UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

ESCUELA DE INGENIERÍA EN MECATRÓNICA

TRABAJO DE GRADO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN MECATRÓNICA

TEMA:

"EXPLORACIÓN DE FRONTERAS PARA ROBOTS MÓVILES EN INTERIORES"

AUTOR: JOEL ALEXANDER ALVARADO ONTANEDA

DIRECTOR: CARLOS XAVIER ROSERO CHANDI

IBARRA-ECUADOR
2021

UNIVERSIDAD TÉCNICA DEL NORTE

BIBLIOTECA UNIVERSITARIA

AUTORIZACIÓN DE USO Y PUBLICACIÓN A FAVOR DE LA UNIVERSIDAD TÉCNICA DEL NORTE

IDENTIFICACIÓN DE LA OBRA

En cumplimiento del Art. 144 de la Ley de Educación Superior, hago la entrega del presente trabajo a la Universidad Técnica del Norte para que sea publicado en el Repositorio Digital Institucional, para lo cual pongo a disposición la siguiente información:

DATOS DEL AUTOR			
CÉDULA DE IDENTIDAD	1105997744		
APELLIDOS Y NOMBRES	ALVARADO ONTANEDA JOEL ALEXANDER		
DIRECCIÓN	Av. Guaranda e Isla San Cristóbal		
EMAIL	jaalvaradoo@utn.edu.ec		
TELÉFONO FIJO	-	TELÉFONO MÓVIL	0991029176
DATOS DE LA OBRA			
TÍTULO	"EXPLORACIÓN DE FRONTERAS PARA ROBOTS MÓVILES EN INTERIORES"		
AUTOR	JOEL ALEXANDER ALVARADO ONTANEDA		
FECHA	29/06/2021		
PROGRAMA	PREGRADO		
TÍTULO POR EL QUE OPTA	INGENIERÍA EN MECATRÓNICA		
DIRECTOR	CARLOS XAVIER ROSERO CHANDI		

UNIVERSIDAD TÉCNICA DEL NORTE
FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

CONSTANCIA

El autor manifiesta que la obra objeto de la presente autorización es original y se la desarrolló sin violar derechos de autor de terceros, por lo tanto la obra es original, y que es el titular de los derechos patrimoniales, por lo que asume la responsabilidad sobre el contenido de la misma y saldrá en defensa de la Universidad en caso de reclamación por parte de terceros.

Ibarra a los 29 días del mes de junio de 2021

Joel Alexander Alvarado Ontaneda
C.I.: 1105997744

UNIVERSIDAD TÉCNICA DEL NORTE
FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS
CERTIFICACIÓN

En calidad de director del trabajo de grado "EXPLORACIÓN DE FRONTERAS PARA ROBOTS MÓVILES EN INTERIORES", presentado por el egresado JOEL ALEXANDER ALVARADO ONTANEDA, para optar por el título de Ingeniería en Mecatrónica, certifico que el mencionado proyecto fue realizado bajo mi dirección.

Ibarra a los 29 días del mes de junio de 2021

Carlos Xavier Rosero
DIRECTOR DE TESIS

Agradecimiento

Agradezco a Dios en primer lugar por bendecirme con la vida y permitirme llegar a la culminación de mi carrera profesional.

A mi madre Lady Ontaneda y a mi padre Miller Alvarado por ser quienes me motivaron día a día para seguir estudiando, me guiaron en todo momento y siempre me brindaron su apoyo, aconsejándome para a adelante sin desfallecer durante todo mi período académico, por los consejos que me han brindado para poder solucionar mis problemas y por su comprensión ante los errores que pude cometer.

A mis abuelitas Ermila Quezada y Maura Alvarado y a mi abuelito Segundo Ontaneda por haberme motivado todos los días con sus palabras de aliento y dulzura, quienes siempre han estado dispuestos a contribuir y comprometidos con mi formación personal y profesional.

A mis tíos Afranio Ontaneda, Lenin Ontaneda, Eduardo Ontaneda y mi tía Nilda Ontaneda en quienes siempre pude confiar, me brindaron su apoyo con consejos de situaciones que vivieron ellos y palabras de aliento para poder culminar mi carrera y por su enorme generosidad al disponer sus recursos en mi beneficio, sin los cuáles no habría podido finalizar mis estudios.

A mis primos, Rogger, Javier, Joel, Jorge, Francisco y Patricio, quienes estuvieron conmigo y abrieron las puertas de su hogar para mí, acompañándome durante mi período académico haciendo mi estancia más cálida en esta ciudad.

A mis hermanos, Eric, Isaac y Gabriel por sus cálidas bienvenidas y palabras de aliento que me mostraron en todo momento.

A mi tutor, Ing. Carlos Xavier Rosero, y mis asesores, Ing. Iván Iglesias Navarro e Ing. Luz María Tobar-Subía, por guiarme en el desarrollo de mi tesis, siendo pacientes al hacerme desenvolver para convertirme en el profesional que soy ahora.

Dedicatoria

Esta tesis la dedico a mi familia quienes siempre han estado ahí para mí en todo momento desde que era un niño hasta la actualidad, ellos fueron mi principal motivacion y sin la ayuda de todos ellos no lo hubiera logrado.

Índice general

Índice de figuras

Índice de tablas

Resumen

En la actualidad el uso de robots móviles esta aumentando exponencialmente en todo el mundo y son utilizados en algunas aplicaciones como explorar áreas que son inalcanzables o peligrosas para los seres humanos, motivo por el cual se busca un robot móvil que sea capaz de realizar esta tarea de forma autónoma. Para explorar un área y obtener un mapa se usa algoritmos de exploración que pueden tener diferentes enfoques para seleccionar sus objetivos de navegación, según los parámetros establecidos para su alcance y desplazamiento programados.

La finalidad de este proyecto es desarrollar un algoritmo de exploración basado en fronteras para áreas en interiores y probarlo en un robot móvil Turtlebot 2. Para lograr esto como primer punto se realiza un estudio del estado del arte basado en algoritmos de exploración de fronteras para analizar las diferentes metodologías que se emplean. Y poder optar por un criterio en el desarrollo del algoritmo.

Para demostrar su funcionamiento es necesario usar la técnica de navegación Localización y Mapeo Simultáneo (SLAM, Simultaneos Localization and Mapping) y obtener la odometría del robot, lo cual sirve para la creación de un mapa. Todo esto se desarrolla en el Sistema Operativo de Robot (ROS, Robot Operating System).

Una vez desarrollado el algoritmo se procede a ejecutar pruebas de funcionamiento con dos variantes de un algoritmo de exploración de fronteras. Para ello se ha creado cinco entornos diferentes donde se puede analizar su funcionamiento tomando en cuenta dos parámetros.

Abstract

Currently the use of mobile robots is increasing exponentially around the world and they are used in some applications such as exploring areas that are unreachable or dangerous for human beings, which is why a mobile robot that is capable of performing this task is being sought, autonomously. To explore an area and obtain a map, the exploration algorithms are used that can have different approaches to select your navigation objectives, according to the parameters established for your programmed scope and displacement.

The purpose of this project is to develop a exploration algorithm based in frontiers for indoor areas and test it in a Turtlebot 2 mobile robot. To achieve this, as a first point, a study of the state of the art based on frontiers exploration algorithms is carried out to analyze the different methodologies used. And to be able to choose a criterion in the development of the algorithm.

To demonstrate its operation, it is necessary to use the Simultaneous Localization and Mapping (SLAM) navigation technique and obtain the robot's odometry, which is used to create a map. All this is developed in the Robot Operating System (ROS).

Once the algorithm has been developed, performance tests are carried out with two variants of a frontier exploration algorithm. For this, five different environments have been created where its operation can be analyzed taking into account two parameters.

Introducción

Descripción del problema

El desarrollo de robots móviles responde a la necesidad de extender el campo de aplicación de la Robótica, restringido inicialmente al alcance de una estructura mecánica anclada en uno de sus extremos. Se trata también de incrementar la autonomía limitando todo lo posible la intervención humana [1]. Para los robots móviles la autonomía está estrechamente relacionada con la cartografía, navegación, localización y evasión de obstáculos [2].

En la actualidad los algoritmos de exploración diseñados para robots móviles tienen como objetivo principal de estudio, la creación de un mapa de manera efectiva. Así, la localización y mapeo simultáneo (SLAM, Simultaneous Localization and Mapping) se establece como un método de generación de mapas [3]. Esta técnica usada en robots y vehículos autónomos sirve para construir un mapa dentro de un ambiente desconocido, manteniendo registro de su posición en todo momento [4].

La exploración de fronteras se define como la selección de puntos de destino que producen la mayor contribución a una función de ganancia específica en un entorno inicialmente desconocido [2]. La frontera corresponde a los puntos extremos que se calculan durante la etapa de cartografía y de navegación entre las zonas conocidas y desconocidas. Navegar a un punto de frontera permite la exploración de información invisible en el mapa. Por lo tanto, el avance secuencial a un próximo punto límite define la exploración basada en la frontera [3].

Existen algoritmos de optimización tales como Dijkstra, Bellman-Ford, Floyd-Warshall, Johnson, que se usan para determinar rutas óptimas y no han sido empleadas ampliamente en la exploración de fronteras. Se propone la formulación de un algoritmo sencillo basado en métodos de optimización para exploración de fronteras, corta y libre de colisiones. Se espera que esta solución sirva como apoyo didáctico que permita posteriormente experimentar con diversas técnicas y aplicaciones.

Alcance

Para el desarrollo del presente trabajo es necesario comenzar con la adquisición de conocimientos referentes a la navegación de robots móviles, sus métodos y algoritmos.

Se comprenderá el funcionamiento de los métodos de detección de fronteras y se propondrá

uno, el cual será analizado, ejecutado y probado en el robot móvil Kobuki, usando el Sistema Operativo de Robots (ROS, Robot Operating System). Esta plataforma presenta ventajas como: procesamiento de imágenes, transformación de coordenadas, navegación, control distribuido, intercambio y multiplexación de mensajes de sensores, control, estados, planificaciones y actuadores [5].

Se comparará el algoritmo propuesto con otros algoritmos ya existentes, y se analizará algunas características relevantes como duración de la trayectoria, tiempo computacional y número de colisiones, para así determinar los puntos fuertes y débiles que posee cada algoritmo.

Objetivos

Objetivo general

Desarrollar un algoritmo de exploración de fronteras para un robot móvil en interiores.

Objetivos específicos

- Evaluar algoritmos de exploración de fronteras del estado del arte en un robot móvil.
- Proponer un algoritmo para la exploración de fronteras.
- Implementar el algoritmo propuesto en un robot móvil para la comparación de su desempeño con otros algoritmos.

Estructura del documento

El presente documento está conformado por cinco capítulos.

El Capítulo 1 es la Introducción que explica el motivo del presente trabajo.

El Capítulo 2 presenta una Revisión Literaria de proyectos donde han realizado estudios y pruebas de mapeo usando la selección de fronteras como método principal, dando una perspectiva de la metodología implementada en el algoritmo que se presenta en el siguiente capítulo.

En el Capítulo 3 se describe el flujograma del algoritmo realizado y como es empleado para mapear un área, explicando el proceso que ejecuta. Además presenta las librerías y repositorios utilizados.

En el Capítulo 4 se explica el proceso de la implementación del algoritmo en el robot real y su comunicación entre nodos. Luego se realiza un estudio comparativo a el algoritmo resultante con dos algoritmos existentes, estableciendo los parámetros y escenarios considerados para el estudio.

El Capítulo 5 muestra las conclusiones y recomendaciones que se obtienen del presente trabajo.

1. Revisión Literaria

En este capítulo se presentan algunas metodologías de exploración de fronteras, que pueden ser empleadas tanto en robots como en micro aeronaves. También se explica cómo funciona el método propuesto.

1.1. Estrategias de exploración de fronteras

Una estrategia para explorar lugares desconocidos basado en fronteras es el algoritmo de Triangulación Incremental [6], que consiste en usar la región de visibilidad capturada por sensores y dividir dicha región en una serie de triángulos conectados en un simple pero eficiente árbol estructural como el de la Figura 1.1.

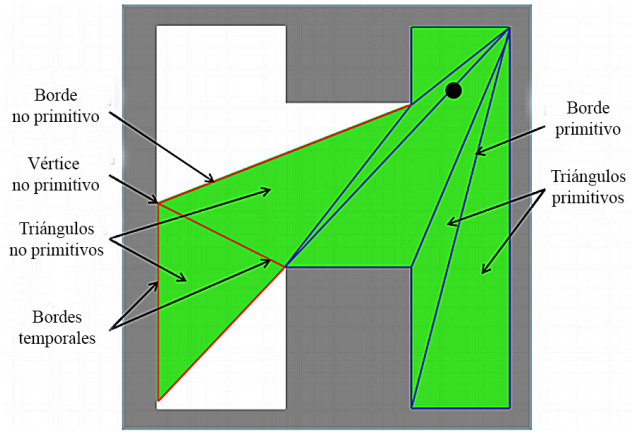

Figura 1.1: Exploración de fronteras mediante triangulación incremental [6]

Los triángulos están divididos en dos tipos: los primitivos que son los que tienen adyacentemente otro triángulo en cada uno de sus lados o bordes límites conocidos y los no primitivos son aquellos que tienen lados adyacentes a áreas inexploradas alrededor, en otras palabras estos contienen al menos un no-límite entre sus tres vértices.

Las áreas inexploradas adyacentes a los triángulos no primitivos llegan a ser las fronteras a explorar y para seleccionar la frontera destino en el caso de que hubiera más de una, se utiliza

parámetros de ganancia y costo. El parámetro costo es simplemente la distancia de viaje requerida para alcanzar su destino, mientras que la ganancia es la información, es decir si una frontera fuera seleccionada como el próximo punto a llegar, la ganancia se puede predecir mediante dos funciones: la primera es el ancho de frontera donde cuanto más amplia más información proporciona y la segunda característica es el número de otras fronteras cercanas.

Existe otro algoritmo de exploración de fronteras que también decide su próximo objetivo dependiendo de los parámetros de costo y ganancia, este utiliza exploración 3D [7] y se lo emplea en un micro vehículo aéreo (MAV).

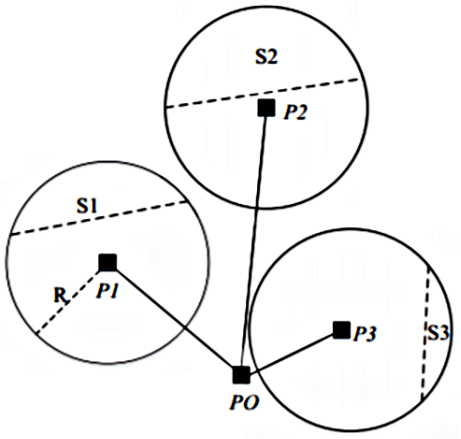

Figura 1.2: Vista de las fronteras a seleccionar por el MAV [7]

En la Figura 1.2 se observa al MAV desde la posición PO, donde $P1$, $P2$ y $P3$ son la fronteras candidatas, sea R el rango del sensor y S la información nueva. De acuerdo con el criterio de costo de viaje, $P3$ es la frontera ideal ya que es la frontera más cercana a PO. En contraste, $P2$ es la opcion a seleccionar por el criterio de ganancia de información esperada, pero en este caso $P1$ es el punto frontera deseado por el criterio de ganancia-costo. La función de evaluación tiene en cuenta los dos factores como resultado, la frontera de meta óptima es la celda en la que puede obtener más información nueva sobre lo desconocido, mientras que el costo de mudarse a él debe ser el más bajo posible.

Para la realización del mapa se clasifica en tres estados: desconocido, libre y ocupado. El espacio desconocido es un territorio que aún no ha adquirido información sensorial; el espacio libre no está ocupado por obstáculos, lo que significa que el vehículo puede moverse en él; el espacio ocupado es la región que contiene obstáculos. Entonces las regiones en el límite entre el espacio libre y el espacio desconocido son las fronteras. Para realizar todo este proceso se utiliza la librería Octomap [8], que sirve para modelado 3D y ayuda con el consumo de memoria al momento de moverse entre fronteras hasta culminar con el territorio deseado.

1.2. Estrategias de exploración de fronteras implementadas en robots móviles

1.2.1. Con un robot

La tarea de exploración puede tener varios objetivos posibles como: la cobertura, la precisión y el mapa. En [9] se usa como estrategia la Exploración Multiobjetivo, este método detecta potenciales destinos, siendo todas las celdas adyacentes a una celda desconocida. Se usa la librería gmmaping [10] para generar un mapa de cuadrículas, se calcula la cantidad de información inexplorada y se mide la localizabilidad.

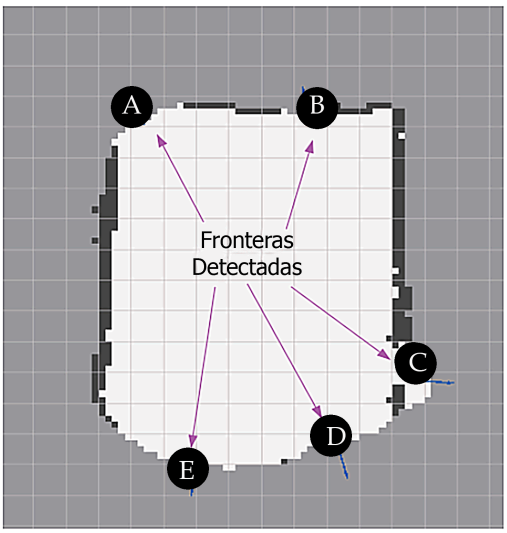

Figura 1.3: Posibles destinos que puede tomar el robot [9]

En la Figura 1.3 se observa un mapa en estado de exploración, el robot fija cinco posibles destinos y la frontera a escoger depende de la función ganancia-costo, es decir, el camino más corto y el destino con mayor información.

Dentro de los requerimientos de exploración en un robot el objetivo es llegar a su destino final usando menor costo, en [11] realizan los movimientos del robot usando la distancia euclidiana [12], con el objetivo de llegar con un movimiento en línea recta a su destino.

Otro método usado es el presentado en [13] que a diferencia del resto no usa una función de ganancia-costo para movilizarse a un punto final, sus movimientos son en zigzag, véase en la Figura 1.4 el movimiento del robot para mapear una área.

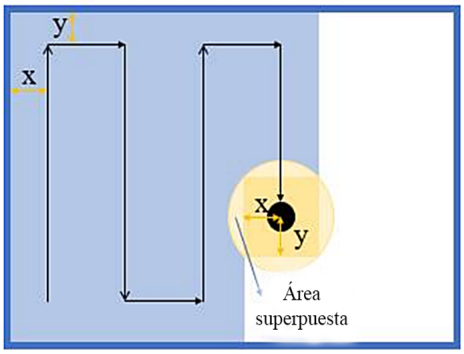

Figura 1.4: Planificación de trayectoria en zigzag [13].

Este algoritmo funciona estableciendo a que dirección puede movilizarse a partir del punto inicial, verificando en cuatro direcciones (izquierda, delantera, derecha, trasera) si existe una frontera a la cual dirigirse mientras establece su dirección, el robot se moviliza hasta llegar al final de la trayectoria y luego procede a evaluar otra ruta hasta completar el mapa del lugar deseado.

1.2.2. Con múltiples robots

Al usar el algoritmo de exploración de fronteras en la mayoría de casos se utiliza mapa de cuadrículas, véase la Figura 1.5, dado que facilita la clasificación de los espacios al nombrarlos como espacio libre (color gris), espacio ocupado (color negro) y espacio no explorado (color blanco) [14].

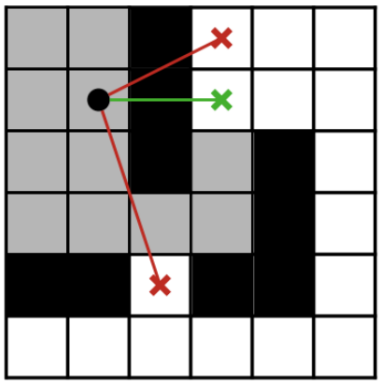

Figura 1.5: Mapa de cuadrícula usado en la exploración de fronteras [15].

En [16] se usa exploración de fronteras y optimizan su algoritmo al utilizar el método de enjambre de partículas, que funciona de tal manera que los robots se dirigen al punto de frontera más cercano verificando que no esté cerca de otro robot, para ejecutar este proceso utilizan un

cálculo de ganancia-costo sobre el área nueva para encontrar la mínima distancia recorrida. Otro método que usa como referencia la ganancia-costo de su trayectoria para moverse es el de Exploración de Frontera Multi-robot [17], aquí la asignación de fronteras es establecida de forma aleatoria en los robots, usando asignación húngara [18], y cada robot representa un espacio ocupado para que de esta manera no se generen colisiones entre ellos.

Otra forma de usar el algoritmo de frontera es usando Inundación de Múltiples Agentes como en [15], este método hace que un robot se dirija a la frontera y en intervalos específicos de tiempo dejan una marca, estas marcas contienen la distancia actual del robot hasta el punto de partida, si un robot encuentra una marca con una distancia mayor que su propia distancia sobrescribe el dato y almacena la marca con el valor más bajo. La distancia almacenada y las marcas de caída permiten a cada robot encontrar el camino más corto conocido desde la posición actual al punto de partida.

1.3. Generalidades del algoritmo propuesto

Tomando como referencia las publicaciones encontradas se procede a realizar un algoritmo de exploración de fronteras utilizando librerías las cuales tienen soporte en cuanto a información confiable preaviamente corroborada, usando como base el algoritmo para navegación autónoma con metodología de detección de fronteras del repositorio *Turtlebot2 autonomous navigation* [19], se propone un algoritmo de exploración de fronteras que evalúe la frontera mejor posicionada utilizando una función de costo-ganancia, para elegir la trayectoria deseada se usa el algoritmo A*, que como heurística de aproximación ejecuta la Distancia Euclidiana; además el robot se moviliza tomando como refencia la distancia media desde su ubicación hasta la frontera y realiza un giro de 360 grados para obtener mayor odometría mientras crea un mapa. Para observar los ejecutables usados en el algoritmo propuesto puede hacerlo en el repositorio *Turtlebot2_frontierexploration* [20].

2. Metodología

En este capítulo se presenta el flujograma del algoritmo propuesto y como el mismo se complementa con el SLAM teniendo como finalidad poder mapear un lugar, además se explica como se realiza cada paso que ejecuta el robot móvil.

2.1. Algoritmo propuesto para detección de fronteras

En la Figura 2.1 se puede observar el flujograma del algoritmo basado en la selección de fronteras y como el mismo se complementa con un algoritmo de optimización.

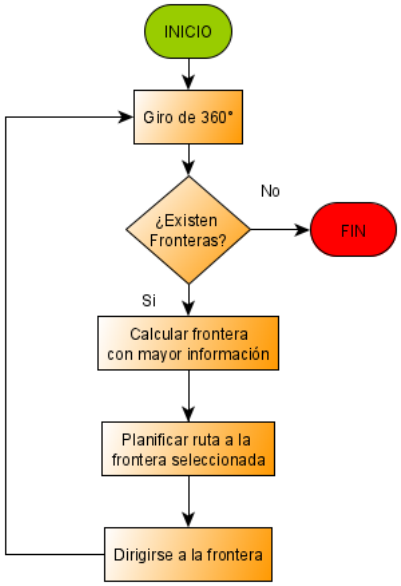

Figura 2.1: Flujograma de metodología de exploración de fronteras

2.1.1. Exploración de fronteras

La exploración basada en fronteras fue desarrollada inicialmente por Yamauchi con el objetivo de obtener la mayor cantidad posible de nueva información sobre un ambiente moviéndose a través de los límites entre el espacio abierto y el área desconocida. Una frontera tiene que ser al menos tan grande como el tamaño del robot [11].

Frontier exploration es un paquete ROS creado para explorar entornos desconocidos y necesita un servidor de mapas en ejecución para actualizarse. La realización de una tarea de exploración de fronteras se puede lograr mediante un área de polígono definida por el usuario a través de RViz o el servicio ExploreTaskAction. El objetivo de exploración contiene un punto inicial para comenzar la exploración y un límite poligonal para limitar el alcance de la exploración [21].

2.1.1.1. Giro del robot

El robot inicia la busqueda de fronteras haciendo un giro de 360 grados, con la finalidad de obtener la mayor información posible desde su ubicación, dado que desde su posición inicial la cámara kinect tiene un rango de visión de aproximadamente 30 grados [22]. En la Figura 2.2 se observa el rango del sensor del robot turtlebot 2.

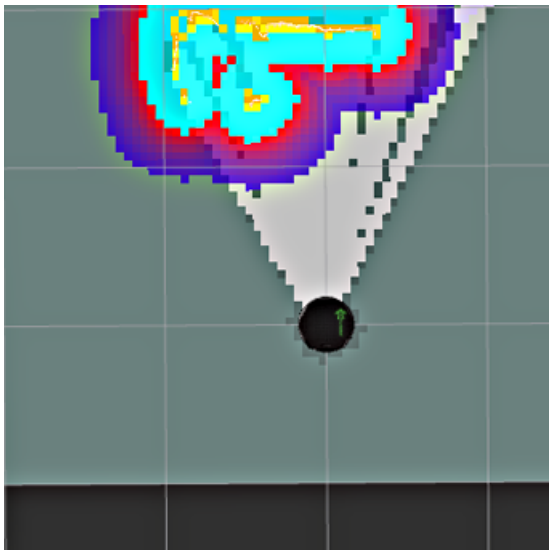

Figura 2.2: Rango de visión de la cámara kinect.

2.1.1.2. Detección de frontera

La asignación de frontera se expresa en cuatro parámetros [11]:

- factor de conglomerado: tamaño de una celda de frontera *i*.

- factor de distancia: distancia euclidiana entre el robot y una celda de frontera *i*.
- factor de liquidación: relacionado con la accesibilidad de una celda de frontera *i*.
- factor de punto inalcanzable: asigna una penalización por celdas fronterizas *i* cerca de las celdas *j* que no se puden alcanzar.

2.1.1.3. Cálculo de frontera

Para calcular la frontera a explorar se usa los 4 parámetros explicados anteriormente, por ejemplo, un gran tamaño de factor de conglomerado indica una mayor probabilidad de que al explorar una región se aumente el conocimiento del mapa; un mayor factor de distancia marca un objetivo que está relativamente cerca del robot; un factor de liquidación alto a menudo sugiere que un objetivo final tiene un camino factible a seguir para ser alcanzado; y un factor alto de punto inalcanzable es una señal de que el destino se encuentra en un área que no es posible alcanzar.

Cada factor recibe sus entradas y sus salidas que se utilizan para calcular la función de utilidad para cada celda *i*. Finalmente, la celda de frontera *i* con la mayor utilidad es calculada por medio de:

$$u_i = k_1.ct_i + k_2.d_i - k_3.ur_i + k_4.cr_i. \tag{2.1}$$

En donde k_1, k_2, k_3 y k_4 son los pesos asociados a cada factor, estos pueden variar según el grado específico de importancia de el entorno determinado. Y ct_i, d_i, cr_i, ur_i son el factor de agrupación, el factor de distancia, el factor de holgura y el factor de punto inalcanzable de una celda de frontera *i* respectivamente [11].

2.1.2. Algoritmo de optimización

El algoritmo de optimización que se usa en este método es el algoritmo Dijkstra. Este algoritmo funciona dado un grafo a cuyos arcos se han asociado una serie de pesos, se define el camino de coste mínimo de un vértice *u* a otro *v*, como el camino donde la suma de los pesos de los arcos que lo forman es la más baja entre las de todos los caminos posibles de *u* a *v*.

El algoritmo de Dijkstra es eficiente, de complejidad $O(n^2)$, donde O es la complejidad y n el número de vértices. Este algoritmo sirve para encontrar el camino de coste mínimo desde un nodo origen a todos los demás nodos del grafo. Fue diseñado por el holandés Edsger Wybe Dijkstra en 1959.

El fundamento sobre el que se asienta este algoritmo es el principio de optimalidad: si el camino más corto entre los vértices *u* y *v* pasa por el vértice *w*, entonces la parte del camino que va de *w* a *v* debe ser el camino más corto entre todos los caminos que van de *w* a *v*, como se observa en la Figura 2.3.

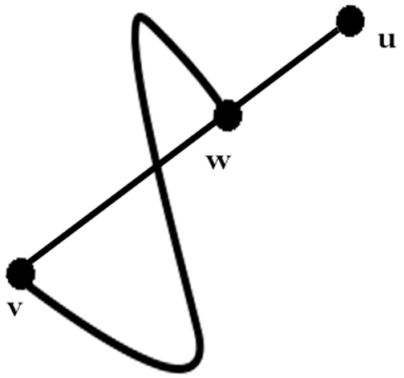

Figura 2.3: Principio de optimalidad [23].

De esta manera, se van construyendo sucesivamente los caminos de coste mínimo desde un vértice inicial hasta cada uno de los vértices del grafo y se utiliza las trayectorias conseguidas como parte de los nuevos caminos [23].

2.1.2.1. Planificación de ruta

Para la planificación de ruta se utiliza el algoritmo A*, basado en el algoritmo de Dijkstra. Para encontrar una ruta óptima el algoritmo A* parte desde un nodo objetivo para realizar un seguimiento de la ruta de costo mínimo, a continuación verifica todos los nodos por los que puede realizar una ruta y cada nodo posible es expandido y almacena los datos con cada nodo sucesor n que sirven para llegar a la ruta de menor costo encontrada hasta el momento.

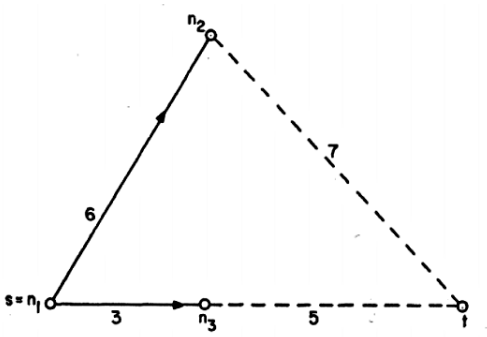

Figura 2.4: Planificación de la ruta mínima [24].

En la Figura 2.4 se puede observar que los cálculos parten de un punto inicial S. El algoritmo debe terminar en este caso en el nodo objetivo t, y no se expanden más nodos. Entonces reconstruye una ruta de costo mínimo de t. En este caso desde el punto inicial S pasando por n_2 hasta el punto final t hay un costo de 6 + 7 = 13, mientras que si recorre por el n_3 hasta t el valor del costo es 8.

Para expandir el menor número posible de nodos en la búsqueda para una ruta óptima, el algoritmo de búsqueda debe tomar una decisión lo más informada posible sobre que nodo expandir a continuación. Si expande nodos que obviamente no pueden estar en un camino óptimo, obtiene una pérdida de esfuerzo [24].

2.1.2.2. Desplazamiento a la frontera

Para desplazarse a la frontera el *Algoritmo A** lo hace usando como heurística de aproximación la *Distancia Euclidiana*, que es la distancia en línea recta entre dos puntos. Para calcularlo se obtiene la hipotenusa de dos puntos dados A y B [25].

$$d_{AB} = \sqrt{(x_A - x_B)^2 + (y_A - y_B)^2}. \tag{2.2}$$

Donde:

- d_{AB} es la distancia entre los puntos A y B
- x_A coordenada x en el punto A
- y_A coordenada y en el punto A
- x_B coordenada x en el punto B
- y_B coordenada y en el punto B

En la Figura 2.5 se puede observar un punto inicial A y un punto final B usando la distancia euclidiana en un mapa de cuadrícula.

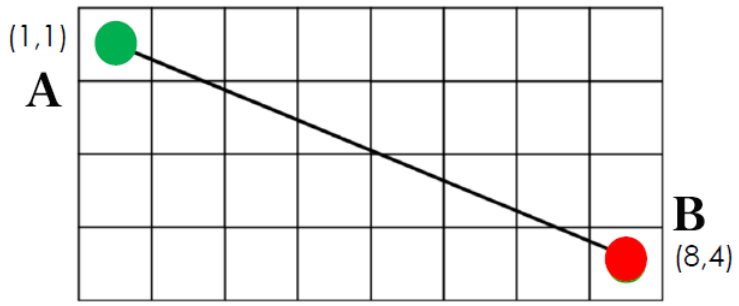

Figura 2.5: Distancia euclidiana.

2.2. Mapeo usando exploración de fronteras

Con la finalidad de conseguir un robot móvil más autónomo se usa el algoritmo de exploración de fronteras en conjunto con un algoritmo de mapeo basado en SLAM. En la figura 2.6 se puede observar el flujograma del algoritmo de mapeo usando la estrategia de exploración de fronteras.

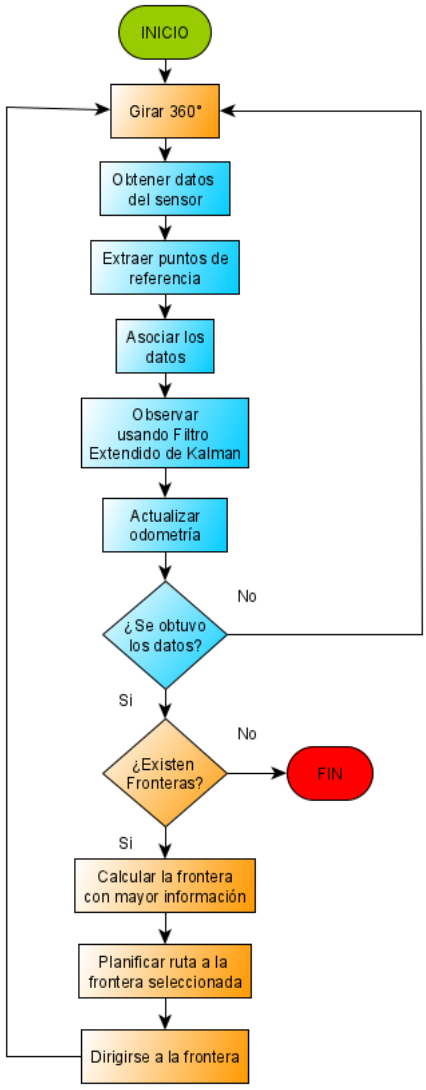

Figura 2.6: Flujograma de mapeo usando exploración de fronteras

2.2.1. SLAM

En la robótica móvil la mayoría de los robots son programados para desplazarse sobre ambientes controlados o conocidos, si se coloca un robot en un entorno diferente al que fue programado el robot puede llegar a estar completamente perdido.

El SLAM propone una solución a esto, al colocar un robot en un entorno desconocido, y que él mismo sea capaz de construir incrementalmente un mapa consistente al mismo tiempo que utiliza dicho mapa para determinar su propia localización [26].

Para navegar correctamente el robot necesita información de su entorno, la cual obtiene de sus sensores. Pero los datos de los sensores están comprometidos debido al ruido. Esto significa que es necesario utilizar algún método de filtrado, un método adecuado es la estimación Bayesiana, que usa modelos probabilísticos para filtrar el ruido.

Para comprender esto se inicia con un robot móvil ubicado en un espacio desconocido empezando desde una localización con coordenadas conocidas X_0.

El robot se desplaza a lo largo del tiempo "t", generando una secuencia de localizaciones "x_t". Esta secuencia proporciona su trayectoria.

$$X_t = \{x_0, x_1, x_2, ...x_t\}. \tag{2.3}$$

Mientras el robot se desplaza va adquieriendo entradas como mediciones de la odometría (u_t) y observaciones del entorno (Z_t), que es la información que establece el robot a través de las mediciones entre las características de m y x_t.

$$U_t = \{u_0, u_1, u_2, ...u_t\}. \tag{2.4}$$

$$Z_t = \{z_0, z_1, z_2, ...z_t\}. \tag{2.5}$$

Se denomina m a el mapa real del entorno. El entorno se compone de puntos de referencia, objetos, superficies, etc, y m describe sus localizaciones. El mapa m se asume que es invariante en el tiempo.

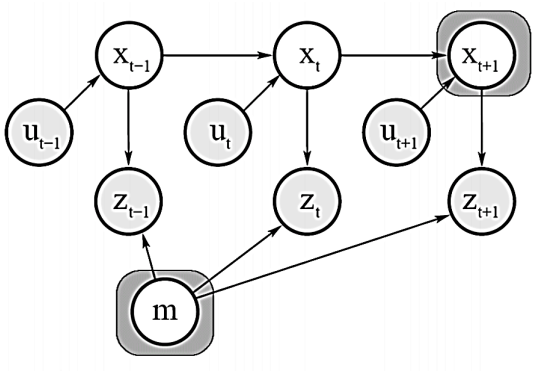

Figura 2.7: Diagrama de SLAM online [26].

En la Figura 2.7 se aprecia como funciona el SLAM online, el método que se usa en este caso. Este SLAM procesa la información en el robot mientas navega en el entorno [27]. Aquí m es el punto de referencia, x_t las posiciones del robot, u_t su odometría y z_t son las observaciones entre la posción de m y x_t. Donde una forma de expresar la estimación de la posición y el mapa mas reciente es:

$$p(x_t, m|z_{1:t}, u_{1:t}). \tag{2.6}$$

Dos modelos matemáticos más son necesarios para resolver los problemas del SLAM. Por un lado, el modelo de observación que relaciona las mediciones z_t al mapa m y a la localización del robot x_t.

$$z_t = h(x_t, m). \tag{2.7}$$

Por otro lado, el modelo de observación que relaciona la odometría u_t con las localizaciones del robot x_{t-1} y x_t

$$x_t = f(x_{t-1}, u_t). \tag{2.8}$$

Todo el procedimiento SLAM realizado por el robot móvil se lo desglosa de la siguiente manera, siguiendo el flujograma descrito anteriormente.

2.2.1.1. Obtención de datos del sensor

Es el paso inicial del programa donde captura datos para ser evaluados y poder obtener información del mapa y la ubicación del robot.

2.2.1.2. Extracción de puntos de referencia

En esta sección el algoritmo busca puntos de referencia que sirvan para facilitar la creación del mapa y poder generar su posición basado en puntos referenciales.

2.2.1.3. Asociación de datos

En esta parte se compara dos tipos de datos, los datos de puntos referenciales que sirven para establecer lugares y distancias específicas y los datos de puntos normales, que se van almacenando en la memoria del robot para poder formar el mapa.

2.2.1.4. Observación

Aquí se usa el Filtro Extendido de Kalman (EKF) que es la metodología necesaria para trabajar cuando se usa SLAM online para reducir los errores en su odometría [27], se encarga de analizar los datos capturados, en caso de no ser un punto referencial o punto observado se toma en cuenta como un punto nuevo de observación que es utilizado posteriormente en caso de ser necesario [26].

2.2.1.5. Odometría

La odometría se calcula según los puntos de observación obtenidos, a medida que se obtengan nuevos puntos de observación la odometría permanece cambiando y a su vez actualizandose para mantener la posición del robot con el mínimo error de ubicación [26].

2.2.1.6. Gmapping

El paquete Gmapping de ROS proporciona SLAM usando un nodo llamado slam_Gmapping. Usando slam_Gmapping se crea un mapa de ocupación 2D (como el plano de un edificio) [28]. Gmapping es un filtro altamente eficiente que desarrolla mapas de cuadrícula a partir de datos de rango láser [29].

- *Tipo de mapa*

Figura 2.8: Funcionamiento del algoritmo SLAM

Al usar el algoritmo SLAM con Gmapping , se utiliza un mapa métrico de rejillas uniformes para que el robot pueda explorar el entorno. El robot ubica los bordes de las zonas inexploradas mostradas en naranja, las celdas exploradas se muestran en blanco y los obstáculos expandidos se muestran en negro, como se observa en la Figura 2.8.

3. Implementación y Resultados

En el presente capítulo se detallan los pasos realizados para la implementación del algoritmo de mapeo en un robot Turtlebot 2 con la finalidad de comprobar su funcionamiento, para analizar su operatividad se realiza una comparación con dos algoritmos que permiten observar su desenvolvimiento con respecto a los mismos.

3.1. Implementación

3.1.1. Puesta en marcha del turtlebot 2

Turtlebot es un robot que consta de una base móvil, un sensor 3D, una computadora y un kit de hardware de montaje como se lo observa en la Figura 3.1, es de bajo coste y con software de código abierto [30]. Turtlebot usa la plataforma Sistema Operativo de Robot(ROS, Robot System Operative) basado en una arquitectura de grafos donde el procesamiento toma lugar en los nodos que pueden recibir, mandar y multiplexar mensajes de sensores, control, estados y planificación de actuadores, más información en [31].

Figura 3.1: Modelo turtlebot 2

3.1.1.1. Testeo de cámara kinect

Primero se debe instalar todos los paquetes necesarios para poder usar Turtlebot con ROS que se encuentran en [32], para verificar que los paquetes necesarios de la cámara kinect están instalados de manera correcta se usa el siguiente comando en un terminal nuevo:

```
roslaunch openni_launch openni.launch
```

Si las líneas resultantes comienzan con proccess[cámara...] como en la Figura 3.2 significa que los paquetes se han instalado correctamente.

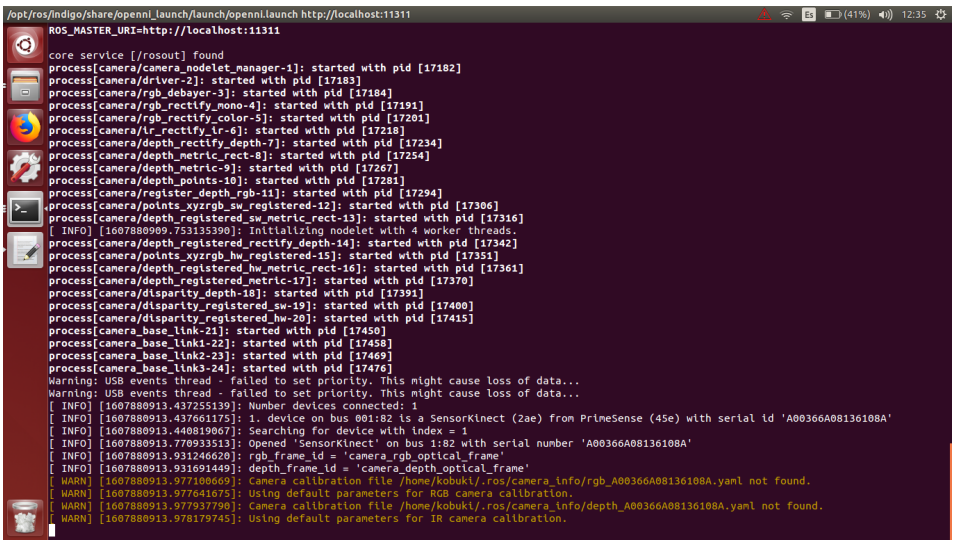

Figura 3.2: Verificación del estado de la cámara

Para probar su funcionalidad se usa el comando:

```
rosrun image_view image_view image:=/camera/rgb/image_color
```

Si la cámara funciona correctamente se puede observar a través del terminal de la computadora lo que observe el kinect, como en la Figura 3.3.

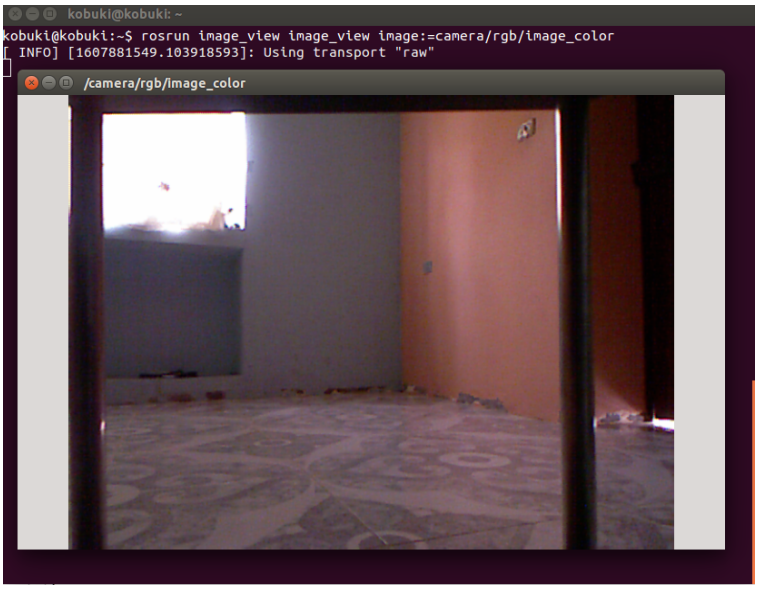

Figura 3.3: Pruebas en cámara

3.1.1.2. Testeo de kobuki

Para verificar que exista la correcta comunicación en la base móvil kobuki se hace un ejercicio simple de teleoperación, usando los comandos encontrados en [33]:

```
roslaunch turtlebot_bringup minimal.lauch
```

```
roslaunch turtlebot_teleop keyboard_teleop.launch
```

En el terminal aparece un tablero de control, véase la Figura 3.4, que indica las teclas que direccionan al kobuki para girar, incrementar o dismuir velocidad.

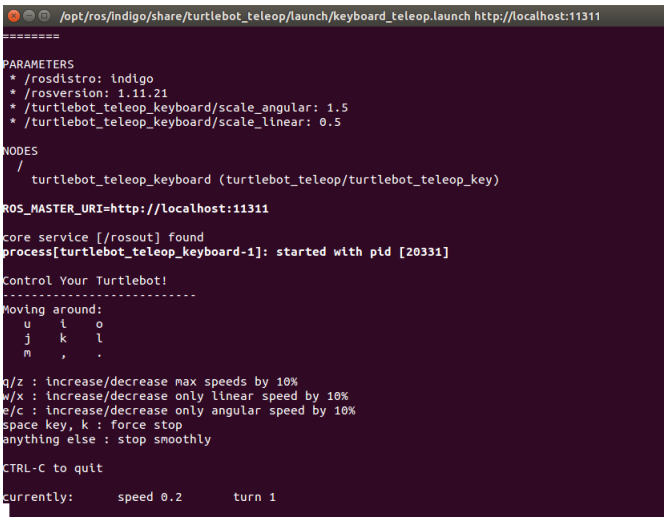

Figura 3.4: Tablero de control para teleoperar el turtlebot 2.

3.1.2. Comandos para mapear un lugar interior usando turtlebot 2

Para poner en funcionamiento el algoritmo realizado en el turtlebot 2, se utiliza seis comandos que se encargan del mapeo, movimiento y cálculo necesario.

- roslaunch turtlebot_bringup minimal.launch

Este comando establece comunicación entre el ordenador y el robot móvil turtlebot 2 y habilita su base móvil.

- roslaunch turtlebot_navigation gmapping_demo.launch

Esta instrucción se encarga de crear el mapa mientras utiliza SLAM en el robot. Tomando en cuenta que este nodo utiliza sensores láser, se utiliza el sensor de una cámara kinect.

- roslaunch turtlebot_rviz_launchers view_navigation.launch

Esta orden despliega una herramienta de visualización 3D que proporciona ROS, aquí progresivamente el mapa se construye a partir de los datos de los sensores.

- roslaunch final_project final_project.launch

Sirve para gestionar el inicio y detección de los ejecutables que se encuentran dentro del repositorio a utilizar que en este caso son mapping.py y control.py.

- rosrun final_project mapping.py

Este comando sirve para llamar al ejecutable que se encarga de la detección de fronteras, espacios libres, espacios ocupados y encuentra los puntos óptimos donde el robot debe dirigirse.

- rosrun final_project control.py

Este nodo se encarga de controlar al subsistema kobuki y en complementación con el otro ejecutable calcula la distancia al punto final y la mejor manera de dirigirse a dicho punto.

Cuando los comandos son ejecutados en el orden mostrado el robot comienza a mapear, inicia realizando un giro de 360 grados y ubica la frontera mas cercana que se encuentre disponible, calcula su distancia media y avanza, repite el mismo procedimiento hasta terminar de cubrir todo el espacio del lugar donde se encuentra, mientras realiza este proceso otro nodo se encarga de guardar su posición y la posición de los demás objetos y se lo puede observar a través de RViz, al final produce un mapa en 2D de la habitación, igual al de la Figura 3.5.

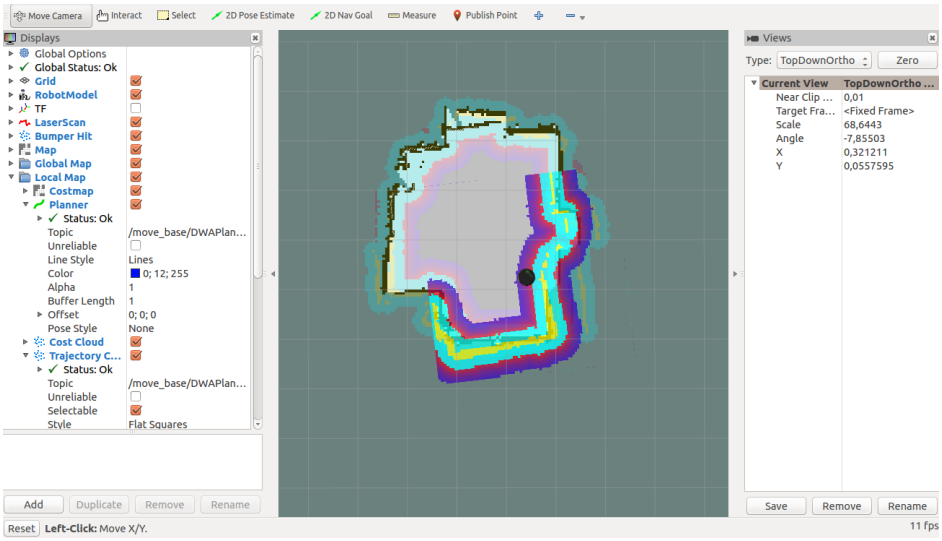

Figura 3.5: Mapa obtenido en RViz.

Cuando el mapeo termina se puede guradar el mapa usando el comando descrito a continuación. El archivo originado se encuentra en formato .pgm [34].

```
rosrun map_server map_saver f /tmp/my_map
```

3.1.3. Comunicación de nodos

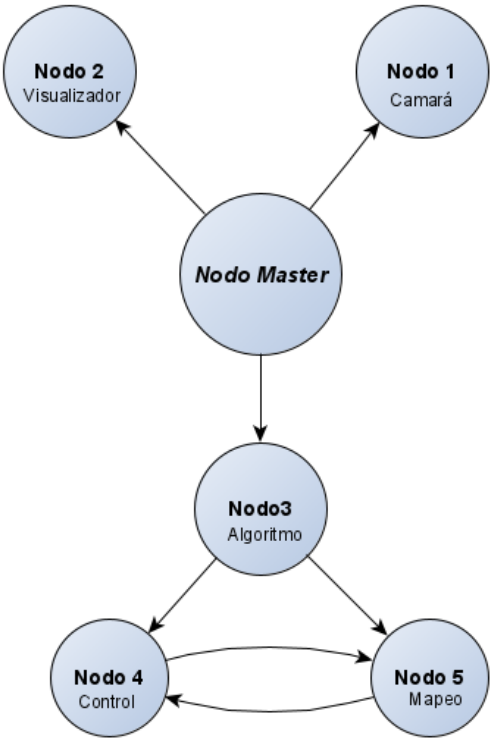

Figura 3.6: Conexión de nodos en ROS

Para que el robot inicie con su objetivo, primero debe tener comunicación con la máquina central que es la computadora, este procedimiento actúa como nodo máster y se encarga de comunicarse con los demás nodos que se observa en la Figura 3.6

- El Nodo 1 ejecuta el paquete para establecer comunicación con el kinect.
- El Nodo 2 compila el paquete del visualizador donde se observa el mapa desde el inicio hasta el final.
- El Nodo 3 ejecuta el repositorio donde se encuentra el código, principalmente los ejecutables control.py y mapping.py
- El Nodo 4 guarda los datos obtenidos del sensor y se encarga de establecer los espacios vacíos, espacios llenos y las fronteras.
- El Nodo 5 controla los movimientos del robot.

3.2. Resultados

3.2.1. Algoritmos de prueba

Para realizar el análisis del funcionamiento del algoritmo propuesto, se utiliza dos algoritmos de exploración de fronteras basados en el algoritmo *Navegación Autónoma con Metodología de Detección de Fronteras* [19], donde la característica principal del primer algoritmo (DFM, Detección de Fronteras de Distancia Máxima) es movilizarse hasta la frontera, a diferencia del algoritmo propuesto que se moviliza hasta una distancia media con respecto a la frontera; el segundo algoritmo (DFS, Algoritmo de Frontera Simplificado) su principal característica es girar 180 grados iniciando desde la derecha, se diferencia del algoritmo propuesto debido a que este hace un giro completo de 360 grados, además este algoritmo se moviliza usando la distacia media de la frontera.

3.2.2. Parámetros de evaluación

A continuación se presentan los parámetros de evaluación elegidos para la comparación del algoritmos basandose en [35] [36]:

Frecuencia de escaneo: Se considera el número de veces que el robot escanea un área, este parámetro sirve para comprender si el tener una mayor cantidad de escaneos ayuda al robot a terminar el mapeo de forma más rápida y precisa.

Duración de la trayectoria: es la cantidad de segundos que el robot usa para completar el mapa, indistintamente de si el mapa se encuentra en buen estado.

Estos parámetros elegidos al complementarse deben presentar un resultado de tal manera que se obtenga un mapa preciso en un tiempo mínimo.

3.2.3. Escenarios de evaluación

Para el análisis de los tres algoritmos se realiza pruebas en cinco escenarios, con la finalidad de medir el tiempo de trayectoria y la la frecuencia de escaneo que requieren. El primer escenario es una habitación vacía, aquí no se ubica ningún obstáculo; el segundo escenario es la terraza de una casa, aquí tampoco hay obstáculos excepto por una columna; el tercer escenario es una habitación que tiene obstáculos cilindricos y obstáculos de prisma rectangular a su alrededor [35]; el cuarto escenario es una habitación que tiene un gran obstáculo frente al robot y este debe rodearlo [37] y el quinto escenario es el pasillo de una casa [38] donde no hay obstáculos. La prueba de los algoritmos en el escenario habitación vacía, con obstáculos y con pared en el centro se realizaron en una habitación de 12 m^2 con una luminiscencia de 810 lumens. El escenario del pasillo se hizo en un área de 6 m^2 con 1100 lumens. El escenario terraza se hizo en una área de 30 m^2 al anochecer. Estas medidas se tomaron en cuenta debido a que el uso de la cámara kinect se lo debe hacer en un ambiente controlado sin exposción a los rayos de sol [39].

3.2.4. Descripción de los resultados

La información expresada en la Tabla 3.1 son el resultado de dos pruebas realizadas por los algoritmos, tomando como datos su mejor resultado. A continuación se presenta el análisis obtenido de las pruebas.

Tabla 3.1: Resultados de los algoritmos en los diferentes escenarios

Escenarios	Parámetros	Algoritmo Propuesto	Algoritmo DFDM	Algoritmo DFS
Escenario habitación vacía	Tiempo	4’ 24”	6’ 50’	4’ 05’
	NºEscaneo	23 veces	35 veces	25 veces
Escenario terraza	Tiempo	9’ 37”	21’ 30”	9’ 45”
	NºEscaneo	59 veces	141 veces	66 veces
Escenario con obstáculos	Tiempo	4’ 01”	9’ 01”	5’ 25”
	NºEscaneo	26 veces	54 veces	33 veces
Escenario con pared en el centro	Tiempo	3’ 50”	7’ 28”	6’ 11”
	NºEscaneo	29 veces	49 veces	40 veces
Escenario pasillo	Tiempo	3’ 02”	4’ 59”	5’ 01”
	NºEscaneo	19 veces	30 veces	34 veces

3.2.5. Resultados del algoritmo propuesto

3.2.5.1. Escenario habitación vacía

Es el único escenario donde no se logra el mejor tiempo de mapeo con respecto a los otros dos algoritmos, pero usa menor frecuencia de escaneo y obtiene el mejor mapa, véase la Figura 3.7.

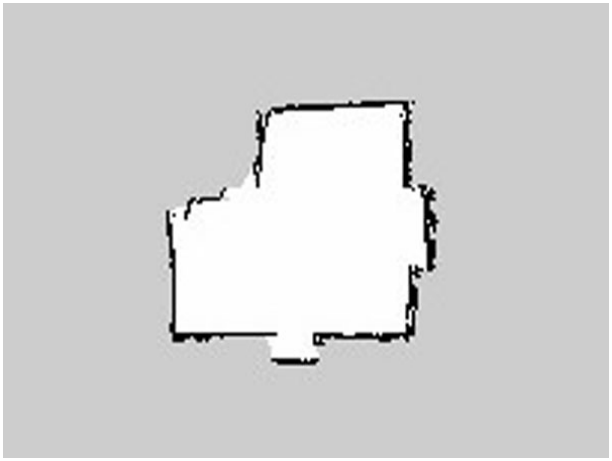

Figura 3.7: Escenario habitación vacía usando el algoritmo propuesto

3.2.5.2. Escenario terraza

El mapeo en este escenario dura más tiempo debido a su dimensión, obtiene el mejor tiempo de todos con una mínima diferencia con el algoritmo DFS, en cuanto al desarrollo del mapa es significativamente bueno como se observa en la Figura 3.8.

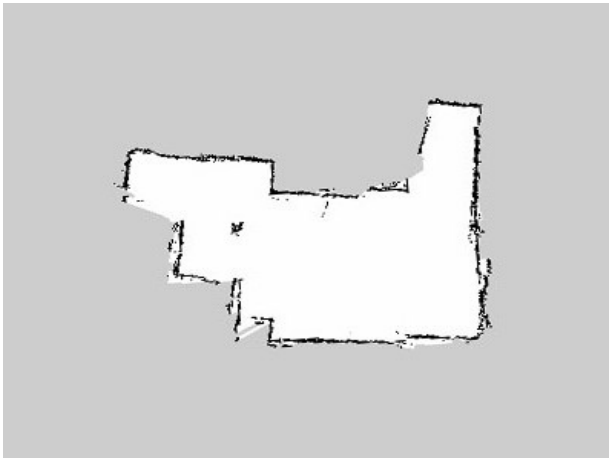

Figura 3.8: Escenario terraza usando el algoritmo propuesto

3.2.5.3. Escenario con obstáculos

Aquí se obtiene un buen tiempo y un menor uso de escaneos, este resultado se puede observar en la Figura 3.9.

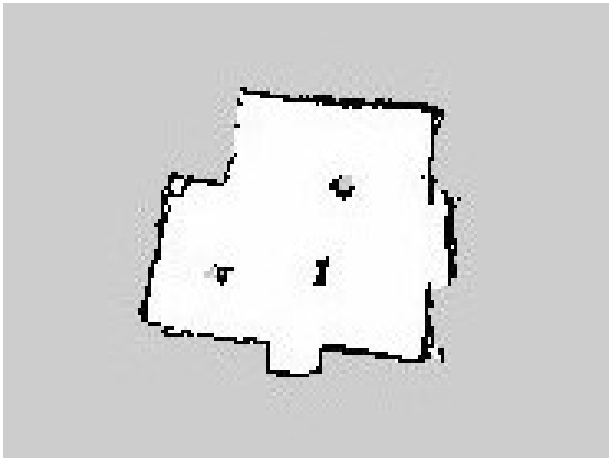

Figura 3.9: Escenario con obstáculos usando el algoritmo propuesto

3.2.5.4. Escenario con pared en el centro

En este escenario el tiempo y la frecuencia de escaneo es considerablemente mejor con repecto a los dos algoritmos, el mapa obtenido en la Figura 3.10 muestra errores, pero se considera que ningún algoritmo obtuvo buenos resultados.

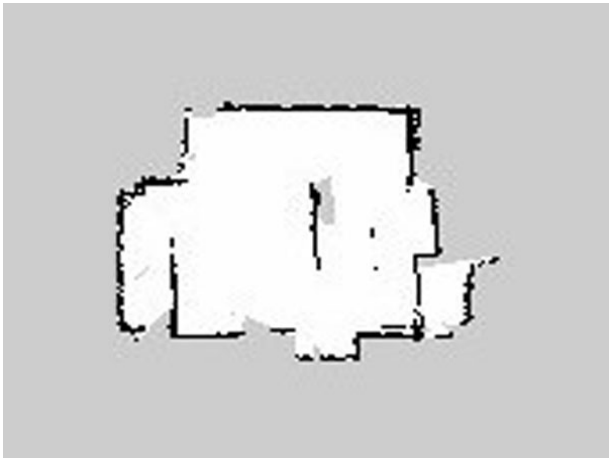

Figura 3.10: Escenario con pared en el centro usando el algoritmo propuesto

3.2.5.5. Escenario pasillo

En el último escenario los valores de los parámetros son menores, debido a la dimensión amplia del pasillo, efectuando que el robot avance solo hacia adelante en esta prueba.

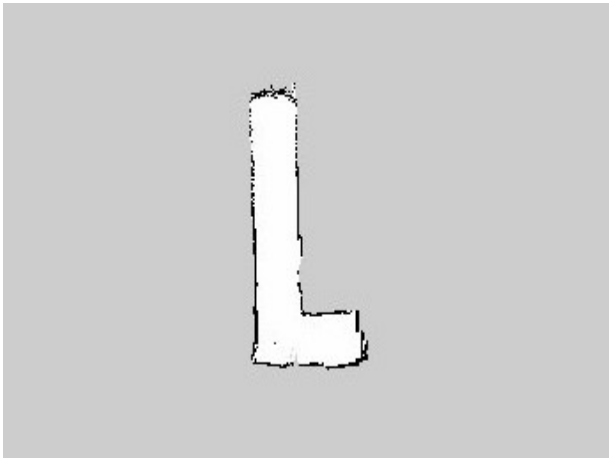

Figura 3.11: Escenario pasillo usando el algoritmo propuesto

3.2.6. Resultados del Algoritmo DFDM

3.2.6.1. Escenario habitación vacía

En esta prueba los parámetros analizados obtienen un valor mayor a los otros dos algoritmos, dando como resultado el mapa de la Figura 3.12.

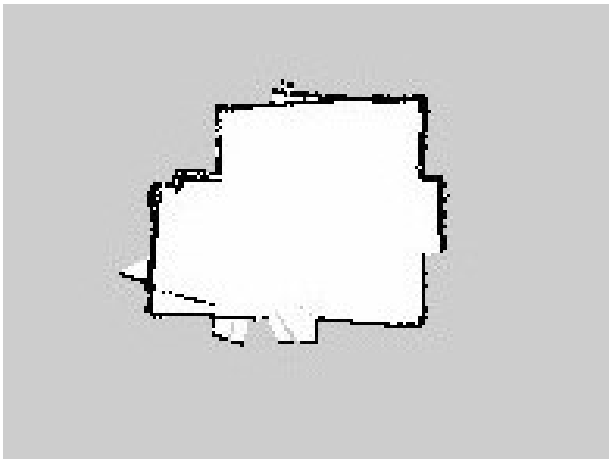

Figura 3.12: Escenario habitación vacía usando el algoritmo DFDM

3.2.6.2. Escenario terraza

En el escenario terraza se obtiene un tiempo y frecuencia de escaneo demasiado altos, una gran diferencia con respecto a los demás algoritmos, siendo esta prueba la de peores resultados en cuanto a valores, véase la Figura 3.13.

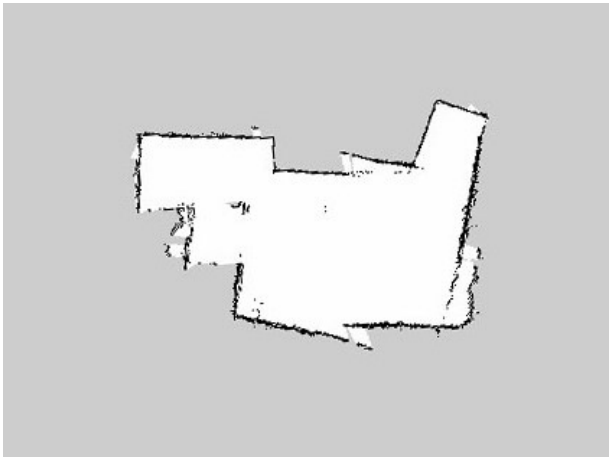

Figura 3.13: Escenario terraza usando el algoritmo DFDM

3.2.6.3. Escenario con obstáculos

Aquí se consigue un tiempo y cantidad de escaneos mayor, siendo aproximadamente el doble, lo rescatable de esta prueba es la imagen creada que es el mejor resultado obtenido, véase la Figura 3.14.

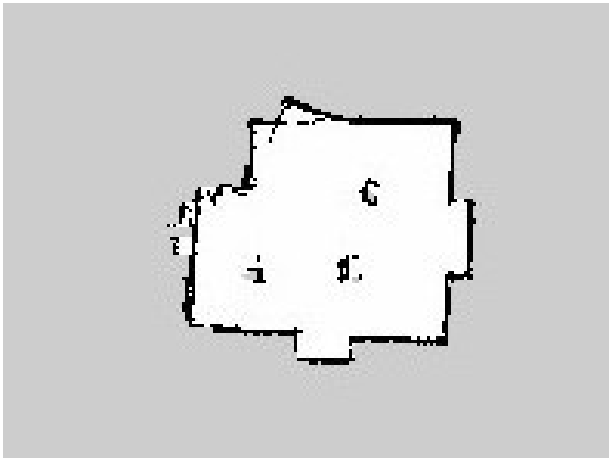

Figura 3.14: Escenario con obstáculos usando el algoritmo DFDM

3.2.6.4. Escenario con pared en el centro

En esta prueba se obtiene un tiempo y frecuencia de escaneo alto y la proyección del mapa no es buena, observar la Figura 3.15.

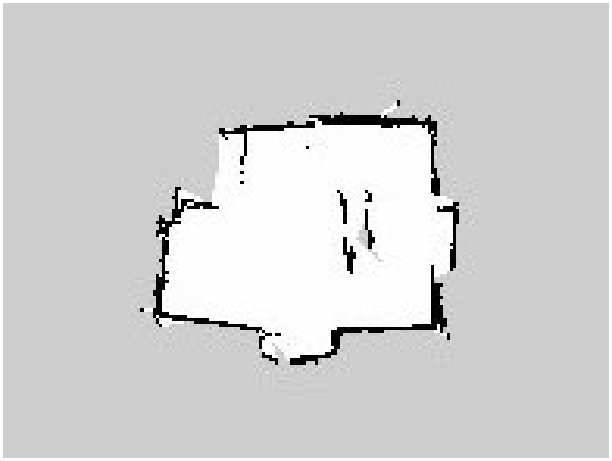

Figura 3.15: Escenario con pared en el centro usando el algoritmo DFDM

3.2.6.5. Escenario pasillo

Aquí el rendimiento fue mejor que el algoritmo DFS en cuanto a los parámetros analizados pero la Figura 3.16 es la peor obtenida, la mayoría de resultados de este algoritmo denota falencias debido a que debe trasladarse cantidades amplias haciendo que su odometría dismiuya.

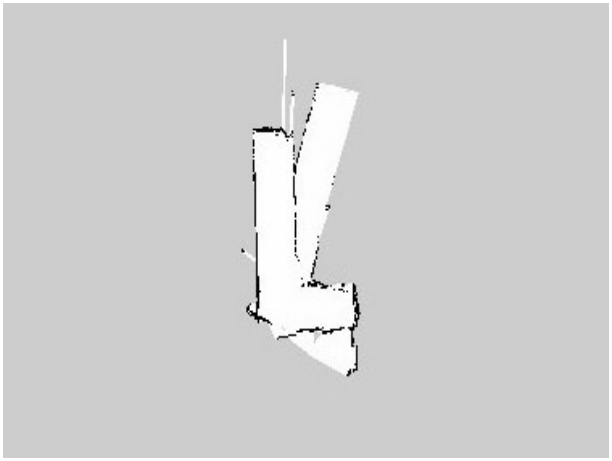

Figura 3.16: Escenario pasillo usando el algoritmo DFDM

3.2.7. Resultados del Algoritmo DFS

3.2.7.1. Escenario habitación vacía

En el escenario habitación vacía se obtiene el mejor tiempo de mapeo, aunque usa una mayor cantidad de escaneos, el mapa obtenido se observa en la Figura 3.17.

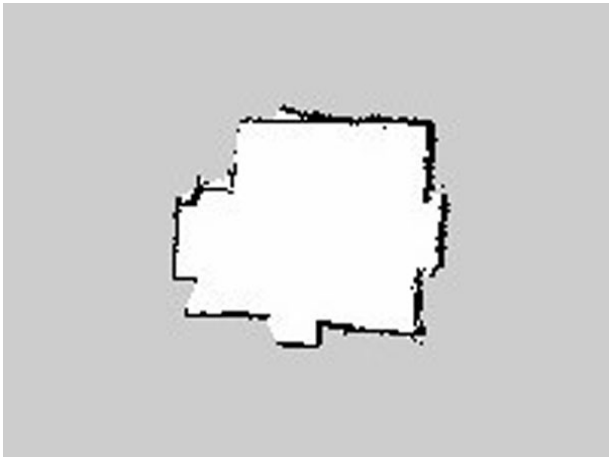

Figura 3.17: Escenario habitación vacía usando el algoritmo DFS

3.2.7.2. Escenario terraza

En este escenario no se alcanza buenos resultados en los parámetros analizados y de esta forma queda la creación del mapa en la Figura 3.18

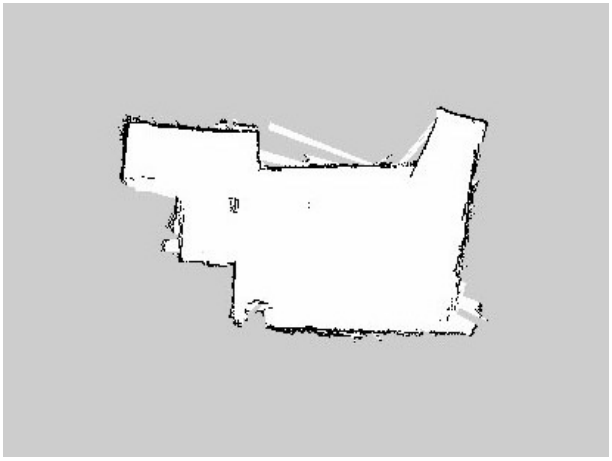

Figura 3.18: Escenario terraza usando el algoritmo DFS

3.2.7.3. Escenario con obstáculos

Aquí se logra obtener resultados casi similares al algoritmo propuesto y el mapa de la Figura 3.19 presenta buenos detalles.

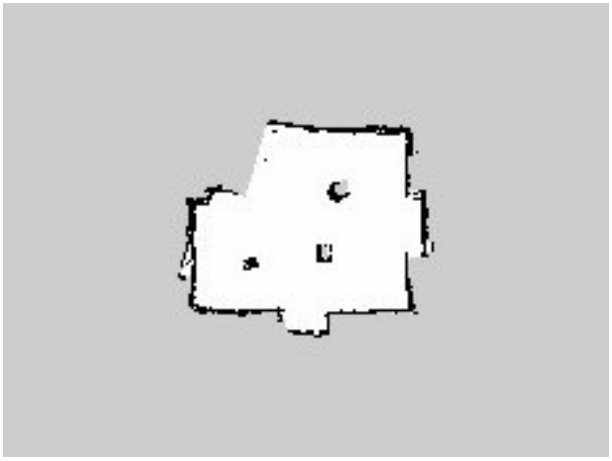

Figura 3.19: Escenario con obstáculos usando el algoritmo DFS

3.2.7.4. Escenario con pared en el centro

En el escenario con pared en el centro no se alcanza los resultados esperados debido a que hizo un tiempo mucho mayor con respecto al algoritmo propuesto y eso conlleva a realizar mas escaneos, además se obtiene el peor mapa entre los demás algoritmos, véase la Figura 3.20.

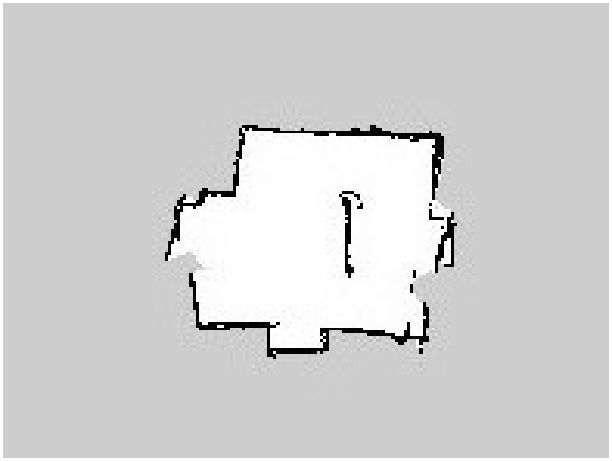

Figura 3.20: Escenario con pared en el centro usando el algoritmo DFS

3.2.7.5. Escenario pasillo

En el último escenario se observa que no obtuvo mucha diferencia con respecto a el algoritmo propuesto y la Figura 3.21 tiene buenos resultados. Pudiendose notar que presenta algunas características similares al algoritmo propuesto, debiendo mejorar la odometría para poder obtener mejores resultados.

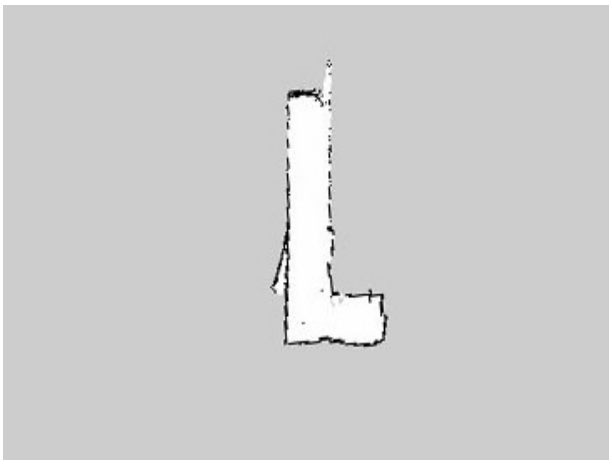

Figura 3.21: Escenario pasillo usando el algoritmo DFS

3.2.8. Análisis de los resultados

El algoritmo propuesto tiene un mejor desenvolvimiento en los escenarios, la distancia media que se desplaza con respecto al punto frontera y la obtención de datos y odometría que logra al girar 360 grados permite que la creación de los mapas sea mejor, dado que al movilizarse la distancia programada hace que el robot conserve de mejor manera su odometría y esto se puede apreciar en las imagenes finales obtenidas, pero los mejores resultados se encuentran en los escenarios con obstáculos o áreas estrechas porque mantiene mejor su posición. El usar los puntos frontera hace que el robot se desplace a lugares donde se puede obtener más información y el moverse de manera prudente logra producir mejores resultados.

4. Conclusiones y Recomendaciones

4.1. Conclusiones

- Se evaluó toda la información necesaria para comprender de manera correcta la metodología de exploración de fronteras y conocer diferentes métodos que sirivieron de guía para proponer un algoritmo que tenga dichas funcionalidades.
- El algoritmo propuesto está desarrollado para que el mapeo de un espacio interior mediante la metodología de exploración de fronteras da como resultado una optimización del tiempo que toma el proceso, al usar la distancia media de la frontera y optimizando su trayectoria mediante un algoritmo de Dijkstra.
- Los resultados de las pruebas de funcionamiento para el algoritmo de exploración de fronteras en interiores determinaron una mayor eficiencia en relación al tiempo, frecuencia de escaneo y mapa obtenido en comparación con los algoritmos de prueba. El algoritmo DFDM solo obtuvo buenos resultados en los mapas obtenidos que fueron mejor en cuanto a legibilidad en comparación al algoritmo DFS, pero esté lo supero en el rendimiento de los parámetros que se puso a prueba.

4.2. Recomendaciones

- Para el desarrollo de futuros temas enfocados en exploración de fronteras es necesario tener conocimientos básicos en el lenguaje de programación python debido a que la mayoría de repositorios digitales con relación a este tema se encuentra de esta manera en GitHub.
- La exploración de fronteras en interiores podría optimizar en una gran gama el proceso de mapeo, este proyecto puede servir como punto partida para futuras investigaciones.
- Debido a la poca información en repositorios digitales con referencia a exploración de fronteras se recomienda ampliar este proceso también hacia el reconocimiento de objetos para usar con mayor eficacia los recursos del robot móvil turtlebot 2.0.

4.3. Trabajos Futuros

La metodología exploración de fronteras se puede complementar con el uso de visión artificial, con la finalidad de aumentar la funcionalidad del turtlebot, haciendo que el robot pueda buscar cualquier objeto en un área desconocida y este pueda indicar su ubicación.

Bibliografía

[1] A. Ollero. *Robótica: Manipuladores y Robots Móviles*. Barcelona: Marcombo, 2001, pp. 8-10.

[2] E. Uslu, F. Cakmak, M. Balcilar, A. Akinci, M. Amasyali and S. Yavuz, "Implementation of frontier-based exploration algorithm for an autonomous robot," in International Symposium on Innovations in Intelligent SysTems and Applications, Madrid, 2015, pp. 2-5.

[3] K. Hidaka and N. Kameyama, "Hybrid Sensor-Based and Frontier-Based Exploration Algorithm for Autonomous Transport Vehicle Map Generation," in IEEE 14th International Conference on Automation Science and Engineering (CASE), Munich, 2018, pp. 1-4

[4] R. Sharman, V. Dayanand and G. Kumar, "Comparative Study of Algorithms for Frontier based," *International Journal of Computer Applications*, vol. 77, no. 8, pp.37-40, Septiembre 2013.

[5] D. Ortega. (2017, Septiembre 21). ¿Qué es ROS?. [Online]. Available: https://openwebinars.net/blog/que-es-ros/

[6] A. AlDahak, L. Seneviratne and J. Dias, "Frontier-based exploration for unknown environments using incremental triangulation," in IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR), Suecia, 2013.

[7] C. Zhu1, R. Ding, M. Lin1 and Y. Wu1, "A 3D frontier-based exploration tool for MAVs," in IEEE 27th International Conference on Tools with Artificial Intelligence, Vietri Sul Mare, 2016, pp 2-4.

[8] A. Hournoung. (2021, Enero 21). Octomap. [Online]. Available: https://github.com/OctoMap/octomap

[9] Z. Fang and L. Zhang, "A Multi-Objective Strategy based on Frontier-based Approach and Fisher Information Matrix for Autonomous Exploration," in IEEE International Conference on Cyber Technology in Automation, Control and Intelligent Systems, Nanjing, 2013, pp. 2-5.

[10] Z. Su, J. Zhou, J. Dai and Y. Zhu, "Optimization Design and Experimental Study of Gmapping Algorithm," in Chinese Control And Decision Conference (CCDC), Hefei, 2020, pp. 1-4.

[11] D. L. Silva Lubanco, M. Pichler-Scheder and T. Schlechter, "A Novel Frontier-Based Exploration Algorithm for Mobile Robots," in 6th International Conference on Mechatronics and Robotics Engineering, Barcelona, 2020, pp. 1-5.

[12] R. Johnsonbaugh, *Matemáticas Discretas*, 6th ed. Chicago: Prentice Hall, 2005.

[13] S. Gao, Y. Ding and B. Chen, "A Frontier-Based Coverage Path Planning Algorithm for Robot Exploration in Unknown Environment," in Proceedings of the 39th Chinese Control Conference, Shenyang, 2020.

[14] M. khawaldah, S. Livatino and D. Lee, "Reduced Overlap Frontier-based Exploration with Two Cooperating Mobile Robots," in IEEE International Symposium on Industrial Electronics, Bari, 2017.

[15] F. Blatt and H. Szczerbicka, "Combining the Multi-Agent Flood Algorithm with Frontier-Based Exploration in Search and Rescue Applications," in International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS), Seattle, 2017.

[16] Y. Wang, A. Liang and H. Huang, "Frontier-based Multi-Robot Map Exploration Using Particle Swarm Optimization," in IEEE Symposium on Swarm Intelligence, Paris, 2011, pp. 3-5.

[17] J. Faigl and M. Kulich, "On Determination of Goal Candidates in Frontier-Based Multi-Robot Exploration," in European Conference on Mobile Robots (ECMR), Barcelona, 2013, pp. 2-3.

[18] Phoemur. (2018, Diciembre 11). Hungarian Algorithm. [Online]. Available: https://github.com/phoemur/hungarian_algorithm

[19] U. Mandavia. (2018, Mayo 28). Turtlebot 2 autonomus navigation. [Online]. Available: https://github.com/ujasmandavia/turtlebot-2-autonomous-navigation

[20] J. Alvarado (2021, Junio 29). Turtlebot2_frontierexploration. [Online]. Available: https://github.com/JoelAlvarado96/Turtlebot2_frontierexploration

[21] S. Bakalarz, A. Mickael and L. Mackeprang. (2019, Junio 4). Robotic Engineering 2 ROS PROJECT. [Online]. Available: https://github.com/koalasession/BSCV5_RobEng2#frontier-exploration

[22] P. López, "Análisis de algoritmos para localización y mapeado simultáneo de objetos," M.S. Thesis, Universidad de Sevilla, Sevilla, 2016.

[23] G. Sánchez and V. M. Lozano, “Algoritmo de Dijkstra. Un tutorial interactivo”presented at VII Jornadas de Enseñanza Universitaria, Madrid, 2016.

[24] D. G. Centelles, “Control de Robots Móviles Autónomos: Planificación de rutas en mapas geométricos”, Universidad Autónoma de Barcelona, Barcelona, 2015.

[25] L.J. Krajewski and L.P. Ritzman. *Administracion de Operaciones*. Mexico: Pearson Education, 2000.

[26] M. M. Lander, “Implementación en una plataforma móvil de la técnica SLAM mediante control inalámbrico”. Thesis, Universidad del País Vasco, Sevilla, 2019.

[27] F. Andrade and M. Llofriu, “SLAM estado del arte”, MINA Network Management Artificial Intelligence, Montevideo, Uruguay, 2019.

[28] D. M. Turnage, “Simulation Results for Localization and Mapping Algorithms,” in Proceedings of the 2016 Winter Simulation Conference, Washington, 2016.

[29] S. Sarkka. *Bayesian Filtering and Smoothing*, New York: Cambridge, 2013.

[30] Turtlebot. (2020). What is Turtlebot?. [Online]. Available: https://www.turtlebot.com

[31] ROS.org. (2017). About ROS. [Online]. Available: https://www.ros.org/about-ros/

[32] ROS.org. (2019). Ubuntu install of ROS Indig. [Online]. Available: http://wiki.ros.org/indigo/Installation/Ubuntu

[33] Turtlebot. (2015). Learn Tutlebot with ROS. [Online]. Available: http://learn.turtlebot.com/2015/02/01/9/

[34] E. Calot, “Reconocimiento de Patrones e Imagenes Médicas basado en Sistemas Inteligentes,” Thesis, Universidad de Buenos Aires, Buenos Aires, 2008.

[35] A. P. Eriksen, “Implementación y Comparativa de Algoritmos de Control y Planificación Local para Robots Móviles utilizando ROS,” Thesis, Universidad Politécnica de Madrid, Madrid, 2017.

[36] F. J. Rodríguez, V. Matellán, J. Balsa and F. Casado, “Cybersecurity in Autonomous Systems: Evaluating the performance of hardening ROS”, Universidad de Salamanca, España, 2016.

[37] J. A. Nieto, M. A. Ibarra and L. A. Ibarra, “CONTRUCCIÓN DE MAPAS DE OCUPACIÓN PARA ROBOTS DE SERVICIO”, *Jóvenes en la Ciencia*, vol. 2, no.1, pp. 1210-1214, Enero, 2017.

[38] K. Cristian, C. Miguel and M. Marcelo, “INTELIGENCIA ARTIFICIAL APLICADA A LA NAVEGACIÓN AUTÓNOMA DE ROBOTS MÓVILES,” in XVII Workshop de Investigadores en Ciencias de la Computación, Salta, 2015.

[39] D. A. Mendoza, “Diseño de la interfaz de control para un cuadricóptero con el dispositivo kinect”, Thesis, Universidad de las Fuerzas Armadas, Quito, 2016.