineal o angular del robot “turltebot 3” mediante el uso del topic /cmd_vel.

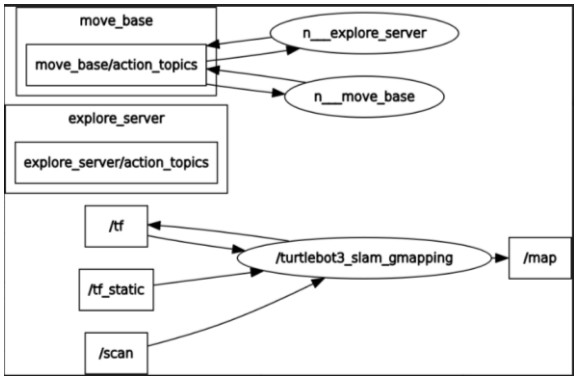

Figura 41. Nodo /turtlebot3_slam_gmapping. Fuente. Autor.

El nodo /turtlebot3_slam_gmapping permite hacer el mapeo del robot alrededor de su trayectoria. Este nodo recibe los valores de posicionamiento del robot permitiendo tener una referencia del robot dentro del mapa.

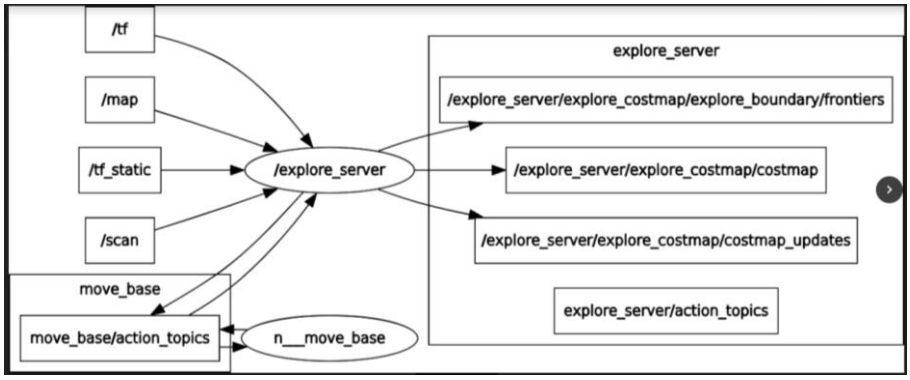

Figura 42. Nodo /explore_server. Fuente. Autor.

El nodo explore_server es el encargado de recibir el valor del punto final de la trayectoria y convertir este valor es un valor binario. Este valor binario va ser recibido por el nodo move_base

para realizar la trayectoria desde el punto posicionado del robot hasta el punto final definido por el usuario.

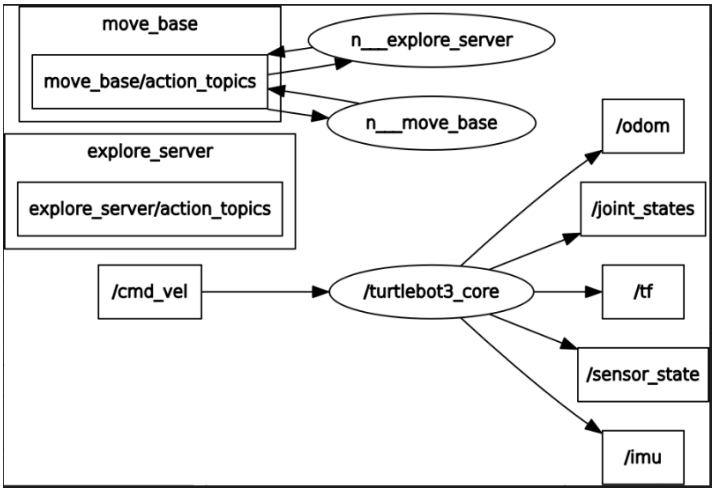

Figura 43. Nodo /turtlebot3_core. Fuente. Autor.

El nodo turtlebot3_core es el encargado de recibir los valores del tópico /cmd_vel y guiar el robot a través de la trayectoria creada por el nodo move_base controlando el valor de posición, orientación y covarianza del robot.

Un archivo launch es el encargado de reunir argumentos, nodos y parámetros en lenguaje xml. Este lenguaje encabeza y cierra el archivo con el nombre <launch>. Dentro de este archivo defino características que quiero agregar en la ejecución de mi proyecto.

9

```
<launch>
  <!-- Arguments -->
  <arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="model type [burger, waffle, waffle_pi]"/>
  <arg name="configuration_basename" default="turtlebot3_lds_2d.lua"/>
  <arg name="sensor_range" default="1.0"/>
  <arg name="cmd_vel_topic" default="/cmd_vel" />
  <arg name="odom_topic" default="odom" />

  <!-- TurtleBot3 and Gmapping -->
  <include file="$(find turtlebot3_slam)/launch/turtlebot3_gmapping.launch">
    <arg name="model" value="$(arg model)" />
  </include>

  <!-- AMCL -->
  <include file="$(find turtlebot3_navigation)/launch/amcl.launch"/>

  <!-- move_base -->
  <node pkg="move_base" type="move_base" respawn="false" name="move_base" output="screen">
    <param name="base_local_planner" value="dwa_local_planner/DWAPlannerROS" />
    <rosparam file="$(find turtlebot3_navigation)/param/costmap_common_params_$(arg model).yaml" command="load" ns="global_costmap" />
    <rosparam file="$(find turtlebot3_navigation)/param/costmap_common_params_$(arg model).yaml" command="load" ns="local_costmap" />
    <rosparam file="$(find turtlebot3_navigation)/param/local_costmap_params.yaml" command="load" />
    <rosparam file="$(find turtlebot3_navigation)/param/global_costmap_params.yaml" command="load" />
    <rosparam file="$(find turtlebot3_navigation)/param/move_base_params.yaml" command="load" />
    <rosparam file="$(find turtlebot3_navigation)/param/dwa_local_planner_params_$(arg model).yaml" command="load" />
    <remap from="cmd_vel" to="$(arg cmd_vel_topic)"/>
    <remap from="odom" to="$(arg odom_topic)"/>
  </node>

  <!-- frontier_exploration -->
  <node pkg="frontier_exploration" type="explore_client" name="explore_client" output="screen"/>
  <node pkg="frontier_exploration" type="explore_server" name="explore_server" output="screen" >
    <param name="frequency" type="double" value="1.0"/>
    <param name="goal_aliasing" type="double" value="$(arg sensor_range)"/>
    <rosparam file="$(find turtlebot3_navigation)/param/costmap_common_params_$(arg model).yaml" command="load" ns="explore_costmap" />
    <rosparam file="$(find turtlebot3_slam)/config/frontier_exploration.yaml" command="load" ns="explore_costmap" />
  </node>
</launch>
```

Figura 44. Archivo frontierexploration.launch. Fuente. Autor.

En la figura 44 se puede observar el archivo “frontierexploration.launch” para la ejecución en el terminal del computador mediante el comando “roslaunch”, junto con el nombre la carpeta principal donde se encuentra este archivo. Dentro de este archivo se definen parmetros del sistema operativo para robots(ROS),los nodos junto con los paquetes o archivos que permite el funcionamiento del robot desde un computador de manera remota.

Para esto previamente se define el robot desde el terminar del computador con la instrucción del sistema operativo para robots(ROS)” export TURTLEBOT3_MODEL=burger” y luego se ejecuta mediante el comando ”roslauncha” el paquete y archivo launch.

En el terminal de la raspberry pi tipo B se ejecuta desde el terminal el archivo “lauch” el cual vincula el funcionamiento del robot dependiendo de las instrucciones recibidas desde el computador remotamente y del sensor “LDS” y tarjeta que controla los dos motores “openCR”.

En archivo “frontier exploration.launch” es el encargado de tomar los archivo de ROS que permiten manejar el robot “TURTLEBOT3 BURGER” con los algoritmos Astar y Disjstra dentro de un plano global y local junto con los nodos, parámetros y tópicos que conecta el ordenador con el robot desde el sistema operativo Ubuntu 16.4.

En la ejecución de la escritura de un archivo launch puede contener la declaración de un nodo mediante <node (el nombre, tipo….etc.)>, parámetros de un nodo mediante el comando de <param> o <rosparam> si es un archivo de ROS, declaración de argumentos pertenecientes de un nodo a otro nodo mediante el comando <remap>,incluir otros archivos de launch mediante <include (nombre del archivo)> ,enviar variables en nodos definidos después mediante el comando<env >, probar los nodos creados mediante el comando <test (nombre del nodo)>,argumentos o archivos pertenecientes a dos archivos launch asociado con un proyecto mediante el uso del comando <arg(nombre,tipo….etc) > y configurar de varios nodos dentro de un grupo mediante el comando <group >.

6.4 Configuración y programación del robot

Para realizar pruebas de funcionamiento de los motores dynamixel, la placa OpenCR tendrá que ser programada utilizando el IDE de Arduino. Utilizando el IDE de Arduino se descarga las librerías de la OpenCR. Esta se descarga desde el ejecutable de Ubuntu.

https://raw.githubusercontent.com/ROBOTIS-GIT/OpenCR/master/arduino/opencr_release/package_opencr_index.json

Para configurar los motores se entra al archivo turtlebot3/turtlebot3_setup/turtlebot3_setup motor y se carga a la placa openCR para configurarse con los parámetros de transmisión de datos de 9600 baudios. Cuando el archivo es cargado dentro de la placa OpenCR es posible definir cuál motor va a mover a la derecha y cual motor a la izquierda según el puerto TTL de la placa openCR.

Luego de realizarse la configuración de los motores en la tarjeta se cargará un archivo localizado en turtlebot3/turtle3-core/ dentro del IDE del Arduino. Este archivo tiene definido la lectura de los motores para entrar en funcionamiento según las ordenes de ROS médiate la tarjeta raspberry pi.

Para comprobar que quedaron bien configurado los motores se utilizan los “switchs” de la tarjeta OpenCR, Al presionarse el “swich1” o “switch2” por un tiempo el robot rotara ciento ochenta grados o avanzara un metro. (Robotis, 2019).

La raspberry pi tipo B después de tener instalado la imagen o el sistema operativo tendrá que descargar paquetes de ROS usando comandos en el ejecutable del sistema operativo.

Después de tener instalado ROS dentro de la raspberri pi tipo B es posible manejar los comandos y las herramientas que permite ROS.

El siguiente paso es configurar las direcciones ip del master y el hostname. El computador remoto será master y la raspberri pi tipo B será definida como “hostname”. Para más información dirigirse a : http://emanual.robotis.com/docs/en/platform/turtlebot3/pc_setup/ .

Luego en el ejecutable del computador remoto se ejecuta el comando “roscore”. Seguido a esto se ejecuta el comando roslaunch turtlebot3_bringup turtlebot3_robot. Launch en el ejecutable de la raspberry pi tipo B.

En el computador remoto se define el tipo de robot que se está utilizando. En este caso es definido el robot turtlebot “burger” y el directorio con un archivo roslaunch frontier_exploration robot.launch.

La programación del robot “TURTLEBOT3” es vinculada al controlador principal u ordenador que se asocia a la raspberry pi y a la tarjeta openCR. En la raspberry pi se instala ROS donde queda con los archivos asociados al robot turtlebot3.

Para programación de la tarjeta opencCR se debe tener un IDE de Arduino.Dentro de esto se

descarga la librería asociada a la tarjeta OpenCR. Dentro de preferencias se pega dentro de "preferencias". (Raw.githubusercontent, s.f.)

Luego se escoge en "tools" la "board" con el nombre de "openCR board" y el puerto que se le asigne, y el programador AVR ISP.

Se abre el archivo turtlebot3_core dentro de la carpeta turtlebot3/turtlebot3_burger. En este archivo se define las variables, constantes, funciones y librerías asociadas al robot "turtlebot3". Dentro de esto se vincula los nodos, tópicos y suscritos que puede manejar el robot con algún ordenador.

Dentro del archivo "turtlebot3_core.h" se encuentran tres archivos: turtlebot3_core, turtlebot3_burger y turtlebot_core_config.

El Archivo turtle_core_confic es realizado para ser incluido dentro del archivo turtlebot_core. Dentro del archivo turtebot3_burger define las variables físicas y constantes del robot "turtlebot3" como son: radio de las llantas, máximo y mínimo valor de velocidad lineal del robot, mínimo y máximo valor del encoder, máxima y mínimo valor de velocidad angular, radio de desplazamiento del robot y el nombre asociado a todas estas variables. En este caso con el nombre "Burger".

En el archivo turtlebot_core contiene varias funciones asociadas a la tarjeta openCR para vincularse con ROS. En la función "void setup" se inicia la configuración de ROS para la tarjeta openCR junto con la velocidad de transmisión de datos en baudios junto con subscriptores de los nodos. Además, inicializamos las funciones del motor, del sensor lds, los controladores de velocidad, los análisis y manejo de odometry, y el tiempo de operación del programa. Dentro de la función void loop se utiliza la funcion *millis()* para controlar los tiempos de utilización de funciones y realizar la publicación en nodos mediante la frecuencia en los paquetes.

Dentro de la *funcion void loop* se utilizan funciones para activar la emisión de mensajes, verificar si es presionado los botones de la tarjeta openCR, emitir y tener respuesta sobre los

subscritos y publicadores de ROS.

Otras funciones son para controlar los motores por medio de su velocidad con el nombre de "commandVelocityCallback", función para emitir y recibir órdenes mediante mensajes con respecto a los sensores, motores, la bateria,funcion para controlar los motores mediante botones de la tarjeta openCR con el nombre "drive test" y otras funciones como la encargada de evaluar y emitir la posición del robot "tft update refine" y "initOdom".

Luego que compilar y cargar el archivo "turtlebot3_core", se programa la tarjeta raspberry pi donde podrá leer y conectarse con la tarjera openCR mediante la librería "*node_handle.h*", y con el ordenador mediante una red local con extensión 192.168.0.X, que se asociara a algún ordenador que vincula su dirección de red y los nodos, subscritos y tópicos.

En la raspberry pi se instala ROS donde instala el espacio de trabajo y los archivos que vinculara las aplicaciones que controlan el robot turtlebot3 Burger. En la raspberry pi se conecta con la tarjeta openCR mediante el archivo turtlebot3_core. launch y con el sensor lds mediante el archivo turtlebot3_lidar. launch. Estos archivos van vinculados al archivo turtlebot3_bringup.launch.

```
<launch>
  <arg name="multi_robot_name" default=""/>
  <arg name="set_lidar_frame_id" default="base_scan"/>

  <include file="$(find turtlebot3_bringup)/launch/turtlebot3_core.launch">
    <arg name="multi_robot_name" value="$(arg multi_robot_name)"/>
  </include>
  <include file="$(find turtlebot3_bringup)/launch/turtlebot3_lidar.launch">
    <arg name="set_frame_id" value="$(arg set_lidar_frame_id)"/>
  </include>

  <node pkg="turtlebot3_bringup" type="turtlebot3_diagnostics" name="turtlebot3_diagnostics" output="screen"/>
</launch>
```

Figura 45. Archivo Launch perteneciente a la programación en el robot turtlebot3. Fuente. Autor.

El archivo "turtlebot3_bringup" asocia un nodo el cual se llamado" turtlebot3_diagnostic",

en el cual se asocia a la tarjeta openCR.

```
:launch>
 <arg name="multi_robot_name" default=""/>

 <node pkg="rosserial_python" type="serial_node.py" name="turtlebot3_core" output="screen">
  <param name="port" value="/dev/ttyACM0"/>
  <param name="baud" value="115200"/>
  <param name="tf_prefix" value="$(arg multi_robot_name)"/>
 </node>
:/launch>
```

Figura 46. Programación de la comunicación con la tarjeta openCR y la tarjeta raspberrypi 3.

Fuente. Autor.

Dentro del archivo turtlebot3_core. launch se enfoca en vincular el nodo de transmisión de 115200 baudios.

```
<launch>
 <arg name="set_frame_id" default="base_scan"/>

 <node pkg="hls_lfcd_lds_driver" type="hlds_laser_publisher" name="turtlebot3_lds" output="screen">
  <param name="port" value="/dev/ttyUSB0"/>
  <param name="frame_id" value="base_scan"/>
 </node>
</launch>
```

Figura 47. Programación en la rasberry pi para unir el sensor lds con el puerto USB. Tomado de Ubuntu

Fuente. Autor.

Dentro del archivo turtlebot3_lidar se enfoca en generar el nodo que se vincula al puerto USB de la tarjeta raspberry pi y en la cual permite obtener información sobre su funcionamiento y convertirlo en binario para ser interpretado y visualizado en “Rviz”.

Dentro del espacio de trabajo se encuentra el archivo CmakeLists.txt donde se adicionan los paquetes roscc, std_msgs, sensor_msgs, diagnostic_msgs y turtlebot3_msgs. Además, se encuentran los directorios que van vinculados al paquete del espacio de trabajo y se instala los paquetes de manejo del directorio y diagnóstico del manejo de funcionamiento del robot de acuerdo a las órdenes recibidas y emitidas desde el robot

En el archivo dentro de catkin_ws/src/turtlebot3/src/turtlebot_bringup/src se encuentra un archivo que define los nodos y vincula los archivos de ros que permite que el robot se conecte con algún ordenador y los dispositivos asociados a la raspberry pi. . En este archivo recibe los mensajes vinculados a la tarjeta openCR y se une a su configuración programada en “Arduino” previamente mediante comandos de diagnóstico definidas en la función void setup y asociadas mediante el archivo diagnostic.cpp.

```
int main(int argc, char **argv)
{
  ros::init(argc, argv, "turtlebot3_diagnostic");
  ros::NodeHandle nh;

  tb3_diagnostics_pub  = nh.advertise<diagnostic_msgs::DiagnosticArray>("diagnostics", 10);
  tb3_version_info_pub = nh.advertise<turtlebot3_msgs::VersionInfo>("version_info", 10);

  ros::Subscriber imu         = nh.subscribe("imu", 10, imuMsgCallback);
  ros::Subscriber lds         = nh.subscribe("scan", 10, LDSMsgCallback);
  ros::Subscriber tb3_sensor  = nh.subscribe("sensor_state", 10, sensorStateMsgCallback);
  ros::Subscriber version     = nh.subscribe("firmware_version", 10, firmwareVersionMsgCallback);

  ros::Rate loop_rate(1);

  while (ros::ok())
  {
    msgPub();
    ros::spinOnce();
    loop_rate.sleep();
  }

  return 0;
}
```

Figura 48. Nodo de unión entre la Raspberry pi y el sensor lds junto con un ordenador.

Fuente Autor.

En la función principal se vincula el nodo de comunicación en ros llamado: "*OnHandle nh*".

Dentro de este archivo se suscribe a las funciones elaboradas en Arduino en el archivo "turtlebot3_core.h" y programadas a la tarjeta openCR. Dentro de las funciones se encuentra: imuMsgCallback, LDSMsgCallback, sensorStateMsgCallback y firmware VersionMsgCallback.

Finalmente se realiza un ciclo que lee los mensajes enviados y recibidos y asociados a ros manteniendo comunicación con los archivos que se puedan ejecutar dentro de un ordenador.

Para empezar a programar se empieza instalando el sistema operativo ROS que se permite mediante archivos la lectura, interpretación y programación mediante el ejecutable la elaboración de una tarea definida.

Al instalar ROS se instalan una serie de archivos en los cuales quedan dentro de la memoria

del sistema operativo de Ubuntu. Este archivo instalado dentro de Ubuntu 16.4 que puede ser usados para la lectura de paquetes, nodos, tópicos, servicios, mensajes junto con solicitud y respuesta para finalmente tener parámetros y archivos Launch; su función es unir los parámetros, variables, nodos junto con los tópicos que contendrá el proyecto.

Después de la instalación de ROS se crea un paquete que será el espacio de trabajo del proyecto en el cual contendrá los archivos vinculados al proyecto y que ROS junto con Ubunto podrá leer y utilizar. Dentro de los archivos que contiene el paquete se encuentra los nodos, src o archivo fuente que promueve archivos Python o C; los cuales estos archivos permiten conectar el robot con el ordenador y ser controlado para realizar tareas específicas.

El robot turtlebot se vincula al ordenador mediante la creación de nodos y su vinculación mediante la subscripción y publicación entre los nodos para realizar tareas mediante mensajes y realizar servicios específicos.

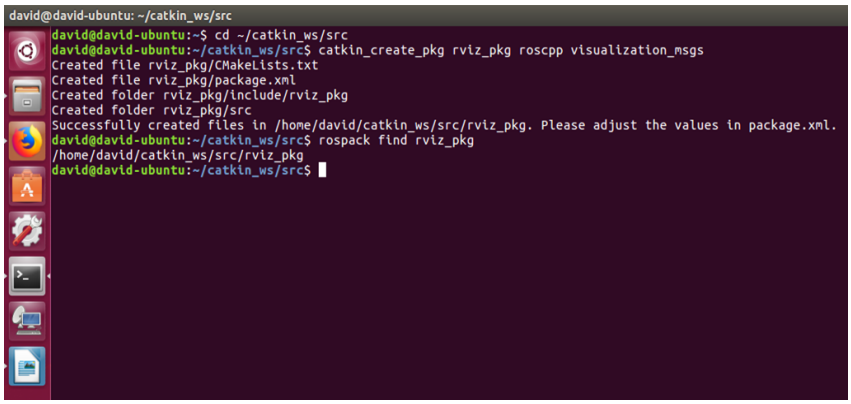

Figura 49. Cargar y programar mediante el ejecutable de Ubuntu. Tomado de Ubuntu 16.4.

Dentro de la carpeta de “src o source” se puede vincular varios paquetes que se vinculan a ROS mediante un archivo con el nombre de Cmake.txt. Este archivo permite encontrar y leer el paquete dentro del Ubuntu. Esto mediante comandos como: rospack list,rospack find[package], rospack depends[package]. La descripción de todos los comandos con paquetes se puede acceder a http://docs.ros.org/independent/api/rospkg/html/rospack.html.

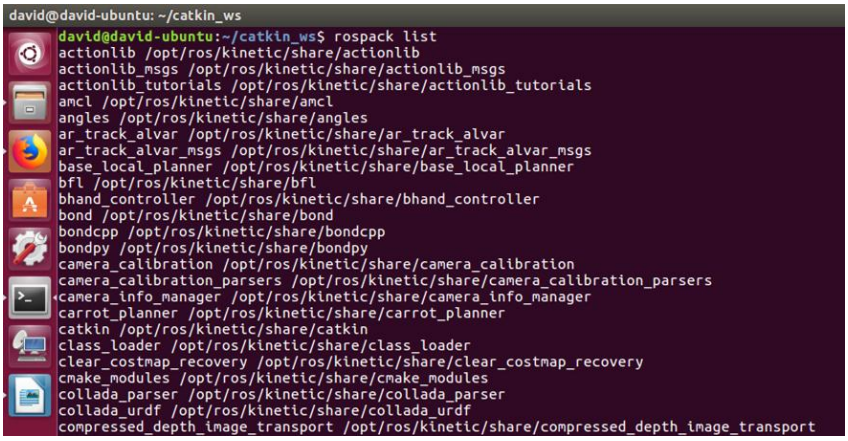

Figura 50. Lectura de los paquetes creados y relacionados con la raspberry pi utilizando comandos de ROS en el ejecutable de Ubuntu. Fuente. Autor.

Luego de creado el paquete se puede acceder mediante el comando: *roscad*.

Luego se vincula al paquete los archivos vinculados a ROS accediendo a la carpeta Cmakelist.tx; dentro de esto se vincula la ruta adecuada para ROS para encontrar el paquete de “catkin” y poderse vincularse al “workspace o espacio de trabajo” donde ROS podrá acceder sin ningún error.

Para realizar la programación se realizan archivos en c++. Esta programación tendrá el análisis de la vinculación entre la lectura de la lectura de los datos del sensor “lds” o sensor laser y la posición del robot dentro del terreno definido de manera hexagonal. El algoritmo buscara trazar una ruta para el robot mediante cálculos matemáticos computacionales en la cual al tener

conocimiento del terreno y de sus obstáculos. Determinará los espacios posibles para poder desplazarse mediante un análisis de color negro o uno para obstáculo y blanco o cero para no obstáculo. Trazará una serie de vectores en la cual mediante suma de trayectorias cortas obtendrá una resultante final. Al tener este resultado se podrá trazar una línea que el robot obtendrá su posición de acuerdo con el mapa que es formada anteriormente y desplazarse cambiando los valores de sus coordenadas utilizando la función "odometry" y aplicar una velocidad a los motores de acurdo a su trayectoria en el cual se dirige.

El cálculo analiza una variable double (x, y) para la variable de inicio y final. Se define las dimensiones en (x, y) del mapa con variables (nx_, ny_ y ns_) en pixeles. Luego de esto se realiza una matriz en (x,y) recorriendo los parámetros que contiene el mapa .

Para analizar el punto final o destino se analiza el costo que este conlleva. Para esto se recorre un vector de n posiciones analizando el costo o valor en celdas que no sean obstáculo para definir cuáles de las celdas o espacios sin obstáculos puede accederse. Todo esto se define a un puntero donde realiza un cálculo de los valores adquiridos y seleccionados para su mejor trayecto. Para esto se lee los valores "heurísticos" h y v analizados teniendo las coordenadas del punto inicial al punto final. Esto analizado desde un plano global.

```
namespace plano_global
class calculador_potencias{
  public:
        calculador_potencias(nx,ny){
              setSize(nx,ny)
        }
        virtual float calculador_potencias(float* potencial,unsigned char costo,int
n,previo_potecial=-1){
```

```
            if(previo_potencial < 0){
                float min_h = std::min( potencial[n - 1], potencial[n + 1] ),
                min_v = std::min( potencial[n - nx_], potencial[n + nx_]);
                    previo_potencial = std::min(min_h, min_v);
            }
             return previo_potencial + costo;
        }


            virtual void setSize(int nx, int ny) {
                nx_ = nx;
                ny_ = ny;
                ns_ = nx * ny;
        }`
    protected:
        inline int toIndex(int x, int y) {
           return x + nx_ * y;
        }


        int nx_, ny_, ns_;
};
}
```

En este código analiza el tamaño del mapa y los puntos de inicio y punto final de recorrido. Realiza una diferencia entre el valor n que es definido como el número de celdas disponibles leídos

y escritos.

Para detectar los obstáculos y definir la mejor ruta se utiliza otro archivo que definí el plano local basado en el algoritmo (DWA) en el cual es unido dentro del archivo final de ejecución del proyecto. (Ver anexo A plano local).

El programa Rviz tiene que tener un código que permite visualizar el robot de manera virtual junto con el mapa que es realizado y mostrado dentro del programa rviz mediante la adquisición de datos por el sensor “lds”. El archivo ve dentro de la carpeta SLAM en la cual se sencuentra dentro del espacio de trabajo. Guardado como frotier_exploration.raviz(Ver anexo B, rviz)

7. Prueba de Resultados y Discusión

7.1 Pruebas de funcionamiento

Para las pruebas de funcionamiento primero el robot hace un reconocimiento del entorno y luego se ejecuta la estrategia de búsqueda de mejor ruta.

7.1.1 Prueba 1.

Para ejecutar la estrategia de búsqueda de mejor ruta primero el robot hace un reconocimiento del entorno, en donde se puede visualizar el espacio de forma hexagonal junto con los obstáculos.

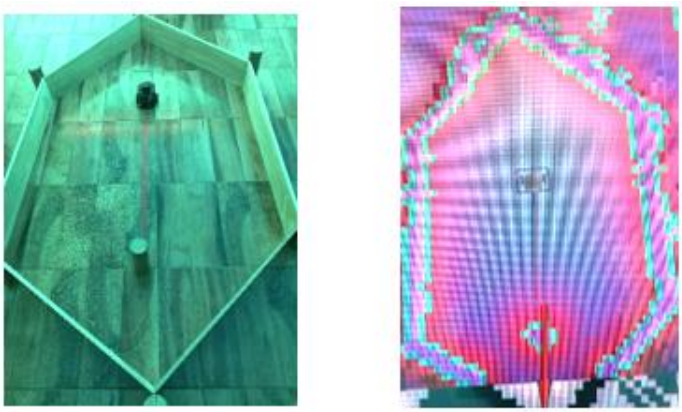

Figura 51. Resultado aplicando el algoritmo A-star(A*) junto con dijktras. Fuente: Autor.

En la figura 51 se puede observar el robot “Turtlebot3 burger” posicionado dentro de una pista hexagonal, en donde el robot realizar un reconocimiento del entorno para ver los límites de la pista y conocer los obstáculos. El reconocimiento del entorno es visualizado utilizando el software libre”Rviz”.

En la primera prueba vemos que al trazar la trayectoria desde un punto inicial a un punto final la trayectoria es una línea recta(figura 52). El robot avanza en línea recta hasta cerca del obstáculo. Cuando el robot está cerca al obstáculo gira de manera que quede en posición final

señalada por la flecha roja.

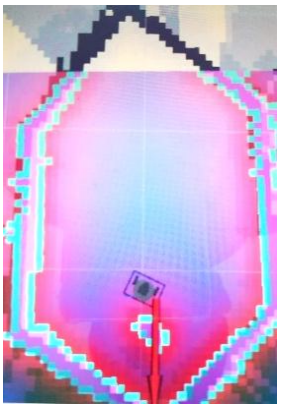

Figura 52. El robot turtlebot3 llega al obstáculo en la posición final. Fuente. Tomada de Ubuntu 16.4.

7.1.2 Prueba 2

En la prueba 2 se analiza los posibles caminos y el comportamiento del robot después de trazarse la mejor ruta desde un punto inicial a un punto final.

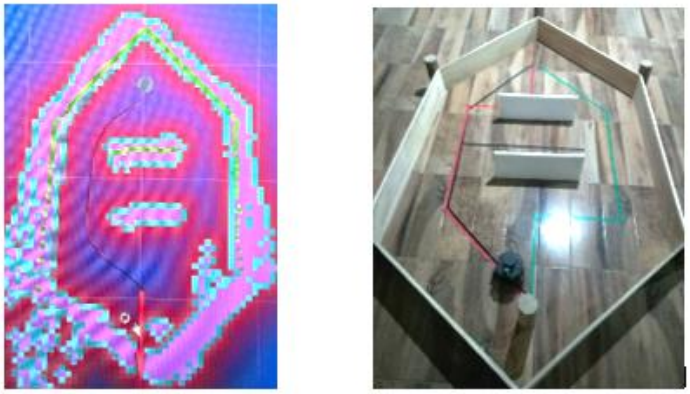

Figura 53. Resultado de la prueba 2. Realizando SLAM (izquierda) y en pista física (derecha). Fuente. Autor.

En la figura 53 vemos que el robot toma el camino de color rojo que tiene menor distancia de recorrido aplicando el algoritmo A-star(A*) junto a el algoritmo “Dijkstra” .El camino verde y

el camino negro son dos posibles caminos que podría tomar el robot, pero son caminos que tomara más tiempo porque requerirá cambiar muchas veces la orientación del robot lo que implica mayor tiempo y más distancia. El robot buscará ir por la mejor ruta partiendo desde una velocidad igual a cero y aumentar hasta velocidad lineal de 0.14 m/s y angular de 0.7m/s.

El robot buscara pasar el obstáculo conduciéndose de manera curva acercándose hacia una esquina más cercana del obstáculo y seguir la trayectoria roja. El robot bajara su velocidad lineal y aumentar moderadamente su velocidad angular medida se acerca al obstáculo.

Figura 54. Trayectoria roja que sigue el robot turtlebot3 burger. Fuente. Autor.

Comparando con el método “breaking ties” con A-star heuclideano el robot turetlebot3 no sigue una línea recta hasta antes de chocar para girar noventa grados para después seguir hasta esquivar el obstáculo y girar nuevamente hasta llegar el punto final. El robot toma tanto antes como después una trayectoria diagonal primero para esquivar lo más pronto posible el obstáculo y luego llegar lo más pronto al objetivo final. Después de esquivar el obstáculo el robot toma una velocidad que no le permite seguir exactamente la línea roja de manera recta pero que busca seguir la

trayectoria roja y llegar al punto final de la mejor manera.(figura 54)

7.1.3 Prueba 3

La prueba número tres costas de dos obstáculos en forma de lámina en paralelo con distancias iguales entre láminas y las paredes del hexagonal. El robot podrá decidir la ruta más corta dependiendo de la posición del robot y los obstáculos alrededor.

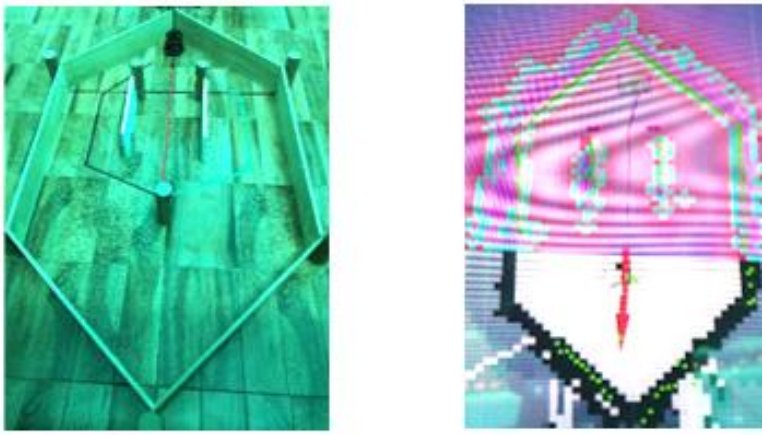

Figura 55. Resultado de la prueba 3. Pista física (izquierda) y realizando SLAM(derecha).Fuente. Autor.

En la prueba 3 vemos que pueden existir tres caminos para escoger el robot "turtlebot3 burger"(figura 3). Los caminos son los que son de color rojo, verde y negro. El camino rojo es el camino más corto y el cual dura menos tiempo(figura 56). El camino de color negro y verde tardaría más tiempo para llegar al objetivo final porque tendría muchos cambios en velocidad lineal y angular del robot durante su trayecto.

El robot sigue la línea roja según se muestra en la figura derecha. El robot empieza en reposo con una velocidad igual a cero. El robot aumenta su velocidad lineal buscando seguir la línea roja trazada y busca estar a una posición no muy cerca a los obstáculos que son localizados

continuamente por los puntos verdes alrededor aplicando el algoritmo DWA. El robot al llegar al objetivo final busca la posición ultima la cual queda en posición a la indicada por la flecha roja con velocidad lineal y angular a cero.

Figura 56. El robot turtlebot3 llega al obstáculo siguiendo la trayectoria más corta. Fuente. Autor.

El robot buscara seguir la trayectoria analizando desde el punto de partida las celdas o nodos que guie hacia el punto final. La celda analizara el valor h y g según sea para encontrar el valor f siguiendo el algoritmo A-star.

7.1.4 Prueba 4

La prueba número cuatro consta de seleccionar un punto final distinto al tomado en la prueba anterior. Con base a esto se analizará el primer y segundo camino más corto que el robot podría tomar.

Figura 57. Resultado de la prueba 4. Ruta corta de color rojo para llegar a la cruz (izquierda) y ruta más corta con SLAM (derecha).Fuente: Tomado de Ubuntu 16.4.

En la prueba número cuatro vemos que la ruta más corta es de color rojo (figura 57). La segunda más corta es de color verde y la última ruta es de color negro. Cuando es definido el punto donde se encuentra la cruz roja el robot toma la ruta de color rojo porque no existe un obstáculo que impida que pueda trazarse y el robot pueda avanzar por ese camino (figura 58).

Figura 58. El robot turtlebot3 llega a la cruz roja siguiendo la trayectoria más corta. Fuente. Autor.

En este resultado el robot empieza con una velocidad lineal y angular igual a cero. El robot t aumenta su velocidad lineal y angular siguiendo la línea azul que define el "quaternio" del robot

por donde el robot avanzara y rotará buscando seguir la línea roja y finalmente llegar al objetivo o punto final.

Antes de realizar su funcionamiento el robot necesitará hacer un mapeo del terreno, el cual requerirá avanzar por el terreno y formar el área en forma hexagonal. Cuando el robot ha realizado el mapa del terreno en forma hexagonal, posicionamos el robot en el punto inicial y definimos el punto final el cual el algoritmo buscara trazar la ruta más corta.

7.1.5 Prueba 5

En la prueba cinco pondremos un obstáculo por el camino más corto. El robot no podrá avanzar por esta ruta. Con esta ruta obstaculizada el robot tendrá que seguir el segundo camino más corto para llegar al objetivo o punto.

Figura 59. Resultado de la prueba 5. Obstáculo en la ruta más corta. Fuente. Autor.

El camino de color rojo esta obstaculizado por dos palos a una altura de 23 centímetros. Los dos palos no permiten el paso del robot y la generación de la ruta por ese camino.(figura 59) El algoritmo A-star(A*) con Dijkstra buscará la segunda ruta más cercana para llegar al objetivo, la

cual será la ruta de color verde. Nótese que el robot buscara llegar al objetivo trazando una ruta que tenga más proximidad al objetivo o punto final, trazando una curva no rectangular (figura 60).

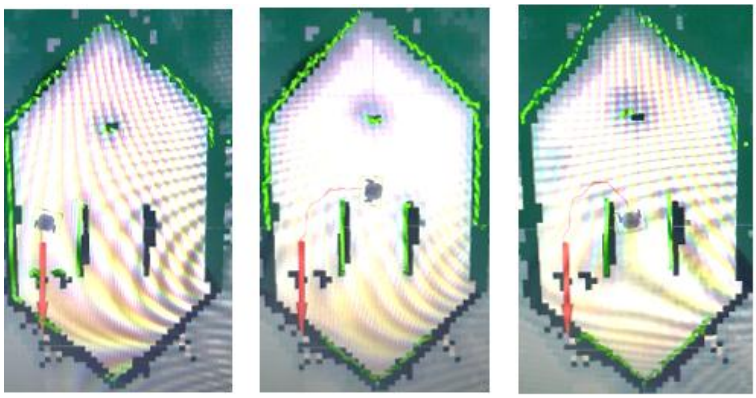

Figura 60. Recorrido del robot turtlebot3 a través segunda ruta más corta utilizando SLAM. Fuente. Autor.

En el trayecto el robot empieza en una posición con velocidad línea y angular igual a cero. Al tener un objetivo o punto final y trazarse el camino por la ruta más corta, el robot turtlebot3 burger aumenta su velocidad lineal y angular hasta permanecer constante. Cuando el robot turtlebot3 burger busca voltear a la izquierda y seguir su ruta, el robot turtlebot3 burger disminuye su velocidad lineal y aumenta su velocidad angular con dirección hacia la izquierda. Cuando el robot gira aumenta su velocidad lineal y angular al mismo tiempo. Finalmente, la velocidad angular tienda a cero siguiendo una velocidad lineal constante hacia su objetivo de manera lineal.

7.1.6. Prueba 6

En esta prueba buscamos realizar colocar varios obstáculos dentro del área hexagonal. Esto permite mediante su funcionamiento buscar la ruta optima, buscar pareces a un pequeño laberinto.

Figura 61. Resultado de la prueba 5 aplicando el algoritmo A-star(A*) junto con dijktras con obstáculos. Fuente. Autor.

En la prueba 6 el robot es colocado en un lugar donde pueda avanzar sin dificultad. El robot puede tener dificultad de movimiento si los obstáculos están muy cerca. El espacio entre obstáculo debe ser algo más del diámetro del robot. Cuanto el espacio es reducido el robot empieza a avanzar y retroceder buscando girar, pero si gira y otro obstáculo puede impedir su avance el algoritmo físico de funcionamiento determinara el detenimiento del avance del robot colocando los motores una velocidad lineal y angular igual a cero.

En la prueba 6 el robot buscara seguir la ruta más corta que permita llegar a la cruz de color rojo. La forma de la trayectoria es de forma zig-zag(Figura 61). Cuando la trayectoria generada es adecuada el robot buscara avanzar de forma lineal a medida que una velocidad angular es aplicada para permitir tomar una curva dentro del camino trazado. La mayoría de las veces seguir la ruta trazada y dependerá de la posición, orientación y sentido del robot. Si el robot se encuentra en una

posición u orientación que sea difícil de avanza tanto lineal como angular, el robot terminara deteniéndose.

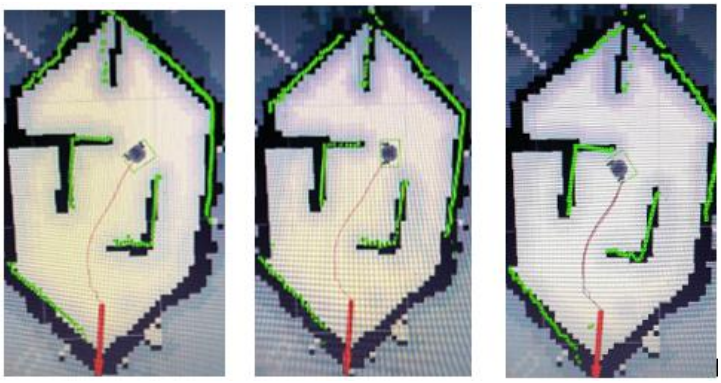

Figura 62. El robot realiza una rotación para cambiar su sentido. Fuente. Autor.

El robot empieza en una posición donde la velocidad angular y lineal es cero. Cuando la ruta más corta es trazada el robot avanza con velocidad lineal constante. Cuanto el robot necesita girar la velocidad lineal disminuye a cero y la velocidad angular aumenta dándole un sentido que el robot pueda seguir. Cuando el robot tiene una orientación la cual puede avanzar de manera lineal, el robot avanza con velocidad lineal y angular constante buscando llegar al objetivo o punto final. El robot puede tener inversión o cambio de giro de los motores para tener desplazamiento hacia atrás si sobre el trayecto permite agilizar su avance(figura 62).

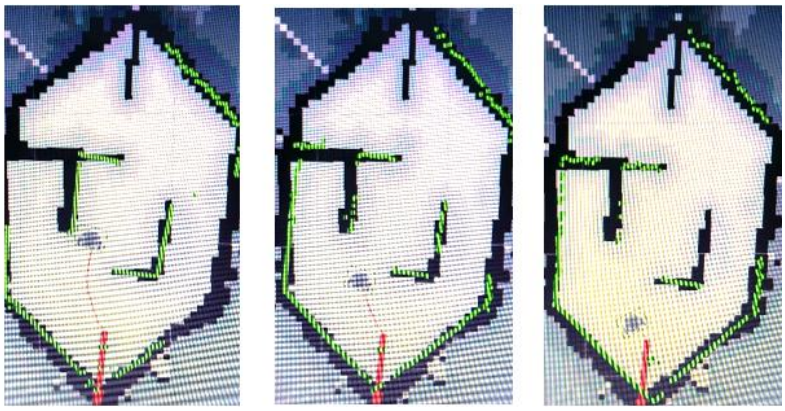

Figura 63. El robot turtlebot3 llega a la cruz roja siguiendo la trayectoria más corta en zig-zag. Fuente. Autor.

El robot llega finalmente a la cruz roja. El robot busca seguir la trayectoria y a medida que avanza la trayectoria roja es más corta trazándose desde la posición actual desde donde se encuentra robot turtlebot3 burger hacia el objetivo o punto final(figura 63).

7.1.7 Prueba 7

En la prueba 7 realizaremos análisis de otras vistas o posiciones de la cámara sobre el terreno usando el programa “Rviz”. Además, analizaremos la trayectoria con un obstáculo rectangular dentro un hexagonal. Mediante este ejercicio podemos analizar la ruta más corta y comparar con la ruta generada en simulación utilizando el robot turtlebot3 burger.

Figura 64. Vista en tres dimensiones del terreno hexagonal con obstáculos (derecha) y la pista en físico(izquierda). Fuente. Autor.

En el programa "Rviz" es posible configurar la vista en tres dimensiones escogiendo la opción "XYOrbit(rviz)" o "Orbit" (figura 65). Con esta vista y utilizando shif junto con click derecho es posible poder poner la cámara en la posición deseada(figura 64).

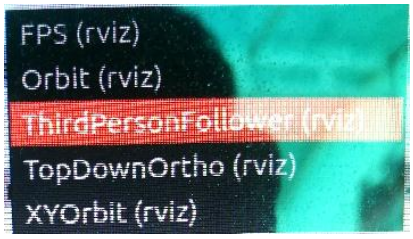

Figura 65. Tipos de vista que permite el programa rviz. Fuente. Autor.

Para definir el objetivo o punto final hacemos click dentro del terreno que anteriormente navegado y finalmente le damos la dirección u orientación final que queremos que el robot termine. Anteriormente deberá seleccionarse "Goal2D" en la parte superior del programa rviz.

Después de determinar la ruta más corta dependiendo y teniendo en cuenta el espacio entre los obstáculos la ruta es trazada y el robot empieza moverse buscando seguir la trayectoria.

El robot siempre buscara poder moverse con dirección lineal buscando llegar de la forma

más rápida a su objetivo.

En esta prueba analizamos un obstáculo dentro del hexágono en el cual permite analizar cual ruta el robot selecciona cuando el obstáculo encierra un espacio por donde podría llegar el robot para luego salirse y tomar la ruta por fuera de este, o incluso desde el principio trazar un camino que este afuera del obstáculo.

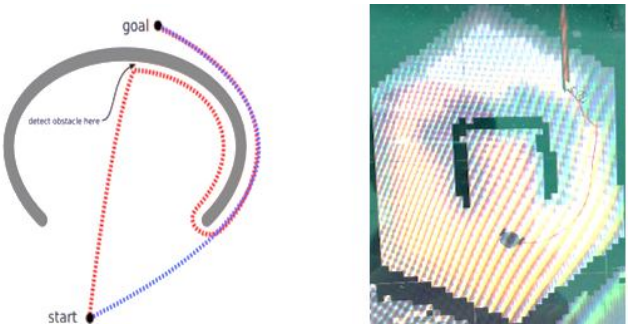

Figura 66. Modelo entre un análisis teoría (izquierda) y practico utilizando el algoritmo A* (derecha). Fuente. Autor.

El robot “turtlebot3 burger” no necesita recorrer una distancia para detectar un obstáculo y tomar una ruta con la forma del obstáculo para finalmente llegar al objetivo final. El algoritmo conoce previamente del obstáculo y genera una trayectoria la cual toma un recorrido por fuera del obstáculo permitiendo ir por una ruta que el robot no se detenga dentro de su recorrido(figura 66).

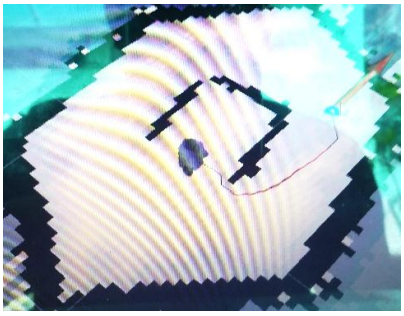

Figura 67. Trazo de ruta con una vista en 3D. Fuente. Autor.

También mediante otra vista podemos ver el trazo de ruta que se servirá de guía para el robot y legar a su objetivo final. La vista permite ver el robot "turtlebot3 burger" junto con la pista hexagonal con los obstáculos de color negro (figura 67).

Cuando el robot avanza y se traslada dentro del espacio junto con el obstáculo, el robot tratara de superar el obstáculo con mayor acercamiento del obstáculo según el algoritmo A-star. En el caso de no poder esquivar el obstáculo por estar dentro del espacio con obstáculos a su alrededor el robot tendrá que retroceder una distancia que le permita salir a un espacio que es mas fasil para avanzar y trasladarse sin quedar detenido dentro una zona. De otra manera el robot terminara deteniéndose sin poder llegar a su objetivo.

El robot turtlebot3 Burger buscara una posición que permita realizar su desplazamiento fácilmente. Cuando la ruta del robot es adecuada y el robot tiene espacio para rotar y tomar el camino con mayor facilidad, avanzado sin dificultades teniendo una velocidad angular y lineal constante antes de superar el obstáculo (figura 68).

Figura 68. Trayectoria del robot turttlebot3 burger para la prueba 7. Fuente. Autor.

La trayectoria es una curva con desplazamiento no rectilíneo con un giro de noventa grados hacia la derecha cuando detecta un obstáculo. En este caso desde el principio de su recorrido gira hacia la derecha de manera que queda en dirección hacia su trayectoria y avanza con una velocidad línea de 0.14m/s y velocidad angular de 0.7m/s. La trayectoria busca siempre ser una curva que es calculada y trazada mediante el algoritmo DWA.

8. Costos

El costo del proyecto abarcara el valor y la cantidad requerida para la realización del proyecto. En la siguiente lista se muestra los elementos primordiales para la realización del proyecto.

Nota: en este listado no se muestra el valor de traída del robot turtlebot3 burger.

Tabla 2.

Costos generales del proyecto.

Descripción	Cant.	V. Unit.	V. Total
Robot turtlebot 3 burger	1	$1,695,998.25	$1,695,998.25
Tablas más corte a 45°	6	-	$85,000
Lamina de icopor de 2 cm	1	$5,000	$5,000
Cinta de enmascarar	3	$1,000	$3,000
Palo redondo	1	$10,000	$10,000
		Total	$1,798,998.25

Fuente. Autor.

Conclusiones

- El algoritmo A* tiene un óptimo desarrollo y fácil programación, ya que matemáticamente calcula un menor trayecto en comparación a algoritmos genéricos o métodos de campos de potencial los cuales calculan campos imaginarios de repulsión, por lo tanto, estos no tienen la misma eficiencia del algoritmo A*.

- Al aplicarse los algoritmos A-STAR y Dijkstra permitió planear la trayectoria, en la cual, las aproximaciones dinámicas de Windows(DWA) hacen que el robot busque una ruta que pueda seguir la trayectoria planeada por el algoritmo.

- Durante las pruebas de funcionamiento con el robot, se tiene una permanente comunicación con la base de algoritmos construida en el computador, de tal forma que sea optimizada la trayectoria por medio de la ubicación y mapeo del entorno.

- Para que el robot tuviera mejor traslación y movilidad dentro del terreno, el robot realiza actualización de su entorno cada 2 milisegundos, posicionandose en el terreno para llegar a un punto de destino eludiendo los obtaculos en el menor tiempo.

Recomendaciones

- Se recomiendo utilizar la versión de Ubuntu 16.4. Las versiones anteriores o superiores posiblemente no lean los paquetes y archivos asociados a ros para el adecuado funcionamiento del robot “TURTLEBOT3 BURGER”
- En la programación se recomienda utilizar programación orientada a objetos en c, la cual variables, librería y archivos puedan asociarse a los archivos de ROS.
- Se recomienda seguir comandos asociados por ROS dentro de Ubuntu 16.4 para poder realizar los paquetes y archivos “launch” que se asocian al ordenador junto con el robot. Es preferible no saltarse los pasos para realizar la programación en el ejecutable de Ubuntu 16.4 porque esto generaría errores del funcionamiento general del robot y el ordenador.
- Para el funcionamiento adecuado del robot “TURTLEBOT3 BURGER” se recomienda la lectura previa del manual general de funcionamiento que se encuentra en la página de ROS. Además, se recomienda seguir las hojas de especificaciones pertenecientes a cada uno de los componentes del robot “TURTLEBOT3 BURGER”.
- Se recomienda utilizar el robot sobre un terreno liso sin ondulaciones. Si se maneja el robot “TURTLEBOT3 BURGER” dentro de un terreno rugoso con ondulaciones podría tener estancamiento en su traslación o rotación sobre su eje. Además, los motores tendrían que hacer más esfuerzo para su movilidad lo que generaría deterioro de los motores.
- Se recomienda colocar la batería de litio que energiza el robot en una posición donde no obstaculice el paso de los cables de los motores dynamixel a la tarjeta openCR. Si la batería obstaculiza los cables puede ocasionar la desconexión del estos.
- Utilizar el robot en zonas poco húmedas o de baja temperatura. Si se encuentra en un ambiente muy caluroso o muy húmedo el probable que se recaliente las tarjetas de procesamiento

o los motores dynamixel.

– Se recomienda no mover demasiado las llantas que se encuentran conectadas a los motores dynamiel. Esto podría desajustar y desprenderse los tornillos pequeños que une las llantas a los motores.

Referencias Bibliográficas

Brillian Math and science. (28 de Junio de 2018). Obtenido de A* search: brilliant.org/wiki/a-star-search/

Ackerman, E. (03 de 26 de 2013). *Que es Turtle?* Obtenido de turtlebot.com: https://www.turtlebot.com/about/

Amits Thoughts on pathfinding. (2019). *Stanford CS theory*. Obtenido de theory.stanford.edu: http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html

Barrientos, A., Peñil , L., Balaguer, C., & Aracil, R. (2006). *Fundamentos de Robotica* (Vol. 2). McGrawHill.

Brilliant. (2019). *The A* Algorithm.* Obtenido de brilliant.org: https://brilliant.org/wiki/a-star-search/

contreras, L. (12 de 18 de 2013). *Raspberry py*. Obtenido de https://histinf.blogs.upv.es/2013/12/18/raspberry-pi/

Fox, D., Burgard, W., & Thrun, S. (2017). *The dynamic Window approach to collision Avoidance*. Obtenido de https://pdfs.semanticscholar.org/cd9e/0901a07262db18fb249efb82094e7c266527.pdf

Frost, D. (2017). *How To Setup and Install Ubuntu Mate 16.04.2 Lts On Raspberry Pi 3 Or 2.* Obtenido de techwiztime.com: https://techwiztime.com/article/setup-install-ubuntu-mate-16-04-2-lts-raspberry-pi-3-2/

Garage, W. (21 de 03 de 2019). *Willow Garage*. Obtenido de http://www.willowgarage.com/pages/software/ros-platform

GeeksforGeeks. (s.f.). *Dijkstra's shortest path algorithm | Greedy Algo-7.* Obtenido de geeksforgeeks.org: https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/

Generation Robots. (2015). *Turtlebot-2 (2015 kit).* Obtenido de generationrobots.com: https://www.generationrobots.com/en/401271-turtlebot-2-assembled-clearpathrobotics.html

Hernández, S. R., Fernández, C. C., & Baptista, M. d. (2014). *Metodología de la Investigación.* Bogotá: Mcgraw Hill. Quinta Edición.

Huang, A. T. (2018). *Rapidly Exploring Random Tree (RRT) and RRT*.* Obtenido de http://demonstrations.wolfram.com: http://demonstrations.wolfram.com/RapidlyExploringRandomTreeRRTAndRRT/

Introduction to A*. (1997). *Algorithms.* Obtenido de theory.stanford.edu: http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html

Julius Fusic, S., Ramkumar, P., & Hariharan, K. (2018). Path planning of robot using modified dijkstra Algorithm. *IEEE*.

Kosari , A., & Teshnizi , M. (2018). Optimal Trajectory Design for Conflict Resolution. *Proceedings of the 6th RSI,International Conference on Robotics and Mechatronics (IcRoM 2018) IEEE*, 23-25.

Koz´ak, V. (2013). *Local planning for a mobile robot.* Obtenido de dspace.cvut.cz: https://dspace.cvut.cz/bitstream/handle/10467/61167/F3-BP-2014-Kozak-Viktor-prace.pdf

Kozak, V. (Junio de 28 de 2013). *Local planning for a mobile robot*. Obtenido de dspace.cvut.cz: https://dspace.cvut.cz/bitstream/handle/10467/61167/F3-BP-2014-Kozak-Viktor-p

Mahdavi, A., & Carvalho, M. (2018). Optimal Trajectory and Schedule Planning for Autonomous Guided Vehicles in Flexible Manufacturing System. *IEEE*.

Open source Robotics Foundation. (2019). *ROS global planner*. Obtenido de wiki.ros.org: http://wiki.ros.org/global_planner?distro=melodic

Parungao, L., Hein, F., & Lim, W. (2018). *Dijkstra Algorithm Based Intelligent Path Planning With topological map and wireless communication.* Obtenido de arpnjournals.org: http://www.arpnjournals.org/jeas/research_papers/rp_2018/jeas_0418_6995.pdf

Parungao, L., Hein, F., & Lim, W. (4 de 8 de 2018). *DIJKSTRA ALGORITHM BASED INTELLIGENT PATH PLANNING WITH TOPOLOGICAL MAP AND WIRELESS COMMUNICATION*. Obtenido de www.arpnjournals.org/jeas/research_papers/rp_2018/jeas_0418_6995.pdf

Patel, A. (22 de 02 de 2019). *Introducción a A**. Obtenido de http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html

Quintana, P. D. (2016). *blogspot.com*.

Raspberry Pi. (2013). *Raspberry Pi.* Obtenido de histinf.blogs.upv.es: https://histinf.blogs.upv.es/2013/12/18/raspberry-pi/

Raw.githubusercontent. (s.f.). *Package opencr index.json.* Obtenido de Raw.githubusercontent.com: https://raw.githubusercontent.com/ROBOTIS-GIT/OpenCR/master/arduino/opencr_release/package_opencr_index.json

Restrepo, J. C., Villegas, J., Arias, A., Serna, S., & madrigal, C. (2012). Trajectory Generation for a Robotic in a Robocup Test Scenery using Kalman filter and B-Spline curves. Antioquia, Colombia: IEEE.

Robotis. (2010). *Robotis e-Manual v1.31.30.* Obtenido de support.robotis.com: http://support.robotis.com/en/product/controller/opencr.htm

Robotis. (2019). *360 Laser Distance Sensor LDS-01 (LIDAR).* Obtenido de robotis.us: http://www.robotis.us/360-laser-distance-sensor-lds-01-lidar/

Robotis. (2019). *EManual TurtleBot 3.* Obtenido de http://emanual.robotis.com/docs/en/platform/turtlebot3/opencr_setup/: http://emanual.robotis.com/docs/en/platform/turtlebot3/opencr_setup/

Robots Components. (2016). *TurtleBot 3 Burger.* Obtenido de roscomponents.com: https://www.roscomponents.com/en/mobile-robots/214-turtlebot-3.html

Romero, C., Vázques, F., & de Castro, C. (2010). *Domotica e inmótica, viviendas y edifcios inteligentes.* RA-MA Alfaomega.

Trapi, R. (18 de 12 de 2017). *Ros*. Obtenido de programacionextrema.es: http://programacionextrema.es/2017/12/08/ros/

willow Garage. (2015). *ROS.* Obtenido de willowgarage.com: http://www.willowgarage.com/pages/software/ros-platform

Wolfram demostration project. (2018). *Rapdly exploring randon tree*. Obtenido de http://demonstrations.wolfram.com/RapidlyExploringRandomTreeRRTAndRRT/

Works on Arm. (s.f.). *Ros.* Obtenido de worksonarm.com: https://www.worksonarm.com/explore/ros/ros-logo/

Anexos

Anexo A. *Código de Plano Local*.

```
#include <vector>
#include <Eigen/Core>

#include <dwa_local_planner/DWAPlannerConfig.h>

/#include <base_local_planner/map_grid_visualizer.h>
#include <pcl_ros/publisher.h>

#include <costmap_2d/costmap_2d.h>

#include <base_local_planner/trajectory.h>
#include <base_local_planner/local_planner_limits.h>
#include <base_local_planner/local_planner_util.h>
#include <base_local_planner/simple_trajectory_generator.h>

#include <base_local_planner/oscillation_cost_function.h>
#include <base_local_planner/map_grid_cost_function.h>
#include <base_local_planner/obstacle_cost_function.h>
#include <base_local_planner/twirling_cost_function.h>
#include <base_local_planner/simple_scored_sampling_planner.h>

#include <nav_msgs/Path.h>
namespace dwa_local_planner {
        class DWAPlanner {
        public:
        DWAPlanner(std::string name, base_local_planner::LocalPlannerUtil *planner_util);
        ~DWAPlanner() {if(traj_cloud_) delete traj_cloud_;}
        void reconfigure(DWAPlannerConfig &cfg);
        bool checkTrajectory(
        const Eigen::Vector3f pos,
        const Eigen::Vector3f vel,
        const Eigen::Vector3f vel_samples);

base_local_planner::Trajectory findBestPath(
        tf::Stamped<tf::Pose> global_pose,
        tf::Stamped<tf::Pose> global_vel,
        tf::Stamped<tf::Pose>& drive_velocities,
        std::vector<geometry_msgs::Point> footprint_spec);
       double getSimPeriod() { return sim_period_; }
void updatePlanAndLocalCosts(tf::Stamped<tf::Pose> global_pose,
        const std::vector<geometry_msgs::PoseStamped>& new_plan);
```

```
private:

    base_local_planner::LocalPlannerUtil *planner_util_;

    double stop_time_buffer_; ///< tiempo que hace que se detecte  el robot cuando se hacerca a
un obstaculo.
    double pdist_scale_, gdist_scale_, occdist_scale_;
    Eigen::Vector3f vsamples_;

    double sim_period_;///< @brief The number of seconds to use to compute max/min vels for
dwa
    base_local_planner::Trajectory result_traj_;

    double forward_point_distance_;

    std::vector<geometry_msgs::PoseStamped> global_plan_;

    boost::mutex configuration_mutex_;
    pcl::PointCloud<base_local_planner::MapGridCostPoint>* traj_cloud_;
    pcl_ros::Publisher<base_local_planner::MapGridCostPoint> traj_cloud_pub_;
    bool publish_cost_grid_pc_;
    bool publish_traj_pc_;

    double cheat_factor_;

    base_local_planner::MapGridVisualizer map_viz_;

    base_local_planner::SimpleTrajectoryGenerator generator_;
    base_local_planner::OscillationCostFunction oscillation_costs_;
    base_local_planner::ObstacleCostFunction obstacle_costs_;
    base_local_planner::MapGridCostFunction path_costs_;
    base_local_planner::MapGridCostFunction goal_costs_;
    base_local_planner::MapGridCostFunction goal_front_costs_;
    base_local_planner::MapGridCostFunction alignment_costs_;
    base_local_planner::TwirlingCostFunction twirling_costs_;

    base_local_planner::SimpleScoredSamplingPlanner scored_sampling_planner_;
  };
};
```

Para la generación de trayectoria se programa mediante el algoritmo “Bresenham” donde mide la longitud entre dos puntos y permite trazar la ruta entre espacios que es posible acceder el robot.

```
private:
  template<class ActionType>
    inline void bresenham2D(ActionType at, unsigned int abs_da, unsigned int abs_db, int error_b, int offset_a,
                            int offset_b, unsigned int offset, unsigned int max_length)
    {
      unsigned int end = std::min(max_length, abs_da);
      for (unsigned int i = 0; i < end; ++i)
      {
        at(offset);
        offset += offset_a;
        error_b += abs_db;
        if ((unsigned int)error_b >= abs_da)
        {
          offset += offset_b;
          error_b -= abs_da;
        }
      }
      at(offset);
    }

  inline int sign(int x)
  {
    return x > 0 ? 1.0 : -1.0;
  }
```

Anexo B. ***Código Rviz*****.**

```
Class: rviz/Displays
  Help Height: 78
  Name: Displays
  Property Tree Widget:
    Expanded:
      - /TF1/Frames1
      - /TF1/Tree1
      - /Map1/Position1
      - /Global Map1/Costmap1
      - /Global Map1/Planner1
      - /Local Map1/Polygon1
      - /Local Map1/Costmap1
      - /Local Map1/Planner1
      - /Explore Costmap1
      - /Explore Costmap1/Costmap1
    Splitter Ratio: 0.5
  Tree Height: 783
- Class: rviz/Selection
  Name: Selection
- Class: rviz/Tool Properties
  Expanded:
    - /2D Pose Estimate1
    - /2D Nav Goal1
    - /Publish Point1
  Name: Tool Properties
  Splitter Ratio: 0.588679016
- Class: rviz/Views
  Expanded:
    - /Current View1
  Name: Views
  Splitter Ratio: 0.5
- Class: rviz/Time
  Experimental: false
  Name: Time
  SyncMode: 0
  SyncSource: ""
Visualization Manager:
  Class: ""
  Displays:
    - Alpha: 0.5
      Cell Size: 1
      Class: rviz/Grid
      Color: 160; 160; 164
      Enabled: true
      Line Style:
        Line Width: 0.0299999993
        Value: Lines
      Name: Grid
      Normal Cell Count: 0
      Offset:
        X: 0
        Y: 0
        Z: 0
      Plane: XY
      Plane Cell Count: 100
      Reference Frame: <Fixed Frame>
      Value: true
    - Alpha: 1
      Class: rviz/RobotModel
      Collision Enabled: false
      Enabled: true
      Links:
        All Links Enabled: true
        Expand Joint Details: false
        Expand Link Details: false
        Expand Tree: false
        Link Tree Style: Links in Alphabetic
Order
        base_footprint:
          Alpha: 1
          Show Axes: false
          Show Trail: false
        base_link:
          Alpha: 1
          Show Axes: false
          Show Trail: false
          Value: true
        base_scan:
          Alpha: 1
          Show Axes: false
          Show Trail: false
          Value: true
        caster_back_link:
          Alpha: 1
          Show Axes: false
          Show Trail: false
```

```
      Value: true
    imu_link:
      Alpha: 1
      Show Axes: false
      Show Trail: false
    wheel_left_link:
      Alpha: 1
      Show Axes: false
      Show Trail: false
      Value: true
    wheel_right_link:
      Alpha: 1
      Show Axes: false
      Show Trail: false
      Value: true
  Name: RobotModel
  Robot Description: robot_description
  TF Prefix: ""
  Update Interval: 0
  Value: true
  Visual Enabled: true
- Class: rviz/TF
  Enabled: false
  Frame Timeout: 15
  Frames:
    All Enabled: false
  Marker Scale: 1
  Name: TF
  Show Arrows: true
  Show Axes: true
  Show Names: false
  Tree:
    {}
  Update Interval: 0
  Value: false
- Alpha: 1
  Autocompute Intensity Bounds: true
  Autocompute Value Bounds:
    Max Value: 10
    Min Value: -10
    Value: true
  Axis: Z
  Channel Name: intensity
  Class: rviz/LaserScan
  Color: 0; 255; 0
  Color Transformer: FlatColor
  Decay Time: 0
  Enabled: false
  Invert Rainbow: false
  Max Color: 255; 255; 255
  Max Intensity: 13069
  Min Color: 0; 0; 0
  Min Intensity: 28
  Name: LaserScan
  Position Transformer: XYZ
  Queue Size: 10
  Selectable: true
  Size (Pixels): 3
  Size (m): 0.0300000007
  Style: Flat Squares
  Topic: /scan
  Unreliable: false
  Use Fixed Frame: true
  Use rainbow: true
  Value: false
- Class: rviz/Image
  Enabled: false
  Image Topic: /raspicam_node/image
  Max Value: 1
  Median window: 5
  Min Value: 0
  Name: Image
  Normalize Range: true
  Queue Size: 2
  Transport Hint: compressed
  Unreliable: false
  Value: false
- Alpha: 0.699999988
  Class: rviz/Map
  Color Scheme: map
  Draw Behind: false
  Enabled: true
  Name: Map
  Topic: /map
  Unreliable: false
  Use Timestamp: false
  Value: true
- Alpha: 1
  Buffer Length: 1
  Class: rviz/Path
  Color: 0; 0; 0
  Enabled: true
  Head Diameter: 0.300000012
  Head Length: 0.200000003
```

```
    Length: 0.300000012
    Line Style: Lines
    Line Width: 0.0299999993
    Name: Planner Plan
    Offset:
      X: 0
      Y: 0
      Z: 0
    Pose Color: 255; 85; 255
    Pose Style: None
    Radius: 0.0299999993
    Shaft Diameter: 0.100000001
    Shaft Length: 0.100000001
    Topic: /move_base/NavfnROS/plan
    Unreliable: false
    Value: true
  - Class: rviz/Group
    Displays:
      - Alpha: 0.699999988
        Class: rviz/Map
        Color Scheme: costmap
        Draw Behind: true
        Enabled: true
        Name: Costmap
        Topic:
/move_base/global_costmap/costmap
        Unreliable: false
        Use Timestamp: false
        Value: true
      - Alpha: 1
        Buffer Length: 1
        Class: rviz/Path
        Color: 255; 0; 0
        Enabled: true
        Head Diameter: 0.300000012
        Head Length: 0.200000003
        Length: 0.300000012
        Line Style: Lines
        Line Width: 0.0299999993
        Name: Planner
        Offset:
          X: 0
          Y: 0
          Z: 0
        Pose Color: 255; 85; 255
        Pose Style: None
        Radius: 0.0299999993
        Shaft Diameter: 0.100000001
        Shaft Length: 0.100000001
        Topic:
/move_base/DWAPlannerROS/global_plan
        Unreliable: false
        Value: true
    Enabled: true
    Name: Global Map
  - Class: rviz/Group
    Displays:
      - Alpha: 1
        Class: rviz/Polygon
        Color: 0; 0; 0
        Enabled: true
        Name: Polygon
        Topic:
/move_base/local_costmap/footprint
        Unreliable: false
        Value: true
      - Alpha: 0.699999988
        Class: rviz/Map
        Color Scheme: costmap
        Draw Behind: false
        Enabled: true
        Name: Costmap
        Topic:
/move_base/local_costmap/costmap
        Unreliable: false
        Use Timestamp: false
        Value: true
      - Alpha: 1
        Buffer Length: 1
        Class: rviz/Path
        Color: 0; 0; 255
        Enabled: true
        Head Diameter: 0.300000012
        Head Length: 0.200000003
        Length: 0.300000012
        Line Style: Lines
        Line Width: 0.0299999993
        Name: Planner
        Offset:
          X: 0
          Y: 0
          Z: 0
        Pose Color: 255; 85; 255
        Pose Style: None
```

```
      Radius: 0.0299999993
      Shaft Diameter: 0.100000001
      Shaft Length: 0.100000001
      Topic:
/move_base/DWAPlannerROS/local_plan
      Unreliable: false
      Value: true
    Enabled: true
    Name: Local Map
  - Alpha: 1
    Arrow Length: 0.300000012
    Axes Length: 0.300000012
    Axes Radius: 0.00999999978
    Class: rviz/PoseArray
    Color: 255; 25; 0
    Enabled: true
    Head Length: 0.0700000003
    Head Radius: 0.0299999993
    Name: Amcl Particles
    Shaft Length: 0.230000004
    Shaft Radius: 0.00999999978
    Shape: Arrow (Flat)
    Topic: /particlecloud
    Unreliable: false
    Value: true
  - Alpha: 1
    Axes Length: 1
    Axes Radius: 0.100000001
    Class: rviz/Pose
    Color: 255; 25; 0
    Enabled: true
    Head Length: 0.300000012
    Head Radius: 0.100000001
    Name: Goal
    Shaft Length: 0.5
    Shaft Radius: 0.0500000007
    Shape: Arrow
    Topic: /move_base_simple/goal
    Unreliable: false
    Value: true
  - Class: rviz/Marker
    Enabled: true
    Marker Topic:
/exploration_polygon_marker
    Name: Exploration Marker
    Namespaces:
      {}
    Queue Size: 100
    Value: true
  - Class: rviz/Group
    Displays:
      - Alpha: 0.699999988
        Class: rviz/Map
        Color Scheme: map
        Draw Behind: false
        Enabled: true
        Name: Costmap
        Topic:
/explore_server/explore_costmap/costmap
        Unreliable: false
        Use Timestamp: false
        Value: true
      - Alpha: 1
        Autocompute Intensity Bounds: true
        Autocompute Value Bounds:
          Max Value: 10
          Min Value: -10
          Value: true
        Axis: Z
        Channel Name: intensity
        Class: rviz/PointCloud2
        Color: 255; 255; 255
        Color Transformer: ""
        Decay Time: 0
        Enabled: true
        Invert Rainbow: false
        Max Color: 255; 255; 255
        Max Intensity: 4096
        Min Color: 0; 0; 0
        Min Intensity: 0
        Name: Explore Boundary
        Position Transformer: ""
        Queue Size: 10
        Selectable: true
        Size (Pixels): 3
        Size (m): 0.00999999978
        Style: Flat Squares
        Topic:
/explore_server/explore_costmap/explore_b
oundary/frontiers
        Unreliable: false
        Use Fixed Frame: true
        Use rainbow: true
        Value: true
```

```
      - Alpha: 1
        Class: rviz/Polygon
        Color: 25; 255; 0
        Enabled: false
        Name: Footprint
        Topic:
/explore_server/explore_costmap/footprint
        Unreliable: false
        Value: false
    Enabled: true
    Name: Explore Costmap
  Enabled: true
  Global Options:
    Background Color: 48; 48; 48
    Default Light: true
    Fixed Frame: map
    Frame Rate: 30
  Name: root
  Tools:
    - Class: rviz/MoveCamera
    - Class: rviz/Interact
      Hide Inactive Objects: true
    - Class: rviz/Select
    - Class: rviz/SetInitialPose
      Topic: /initialpose
    - Class: rviz/SetGoal
      Topic: /move_base_simple/goal
    - Class: rviz/Measure
    - Class: rviz/PublishPoint
      Single click: true
      Topic: /clicked_point
  Value: true
  Views:
    Current:
      Angle: -1.57079637
      Class: rviz/TopDownOrtho
      Enable Stereo Rendering:
        Stereo Eye Separation: 0.0599999987
        Stereo Focal Distance: 1
        Swap Stereo Eyes: false
        Value: false
      Invert Z Axis: false
      Name: Current View
      Near Clip Distance: 0.00999999978
      Scale: 100
      Target Frame: <Fixed Frame>
      Value: TopDownOrtho (rviz)
      X: 0
      Y: 0
    Saved: ~
Window Geometry:
  Displays:
    collapsed: false
  Height: 1027
  Hide Left Dock: false
  Hide Right Dock: true
  Image:
    collapsed: false
  Selection:
    collapsed: false
  Time:
    collapsed: false
  Tool Properties:
    collapsed: false
  Views:
    collapsed: true
  Width: 1279
  X: 0
  Y: 0
```