Localización y mapeo simultáneo por robots móviles con ruedas basados en mapas de ocupación y matrices dispersas

Elizabeth Morales Muñoz

Orientador: Prof. Dr. Dennis Barrios Araníbar

Tesis profesional presentada a la Escuela Profesional de Ciencia de la Computación como parte de los requisitos para obtener el Título Profesional de Ingeniera en Informática.

UCSP- Universidad Católica San Pablo
Abril de 2018

A mi papá, mamá, mis hermanas y el LARVIC

Abreviaturas

GPS *Global Positioning System*

IMU *Inertial Measurement Unit*

IPS *Internet Provider Security*

MPTE-SLAM Mapas Provisionales Topológicos Esparsas

PPR *Pulses Per Revolution*

RFID *Radio-Frequency Identification*

ROS *Robot Operating System*

SLAM *Simultaneous Localization and Mapping*

XML-RPC *Extensible Markup Language Remote Procedure Call*

Agradecimientos

A Dios, a mis hermanas, a mi papá que fue un pilar en mi vida dejando siempre ejemplo en los que lo aman y quedan. A mi mamá porque sin su infinito amor, infinita paciencia y perseverancia, me hubiero rendido hace mucho. A la Universidad Católica San Pablo por los buenos profesores, la formación humana y todas las oportunidades que me han dado. Al LARVIC donde he aprendido que todo se hace con pasión y valores en Cristo. Al profesor Dennis mi mentor al que admiro tanto, tuve la dicha de conocer, que siempre ha compartido su conocimiento y me ha enseñado con su ejemplo el verdadero fin de nuestra profesión. A la profesora Raquel una mujer fuerte, compasiva, muy inteligente, exigente y madre admirable. A Beto, Claudita, Percy, Alan, Vitoco, Yilbert, Yeka, Roger y Edwin. Para mis grandes amigos Mafer, Lichi, Daniel y Kevin, que han marcado mi vida con todas sus virtudes, a todos los antes nombrados, les dedico el trabajo que más me ha costado realizar.
Agradecimiento al Fondo para la Innovación, Ciencia y Tecnología (FINCyT), Perú, bajo contrato 216-FINCyT-IA-2013.

Resumen

El trabajo presentado a continuación se enfoca, en almacenar posiciones previas y posteriores de las lecturas (sensores) optando por *Simultaneous Localization and Mapping* (SLAM), para almacenar todos los estados percibidos del ambiente. Se incluyó sensores de profundidad para percibir el ambiente y en cuanto a la orientación, se trabajó con los *encoders* de las ruedas. Los *encoders* permiten aplicar odometría, una técnica que retorna tres valores importantes. La posición en 2D (x,y) y el ángulo de orientación del robot. A medida que inicia el envío de las lecturas de los sensores, se grafica el resultado, en tiempo real. Pero a medida que la información de los sensores, se incrementa se aplican, diferentes estructuras de almacenamiento, que mejoran el desempeño. Las técnicas más utilizadas en mapeamiento y que se pusieron en contraste en este trabajo, son los mapas topológicos y los mapas de ocupación. En cuanto a los mapas de ocupación, definimos que la ventaja principal, se encuentra en el almacenamiento práctico para la percepción retornada por el robot. Las desventajas principales son, el almacenamiento de datos en matrices de n x n, las cuales desperdician memoria debido a los recorridos del robot, en ambientes complejos. Los mapas topológicos por otro lado, tienen por ventaja principal el manejo eficiente de memoria debido a que trabajan con grafos. Con respecto a las desventajas en el abordaje con grafos, la principal se resume en la función que convierte un grafo en un mapa. Esta complejidad se encuentra en la distribución de ambientes en nodos y el retorno de las características de ese mapeamiento. La estructura propuesta MPTE-SLAM, surge tanto de las ventajas en los mapas topológicos y los de ocupación. MPTE-SLAM, es una matriz dispersa con nodos del tipo matriz, que almacena los valores retornados de la odometría, y se actualiza dando forma a los mapas. Los resultados obtenidos de todos los procesos, retornan un mapa adecuado del ambiente y el recorrido de este, con un mejor manejo de la memoria brindando un pequeño aporte al trabajo de mapeamiento con sensores de profundidad. Los mapas resultantes en la segunda parte de los resultados, poseen mayor precisión y mejor uso de la memoria. Pudiendo concluir, que esta estructura dispersa de nodos matriz, realmente genera una intersección de las ventajas en las técnicas más utilizadas. En cuanto a la trata de las actualizaciones, también mejora el resultado cuando es comparado con una matriz de ocupación.

Abstract

The main focus of this work is to store the lectures of the previous and posterior positions with sensors, Simultaneous Localization and Mapping (SLAM) approach was chosen. In the case of robot orientation, wheels had encoders. This encoders let the odometry technique able to be adapted. Odometry returns three important values, the position in 2D (x,y) and the robot orientation angle. The feedback of the information and the result should be stored, in real time. But as the data increases, more information has to be stored in structures to improve the performance of mapping. The most applied techniques in mapping are topological maps and occupancy maps. Regarding occupancy maps, the main advantage here, is the way information is storaged and returned by the robot. The disadvantage of occupancy maps, relies in memory usage in matrices of size n, where memory is wasted due to the robot path in the environment. On the other hand topological maps, has as their main advantage the efficient management of memory because they work with graphs. With respect to the disadvantages of this kind of structure, the main one is summarized in the function that converts a graph into a map. This complexity is found in the distribution of environments in nodes and then returned as a map.The proposed structure in this work, MPTE-SLAM, arises from both advantages in topological maps and occupancy maps. MPTE-SLAM is a sparse matrix with matrix type nodes, which stores the data returned from the odometry and updates it, while shaping the map. From all this processes, the return is an adequate map of the environment and the route, with better management of memory providing a small contribution to the mapping work with depth sensors. The resulting maps in the second part of the results, have better precision and better use of memory. Concluding that this spatial structure of matrix nodes really generates an intersection on the advantages from topological mapping and occupation maps. Regarding in updates, MPTE-SLAM also improves the result with a occupation matrix.

Índice general

Índice de figuras

Capítulo 1

Introducción

Hoy en día simulamos inteligencia en las computadoras con el objetivo de desarrollar tareas específicas; incluyendo la interacción de robots en el mundo real. Así, es importante destacar que un robot es un dispositivo mecánico versátil equipado con sensores y actuadores, controlado por un sistema computacional, capaz de extraer información del ambiente y usar ese conocimiento al respecto del mundo para actuar sobre el mismo a través de movimientos [1].

El término robot, surge en *Rossum Universal Robots* y fue definido como robota o labor forzada. El término se tornó real durante el año 1928, cuando crean el primer humanoide mecánico. Este prototipo fue presentado en la *Model Engineers Society*, el cual, además de poder mover sus manos de aluminio, tenía una cabeza que era manipulada por un control remoto [7].

Es importante destacar, que a pesar que el término robot naciera en 1928, en los años 1737, ya existía tecnología relacionada en mecánica como los autómatas; con creaciones como el tamborilero y el pato que digiere, los cuales se consolidaron como los primeros mecanismos previos a los robots. El pato, que aún puede verse en un museo de Europa, mueve sus alas, come grano, lo digiere y retorna unas bolitas en un compartimiento escondido [7].

En cuanto al tema ético de los robots, surgen muchos debates sobre la aplicación de los mismos en la sociedad. En 1941 se formulan las tres leyes de la robótica [7], las cuales son de suma importancia porque se acuña el verdadero fin de la aplicación tecnológica para la sociedad y estas reglas son:

1. Un robot no hará daño a un ser humano o, por inacción, permitirá que un ser humano sufra daño.
2. Un robot debe obedecer las órdenes dadas por los seres humanos, excepto si estas órdenes entran en conflicto con la 1era Ley.
3. Un robot debe proteger su propia existencia en la medida en que esta protección no entre en conflicto con la 1era o la 2da Ley.

Años más tarde el instituto Neurológico Burden en Bristol, Inglaterra entre 1948-1949 crea el primer robot autónomo. Quería probar que las conexiones entre un número de células cerebrales pueden generar comportamientos muy complejos en los robots. Este uso de inteligencia artificial, imita el comportamiento de una parte del cerebro humano. Los nombres de sus robots eran Elmer y Elsie los cuales eran descritos como tortugas por sus movimientos lentos, lo más interesante era que podían encontrar la dirección para recargar sus baterías en una estación, configurando así, la primera generación de robots autónomos [20].

La segunda generación de robots fué desarrollada entre 1990, hasta nuestros días. Estas máquinas pueden ser estacionarias o móviles, autónomas o controladas con una programación sofisticada. Se caracterizan por un hecho tecnológico histórico, el surgimiento de la electrónica análoga para emular los procesos cerebrales. Se introdujo el análisis de los procesos mentales en términos de computación digital [25]. Lo anterior permitió definir al robot de la segunda generación como un dispositivo con un mínimo de autonomía, capaz de actuar y sentir el mundo tomando decisiones para actuar de forma inteligente en éste, enlazando el sentir y el actuar del robot [25].

Actualmente la robótica tiene muchos campos de investigación; sin embargo, al referirse a una acción inteligente nos referimos a lograr autonomía [5]. El proceso de explorar nuevos ambientes sin tener previo conocimiento del lugar y retornando su camino, brinda al robot un mínimo grado de la misma. Retornar un mapa, por medio de sus sensores y un tramo del recorrido es el reto que se propone en este trabajo de investigación.

1.1. Motivación y Contexto

En mapeo para navegación de robots, se busca construir y actualizar el mapa, de un ambiente desconocido, mientras que simultáneamente se mantiene el tramo de todo el recorrido del robot. Conocido como *Simultaneous Localization and Mapping* (SLAM), busca resolver la autolocalización del robot para mejorar el sistema de navegación autónomo. Lograr la autonomía en un robot que mapea ambientes, permite brindar un mapa del espacio recorrido en el que se encuentra (suelos complejos) y la detección de obstáculos y posición.

Toda esta información es aprovechada para que el robot pueda ejecutar tareas en ambientes reales; siendo la búsqueda y rescate de personas una tarea de extrema importancia en nuestro continente, debido a los constantes cambios climáticos y desastres naturales. es así que este trabajo se enmarca dentro de la necesidad de dotar a robots de rescate de la capacidad de auto localizarse y mapear un ambiente desconocido.

Existen múltiples trabajos de investigación que plasman soluciones para SLAM, pero siempre existen posibilidades de mejora en diferentes ámbitos [40]. Uno de ellos es el costo computacional a nivel de procesamiento, también a nivel de uso de memoria y a nivel de comunicación.

1.2. Planteamiento del Problema

Cuando se desea dar una solución a SLAM, el robot debe ser capaz de recorrer ambientes, mientras que al mismo tiempo se incluyen las lecturas de objetos nuevos en espacios continuos, aumentando el costo de procesamiento constantemente, por los valores previos, actuales y posteriores de los resultados.

Una forma de hacer SLAM es conocida como mapas de ocupación, los cuales almacenan toda la información en una matriz estática, lo que implica un desperdicio y alto costo de memoria. Lo anterior se vuelve evidente si recorremos estructuras no cuadradas grandes como edificios, o si es que la división del mundo es muy pequeña (al buscar que genere un mapa, más grande y preciso). Aunque existen muchos enfoques para resolver SLAM, los mapas de ocupación son bastante acertados dentro de espacios continuos para obtener obstáculos, pero generan estructuras muy pesadas y no son óptimos en espacios grandes [32].

Existe también un tipo de solución basado en mapas topológicos, los cuales son capaces de almacenar muchos nodos con representaciones de ambientes. Cada nodo representa una matriz, solucionando el problema de la pérdida de espacio que ocurre en los mapas de ocupación, pero pueden introducir errores en la creación de nuevos ambientes para los nodos y son poco predecibles geométricamente en cuanto a su estructura. Reconocen nuevas habitaciones por ingresos a puertas o ventanas. Por ejemplo, si la habitación cuenta con un almacén, contaría el almacén como un nuevo nodo. En este tipo de solución se hace necesario el uso de una función compleja que ordena las lecturas, uniéndolas a cada nodo; pues si vemos un grafo sin este procesamiento, no podríamos reconocer un mapa [27].

La mayoría de trabajos buscan almacenar grandes cantidades de información para resolver SLAM, por lo que a medida que han ido avanzando en búsquedas de soluciones, se tomaron los mapas topológicos como referencia, trabajando únicamente con grafos. Los grafos en generaciones de mapas como ya se mencionó, hacen que la tarea de poder trabajar geométricamente no sea natural para las personas. En este sentido, es importante desarrollar algoritmos y soluciones que exploten las ventajas de las dos formas actuales de resolver el problema de SLAM.

1.3. Objetivos

1.3.1. Objetivo General

Desarrollar una estructura de almacenamiento para mapas de ocupación, que brinde una solución al problema de localización y mapeo simultáneo permitiendo el crecimiento dinámico de los mapas y la optimización del uso de memoria.

1.3.2. Objetivos Específicos

- Desarrollar una estructura de almacenamiento dinámica de mapas de ocupación que permita resolver el problema del uso de estructuras de almacenamiento y que permita el crecimiento del mapa en todas las direcciones.
- Optimizar el uso de memoria para el almacenamiento de mapas de ocupación en la solución del problema de localización y mapeo simultáneo.

1.4. Organización de la tesis

Estructuralmente este trabajo se divide de la siguiente manera:

- En el capítulo 2, el marco teórico, se define como los sensores obtienen el mundo para el robot, por medio de diferentes abordajes. Luego, se analiza cómo es la representación del mundo para el robot, explicando cómo trabajan los mapas topológicos y los mapas de ocupación. Incluyendo las consideraciones finales, que abordan el comportamiento de una matriz dispersa versus grafos
- En el capítulo 3 se presenta el estado del arte de las técnicas que aplican mapas de ocupación y SLAM.
- La técnica propuesta es descrita en el capítulo 4, con la clasificación de la misma. Dando una descripción de las estructuras y métodos aplicados.
- En el capítulo 5 de implementación, se muestran las características del robot y sus sensores. Técnicas aplicadas, la obtención de objetos y obstáculos en el ambiente. Como también, la inclusión de los algoritmos implementados en MPTE-SLAM.
- En el capítulo 6, se analizan los resultados para las pruebas realizadas sobre la propuesta.
- El capítulo 7 muestra las conclusiones y trabajos futuros, finalizando con la bibliografía.

Capítulo 2

Marco Teórico

Las técnicas y estructuras explicadas a continuación son aplicadas por diferentes autores y en este trabajo para dar soluciones a SLAM.

2.1. El mundo para el robot

2.1.1. SLAM

Definimos que SLAM, busca en lo calculado de la data sensorial, una posición $x1$ en un tiempo t ($x1_t$) a través del mapa en vez de solamente una posición. La Figura 2.1 explica cómo se plantea el problema de SLAM, basándose en estados almacenados. Las mediciones obtenidas son las variables z_{k-1}, z_k, z_{k+1} y las variables de control como, u_k, u_{k+1}, u_{k+2}.Para representar los valores de la Figura 2.1 establecemos:

- El robot es controlado por lo tanto se define el conjunto de variables de control como,
  $u_k{:}_T$={ u_k, u_{k+1}, u_{k+2},..., u_n }.
- Un sensor de profundidad tiene observaciones del mundo (como luce el ambiente), con los sensores de profundidad. En el caso de la Figura 2.1, se muestra la representación para cámaras.
  $z_k{:}_T$={ z_{k-1}, z_k, z_{k+1},..., z_n }.
- La representación del ambiente para el robot:
  $x_0{:}_T$={ x_{k-1}, x_k, x_{k+1},..., x_n }.

La estimación posterior sobre la posición momentánea del robot en el mapa es la parte probabilística de la perspectiva de SLAM, con $p(x_k :_T, m|z_k :_T, u_k :_T)$ [19]. El valor de p representa la distribución de probabilidad, donde se debe crear el camino del robot como una posición x_k en un tiempo dado T representado en $x_k :_T$ y un mapa m en un tiempo T, el control del robot es $u_k :_T$.

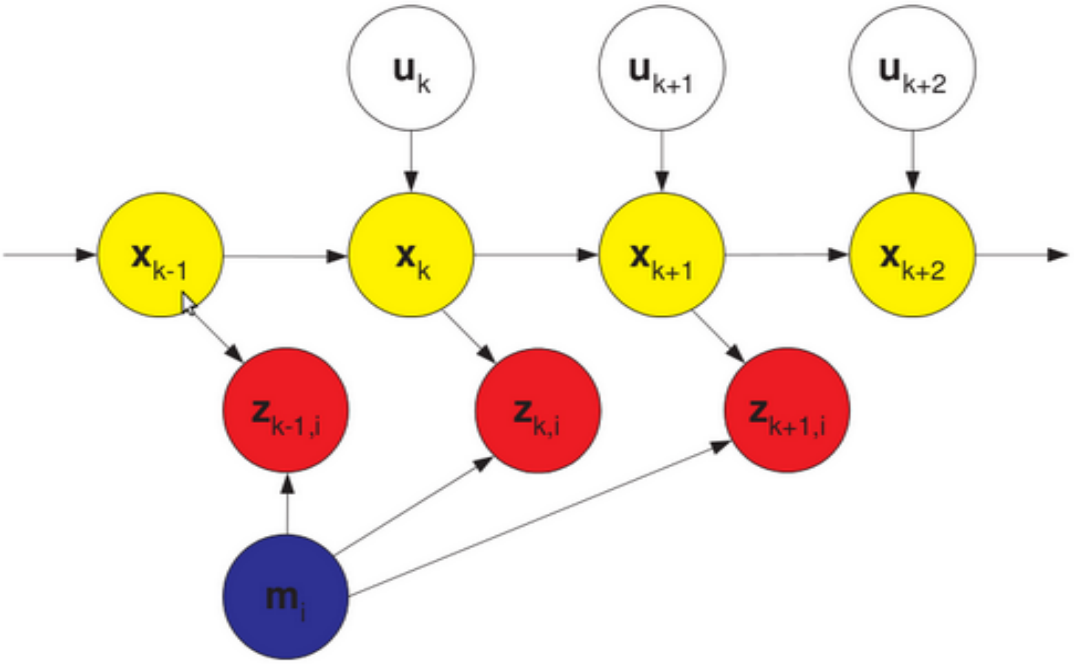


Figura 2.1: Forma gráfica de SLAM [15]

SLAM, es la solución en tiempo real que busca la construcción de un mapa en un ambiente desconocido por un robot móvil mientras que, a la vez navega ese ambiente usando el mapa que ha creado [13]. Los pasos aplicados en SLAM son los siguientes:

1. Cálculo de las coordenadas del robot en el mundo, a partir de los sensores.
2. Pre-computación de los valores obtenidos para referenciar los datos obtenidos.
3. Actualización de la estimación de la existencia o no de obstáculos.
4. Actualización del estado del robot y de las coordenadas de los objetos que existen en el mundo.

2.1.2. Odometría

La odometría es el estudio para la estimación de la posición de vehículos con ruedas durante la navegación. Para realizar esta estimación se aplica información sobre la rotación de las ruedas [22]. Es importante notar que al recorrer un camino, el error de estimación del recorrido va incrementándose. Este incremento se modela en base a unas elipses de error, las cuales son mayores a medida que el tiempo de recorrido se incrementa; como se muestra, en la Figura 2.2 [27].

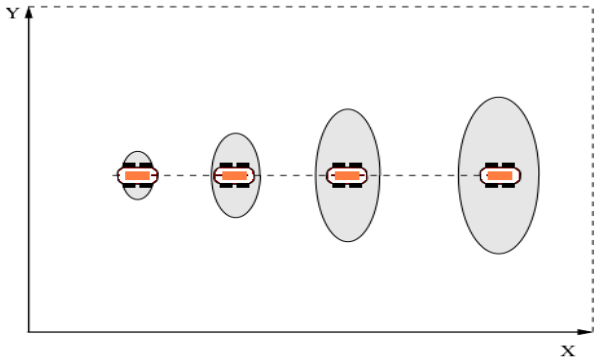


Figura 2.2: Elipses de error en una posición estimada [39]

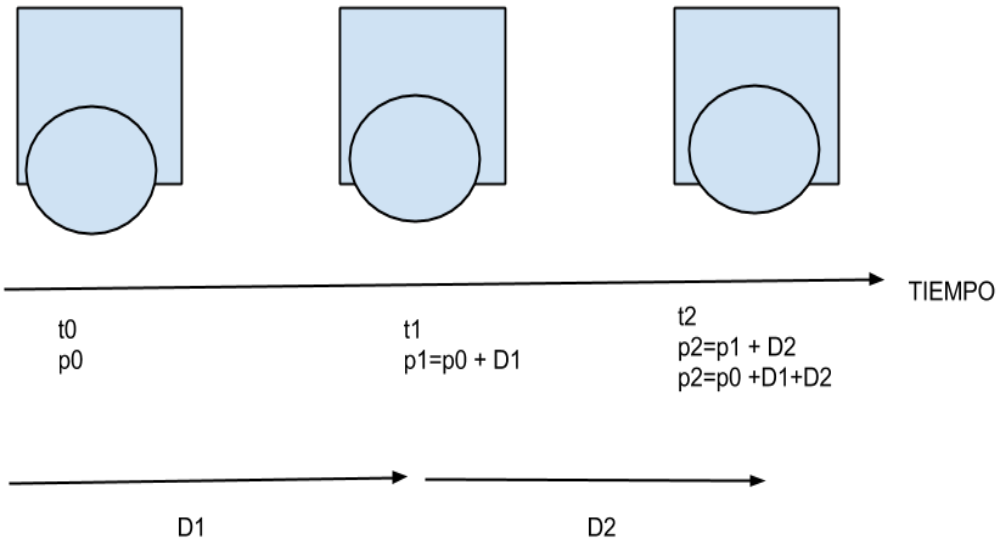


Figura 2.3: Cambios en las ruedas con odometría (Fuente: Creación propia)

Para definir el valor de la elipse de error generada, se calcula la posición y movimiento. Se definen las fórmulas, como en la Figura 2.3, el punto de inicio parte desde 0 con t0 como tiempo y p0 como posición inicial. A medida que el robot, avanza, la distancia y la posición anterior aumentan, por este incremento, sabemos que el robot está en movimiento. A partir de las mediciones de los *encoders*, los cuales son sensores que miden las revoluciones de los ejes, conociendo algunos parámetros cinemáticos del robot, incrementan este valor, por un arco de circunferencia en el sensor. Las siguientes fórmulas definen la velocidad lineal y angular obtenida por el robot para conocer los desplazamientos del robot. Aproximando el movimiento ejecutado por el robot por un arco de circunferencia, mostrado en 2.3, donde r es el radio del giro del robot, donde tenemos que la velocidad lineal será:

$$V = wr \tag{2.1}$$

De la misma forma las velocidades lineales de cada rueda será dado por:

$$Vd = w(r + \frac{b}{2}) \tag{2.2}$$

$$Vd = w(r - \frac{b}{2}) \tag{2.3}$$

Sumando ambas ecuaciones 2.1 y 2.2 tenemos de resultado la ecuación sustituyendo el resultado en la ecuación 2.1 obtenemos::

$$V = \frac{Vd + Ve}{2} \tag{2.4}$$

La velocidad angular del robot, puede ser obtenida restando la ecuación 2.3 y de ecuación 2.4. V es la velocidad tangencial, w es la velocidad rotacional y r, el radio vector.

2.1.2.1. Los Encoders

Para poder obtener los ángulos para la odometría, se utilizan los *encoders* que son, sensores que miden las revoluciones de un eje. Los *encoders*, son los que brindan información relacionada a cuántas vueltas ha dado un eje en un intervalo de tiempo dado. En un robot móvil, los *encoders* se suelen situar en los ejes de las ruedas, en cada uno de los grados de libertad de movimiento que pueda tener cada una de esas ruedas, para evitar elipses de error con valores elevados, que midan las revoluciones de los motores servos previamente, los cuales trabajan con grados para tener mayor precisión en el movimiento.

[39].

Los trabajos de investigación refieren que el uso de *encoders* ópticos permiten un mejor control del grado de desplazamiento en las ruedas del robot. Esto se obtiene generalmente por codificadores ópticos cuyo principio de funcionamiento se basa en la transmisión y recepción de la luz a través del disco perforado que se acopla al eje de cada rueda. Se hace mucho hincapié en los *encoders* debido a que se encuentran en los motores del robot y los *encoders* manejan sus coordenadas por grados [6].

Ventajas del *encoder*:

- Un *encoder* lleva registro del movimiento dentro de su propio marco de referencia absoluto.

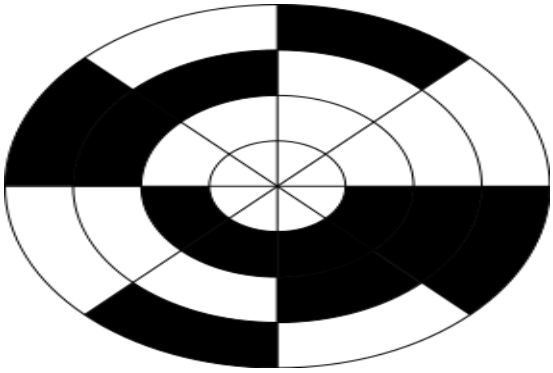


Figura 2.4: Disco *Encoder* (Fuente: Creación propia)

- Siempre, el *encoder* conoce su posición; y su punto de inicio puede ser definido electrónicamente a gusto del programador.
- Los contactos del *encoder* producen una cuenta binaria estándar cuando rota el disco [39].

Desventajas del *encoder*

- El *encoder* típico específica un solo código de ángulo, como se muestra en la Figura 2.4.

2.2. Representación del mundo en el robot

2.2.1. Mapas topológicos

El uso de un grafo para describir un ambiente parece ser, intuitivamente, una buena opción debido a que la estructura compacta para el almacenamiento de la memoria del sistema computacional del robot puede ser utilizada para la resolución de problemas de alto nivel como la planificación de tareas.

El problema con la representación consiste en la ausencia de un patrón de definiciones de las estructuras asociadas a los vértices. La localización de un robot utilizando mapas topológicos es abstracta, porque no existe cómo definir ícitamente la posición y la orientación del robot. Sin embargo, se puede afirmar en que nodo del grafo se encuentra.

La Figura 2.5 ilustra el proceso de construcción de un mapa topológico en un ambiente determinado representando las 3 habitaciones con nodos [5]. La construcción de un mapa topológico se torna compleja en función de la dimensión del ambiente. Los ambientes grandes y complejos pueden presentar información sensorial ambigua, generalmente, son dispersas o podemos cargar fallas al identificar compartimentos que ya han sido mapeados. Este aspecto hace que se dificulte significativamente la construcción y manutención de este tipo de datos. Otro problema encontrado en este tipo de representaciones se encuentra en la capacidad de definir qué elementos deben ser considerados nodos o aristas

de los grafos. Kuipers y Byun utilizan los nodos de un mapa para representar los lugares, caracterizados por estrategias de control [6]. Thrun aplica mapas topológicos obtenidos a partir de un grado de ocupación probabilística particionada en regiones separadas por aristas, por medio de un método de extracción topológica basada en un diagrama generalizado de Voronoi. Formando un esqueleto de las imágenes que retorna un láser, para producir información topológica adecuada [22].

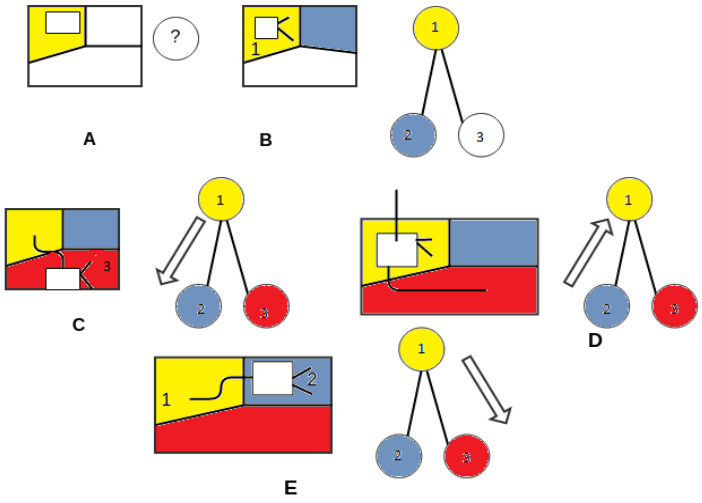


Figura 2.5: Funcionamiento de los mapas topológicos [4]

Los algoritmos de mapeamiento topológico definen cuales son los lugares sensorialmente distintos en un ambiente, los cuales, el robot tiene plena capacidad de reconocer. Independientemente de que elementos son representados, tales estructuras son normalmente enriquecidas con información sobre una posición de referencia para que el robot pueda localizarse y definir su estado en este tipo de mapa. Cada nodo representa una habitación separada por el ingreso del robot a través de una puerta.

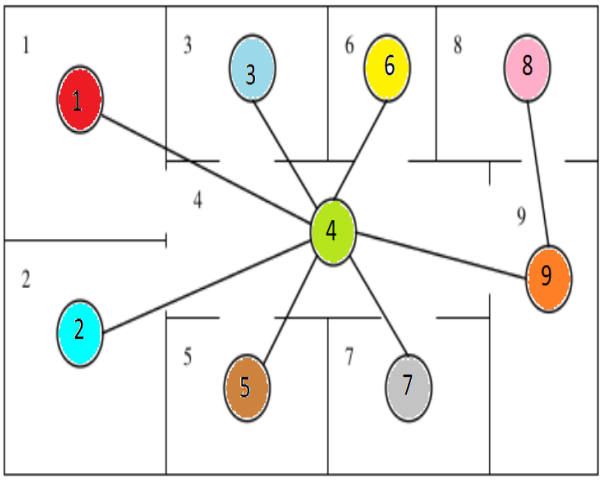


Figura 2.6: Mapa topológico [11]

Un gran problema en la determinación de un mapa a través de los datos sensoriales consiste en la información recolectada que siempre contiene errores en la medición. Estos errores se originan en la incerteza de la localización del robot y las incertezas relativas dependen del tipo de sensor utilizado. En la mayoría de abordajes sobre creación de mapas, cada nodo interno tiene una matriz que guarda la información del nodo, estos nodos son pequeños mapas de ocupación conectados de forma estructural por medio de un grafo. El problema radica en que las puertas o ingresos amplios no necesariamente resultan ser la parte divisoria de una habitación.

Ventajas:

- Permite una planificación, poco espacio de complejidad (la resolución depende de la complejidad del entorno).
- No requiere determinación exacta de la posición del robot

Desventajas:

- La estructura es muy compleja debido al manejo de nodos como habitaciones (*landmarks*).
- El reconocimiento de los lugares (basados en puntos de referencia) a menudo es ambigua y sensible al punto de vista del observador.
- Puede producir trayectorias erróneas.

2.2.2. Mapas de ocupación

Los mapas de ocupación, generan representaciones con un determinado grado de exactitud, en cuanto a geometría del ambiente en el cual el robot se encuentra inmerso. Las paredes, obstáculos y pasajes son fáciles de identificar en este tipo de abordajes. Generando un mapa que mantiene una buena relación topográfica con el mundo real que está recorriendo.

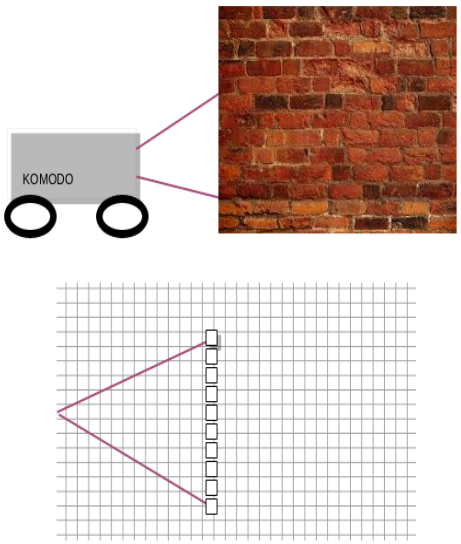


Figura 2.7: Mapas de ocupación (Fuente: Creación propia)

El enfoque estándar en cuanto a los mapas de ocupación, se encuentra en romper el problema global en problemas más pequeños con la estimación de $p(m_{x,y}|Z_{0:t})$ para todas las cuadrículas $m_{x,y}$, donde la distribución de la probabilidad nos pide el retorno de un mapa en 2D, por medio de la variable $Z_{0:t}$ en un tiempo t. Este desglose es conveniente para no perder parte de la estructura del problema, debido a que no permite dependencias de modelado entre las celdas vecinas [9].

Ventajas:

- Son fáciles de implementar, y mantener (en cuanto a las actualizaciones).
- El reconocimiento de lugares (se basa en geometría) , no es ambigua y lo que retorna es independiente.
- Hay facilidad para retornar los caminos.

Desventajas:

- Utiliza demasiado espacio (la resolución no depende de la complejidad del ambiente).
- Requieren información óptima de la posición del robot.

2.3. Estructuras para nuevas soluciones

2.3.1. Matriz Dispersa

Las matrices dispersas son estructuras computacionales, que almacenan información en posiciones específicas. Existen dos tipos de matrices dispersas; las estructuradas y las no estructuras. Una matriz estructurada es aquella cuyas entradas diferentes a cero forman un patrón regular, usualmente cuentan con muchas diagonales. Las matrices no estructuradas, no representan un patrón en sus diagonales. Alternativamente, los elementos diferentes de cero pueden encontrarse en bloques (las submatrices densas, cuentan con muchos elementos diferentes de cero) con el mismo tamaño forman un patrón regular, típicamente por un número pequeño de (bloques) diagonales.

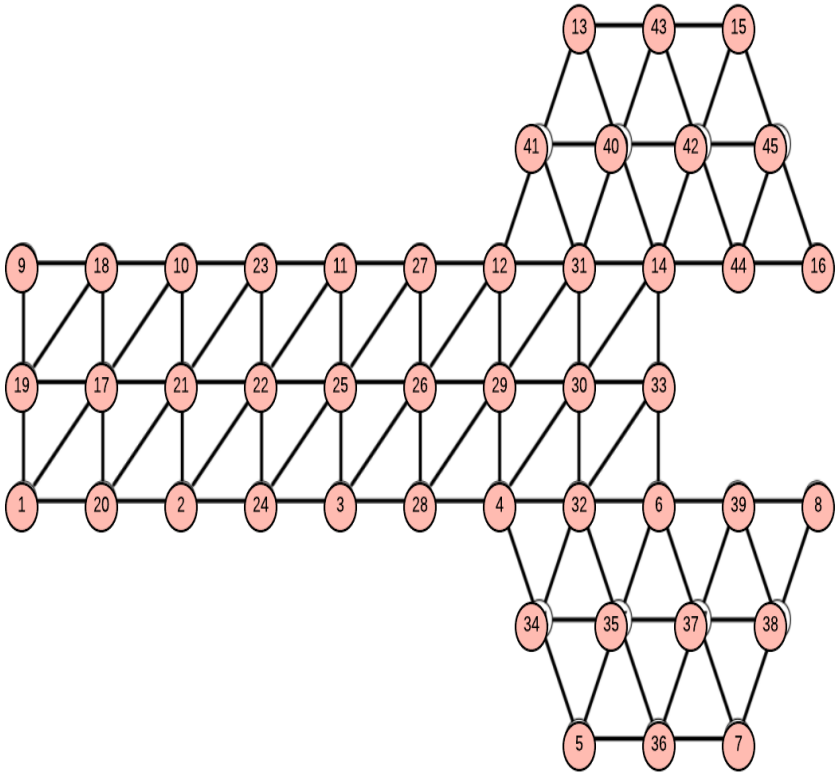


Figura 2.8: Problema de grilla de elementos finitos [30]

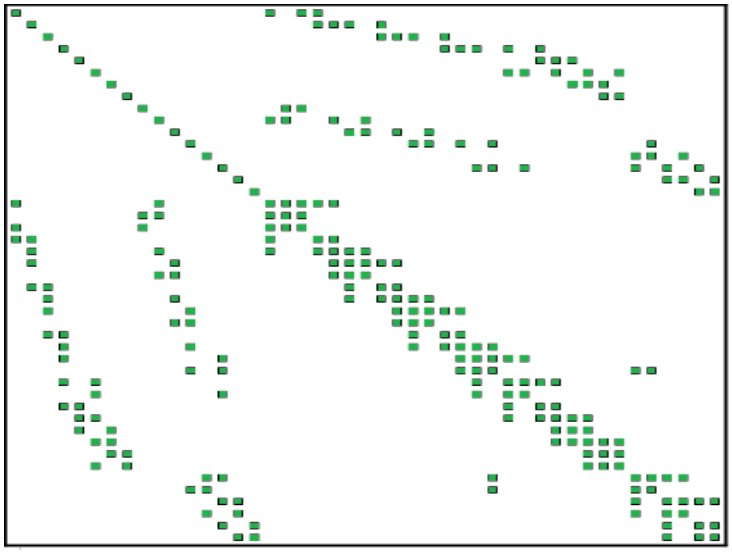


Figura 2.9: Matriz dispersa asociada a un conjunto de elementos [30]

Una matriz que tenga un porcentaje pequeño de elementos que no sean ceros es llamada dispersa. En un sentido práctico una matriz n x n es clasificada como dispersa si tiene un orden n de elementos de valor diferente a cero [29]. Este tipo de estructura está compuesta por muchos elementos con ese valor que se encuentran muy dispersos en la matriz y sin relación entre sí. Para mantener un crecimiento ordenado entre cada dato dentro de la matriz dispersa se recurre a una conexión estructural basada en listas simplemente enlazadas o doblemente enlazadas en caso de recorridos para hacer cálculos dentro de la estructura [26].

Existen también las matrices de estructura irregular en las entradas. El mejor ejemplo de una matriz de estructura irregular es una matriz que consiste solamente de pequeñas diagonales. Matrices de diferencias finitas en mallas rectangulares, como se muestra en la Figura 2.8, son ejemplos típicos de matrices irregulares [30]. La figura 2.9 muestra la estructura de la matriz dispersa, la cual es asociada con el conjunto finito de elementos mostrados en la Figura 2.8.

Este tipo de matrices dispersas fueron investigadas por Gustavson en el centro de investigación de IBM siendo que, muchos reportes de investigación sobre este tema se desarrollaron entre 1972 y 1976. Este tipo de matrices aparecen en la programación lineal, análisis estructurales, teoría de redes con sistemas de distribución de energía, soluciones de ecuaciones diferenciales y teoría de grafos [14].
Ventajas:

- Las matrices dispersas reducen los costos de memoria, son muy eficientes en el uso de memoria para almacenar datos y son prácticas para los accesos de información almacenada [35].
- Otra ventaja está dada por tamaños y la velocidad en que se pueden retornar los resultados, porque solo ocupan una parte de la matriz.
- Si la matriz se llena de elementos completándose, no existiría una ventaja, pero en la aplicación de generación en mapas es poco probable debido a la orientación de

datos retornados.

Desventajas:

- Si se desconoce cuán dispersos están los valores retornados, es complejo definir operaciones y desempeño de la misma, a su vez si se desconoce la memoria utilizada no resulta adecuado utilizar matrices dispersas.
- La definición de un comportamiento o estructura para las conexiones dentro de una matriz dispersa, debe ser definida previamente para ver el tipo de comportamiento y estructura interna que se aplique [10].

2.3.2. Representaciones con grafos

La teoría de grafos es una herramienta ideal para representar las estructuras de matrices dispersas y por esta razón juega un rol muy importante. Por ejemplo, la teoría de grafos es clave para aplicarse en eliminaciones gaussianas dispersa o en técnicas de pre condicionamiento. Un grafo se caracteriza por dos conjuntos de vértices $V = v1, v2, .., vn$, y un conjunto de aristas E que consisten de pares (vi, vj), donde vi, vj son elementos de V. Este grafo $G = (V, E)$ es usualmente representado por un conjunto de puntos en el plano unido por una línea entre los puntos que están conectados por una arista. Un grafo es una forma de representar una relación binaria entre objetos del conjunto V. Por ejemplo, V puede representar las ciudades del mundo. Una línea dibujada entre cualquiera de las dos ciudades que están conectadas por una aerolínea como conexión. En la Figura 2.10 se hace la comparación, entre la matriz dispersa y el grafo, en las mismas posiciones para mostrar el comportamiento.

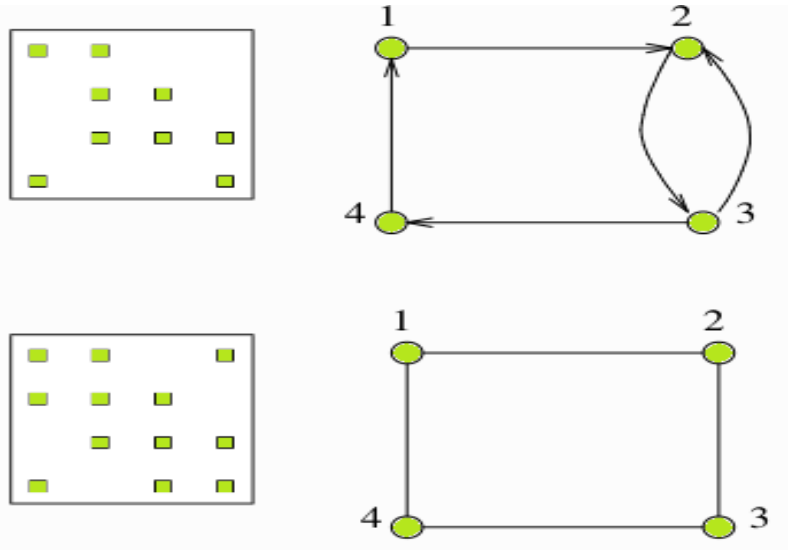


Figura 2.10: Grafos de 4x4 y matriz dispersa [30]

Los mapas topológicos, son representados computacionalmente a través de un grafo. Los grafos son importantes debido a su capacidad de modelar estructuras en cuanto a ciencias de la computación como problemas *NP*- completos, la generación de los caminos más cortos entre ciudades y estructuras para preservar datos y lograr búsquedas más rápidas en menos espacio y conexiones de circuitos de placa impresos. Por este tipo de características los trabajos de robots autónomos también pueden ser fácilmente representados por grafos, reduciendo costo computacional para poder generar mapas de grandes distancias recorridas.

2.4. Consideraciones finales

Luego de la investigación de enfoques que aplican tanto, mapas topológicos y matrices de ocupación para generar mapas del ambiente, surge una nueva propuesta que se basa en la idea de recorrer un camino guardando lo reconocido por el robot en un mapa que no desperdicie espacio en la memoria, y con las actualizaciones constantes que debe aplicarse. Como lo plantea Yousef Saad profesor distinguido del departamento de ciencia de la computación de la Universidad de Minnesota. La diferencia, entre grafos y matrices dispersas no es muy notoria y no ha recibido mucha atención en los últimos años. Sin embargo, la distinción puede ser importante para métodos de soluciones iterativas. El desempeño de este tipo de operaciones puede diferir significativamente en computadoras con múltiples núcleos, dependiendo de si están estructuradas de forma adecuada o no. Por ejemplo las computadoras de vectores que guardan las diagonales de matrices son ideales, pero el esquema general puede sufrir porque requiere direccionamiento indirecto. Se puede definir también que el almacenamiento de un matriz dispersa reduce la creación de mapas basados en edificaciones muy grandes o grandes distancias que recorre un robot, como otro tipo de abordaje para solucionar la localización y mapeo simultáneo trabajando con odometría y sensores de profundidad. Se puede concluir, que el mapa propuesto en este trabajo es una matriz dispersa con nodos del tipo matriz, capaz de unir lo mejor de ambos abordajes con mayor incidencia de aplicación [30].

Con lo expuesto anteriormente, se busca aprovechar las ventajas que poseen ambas técnicas de mapeamiento, tanto para el consumo de espacio en grandes distancias y que el resultado sea geográficamente natural, que refiere a que las personas noten que lo impreso no tenga internamente una función compleja para que se logre un mapa. La mayor parte de las contribuciones de Saad están incluidas en cálculos matriciales, incluyendo métodos iterativos para solucionar grandes sistemas algebraicos del tipo dispersa y computación paralela. El refiere que en la matemática un grafo y una matriz son lo mismo pero como se mostrará más adelante el grafo no nos da noción de ambiente, y una matriz dispersa no solo muestra la discretización del ambiente sino también el recorrido del robot.

Capítulo 3

Estado del Arte

3.1. Estructuras para almacenamiento de mapas

En este capítulo se hace el análisis de los trabajos que proponen el uso de mapas de ocupación porque son las estructuras con las que se contrasta en este trabajo. Así mismo es importante destacar que las propuestas que proponen el uso de mapas topológicos no han sido analizadas porque tienen un comportamiento similar en complejidad computacional.

3.1.1. Mapas de ocupación

En el cuadro 3.1 se presenta la evolución de los mapas de ocupación, con un aporte para estructuras modernas que aplican matrices y el uso de memoria, aplicando diferentes enfoques similares a los comparados en este trabajo.

La aplicación de matrices para el almacenamiento de las lecturas de sensores con corrección en errores de posicionamiento se inicia en 1989 [17]. Los siguientes abordajes se basan en trabajos, con proyección, en espacios tridimensionales [24] y en propuestas para mejorar la estimación de existencia de objetos en el mundo. Es así que los mapas posteriores, utilizan un enfoque que aplica matrices, con arreglos tridimensionales pero con las mismas correcciones con probabilidad de error en las posiciones [2].

En el año 2002 Sebastian Thrun, presenta un trabajo que aplica mapas de ocupación, capaces de reconocer pequeños objetos que posteriormente aplica en planeamiento de caminos para *google cars* [38]. La propuesta de Yang y Wang, utiliza un enfoque para matrices diferente, en lugar de estimar estados de ocupación en matrices, mantienen estimaciones estocásticas para los estados del ambiente en movimiento [42]. Souza y Gonzalves, proponen una estructura para mapas de ocupación, que almacenan 3 valores sobre el espacio que existe. Pudiendo generar un terreno en 3D, por la conversión de arreglos tridimensionales convertidos a matrices [36]. En los últimos trabajos, proponen comparar el plano con la imagen retornada por el robot para el alineamiento del gráfico por medio de SLAM [21].

Cuadro 3.1: Evolución de mapas de ocupación

Año	Autores	Aporte	Congreso
1989	Elfes	Primer modelo de sensores para mapas de ocupación en 2D, los mapas fueron mejorados especialmente en locaciones de tipo espejo donde había errores de posicionamiento en sensores de profundidad.	Publicación [17]
1992	Leonard, Durrant−Whyte y Cox	Primera implementación de modelos para sensores en mapas de ocupación en 3D, con sensores de proyección utilizando nuevas técnicas. Se utilizaba mucha memoria para mapas del tipo matriz.	EEE International Workshop on Intelligent Robots andSystems [24]
1996	Borenstein, Everett y Feng	Primera generación de mapas de ocupación en 3D, para visión estereoscópica. El problema en cuanto a la memoria en este trabajo, se encuentra en la memoria utilizada, para almacenar todos los datos que van a ser actualizados. Se aplican matrices con tamaños muy grandes para los mapas	Publicación [3]
2002	Thrun	Primera combinación de luz texturizada, trinocular estereoscópica para mapas de ocupación y reconocimiento de pequeños objetos.	Communica-tions of the ACM [38]
2011	Yang y Wang	Por medio de modelamiento de sensores, en lugar de asumir un mundo estático con matrices de ocupación, se enfocan en el movimiento de los objetos. Las matrices mantienen estimaciones estocásticas que estiman el ambiente.	IEEE International Conference on Robotics and Automation (ICRA) [42]

continúa en la siguiente página

Cuadro 3.1 continúa en la siguiente página

Año	Autores	Aporte	Congreso
2015	Souza y Gonzalves	Se crea la estructura occupancy-elevation Grid (OEG), donde cada celda en el mapa de ocupación, almacena la probabilidad, la elevación del terreno y su varianza. Con mejor uso de memoria, por la aplicación de vectores tridimensionales.	Publicación [36]
2017	Kakuma, Tsuichihara, Ricardez, Takamatsu, Ogasawara	Proponen un método que utiliza la coincidencia de gráficos para alinear un mapa de ocupación con un mapa de cuadrícula de ocupación generado por SLAM.	Publicación [21]

3.1.2. SLAM

El cuadro 3.2 muestra cómo ha ido evolucionando SLAM, a través de los años. La evolución de SLAM, surge a raíz de un congreso donde se plasman los primeros enfoques teóricos sobre la solución posible a los problemas que se tenían para localización de un robot y la generación del mapa [17]. En este trabajo se presentó una propuesta que muestra la similitud entre, la localización y uso de probabilidades para relacionar un mayor número de cálculos con posiciones en marcos [33].

Crowley, propone en 1989 relacionar la posición del robot con estados previos y posteriores. A raíz de este enfoque Sebastian Thrun, presenta la aplicación de filtros de Kalman para la localización pero aplicando el uso de probabilidad en cada marco retornado, para que lograrán mayor precisión [14].

Feder inicia trabajando con mapas topológicos, para el almacenamiento de mapas de grandes distancias recorridas y las actualizaciones con las que se debe trabajar [12]. En el año 2002, en Suecia diferentes investigadores trabajan en conjunto para resolver el problema de SLAM. Estableciendo una serie de pasos, que definen la teoría en cuanto a SLAM. Aperturando, la investigación por medio de lo definido para generar diferentes técnicas en base a su planteamiento formal [5].Es así que David W. Murray, aplica otro tipo de sensores para acoplar la visión computacional [15].

Los trabajos posteriores, incluyen tanto la odometría, para posicionamiento, y filtros de partículas para generar ambientes tanto para mapas como para la visualización de los

mismos [28]. Surgen en el 2010, los primeros mapas en 3D de los espacios recorridos, tanto para rescate en minas y diferentes lugares [34].

Con el nacimiento de diferentes tipos de robot, tanto drones, humanoides de forma comercial, inicia la investigación de poder solucionar SLAM, para múltiples tipos de robots [16].

Ya en el 2017 combinan la odometría para la combinación de puntos en los espacios recorridos por el robot. Con mejoras para la luz, uno de los principales problemas en mapeo de ambientes debido a su aplicación en zonas donde las personas no pueden ingresar [41].

Lo anterior muestra que las investigaciones relacionadas a SLAM en los últimos años se han enfocado en incluir modelos 3D y diferentes tipos de robots, optimizando la percepción y no la estructura de almacenamiento, lo que hace de la propuesta de esta tesis importante en el sentido de que retoma la discusión sobre las estructuras de almacenamiento permitiendo aprovechar las ventajas de los modelos topológicos y de mapas de ocupación.

Cuadro 3.2: Evolución de SLAM

Año	**Autores**	**Aporte**	**Congreso**
1986	Peter Cheeseman,Jim Crowley y Hugh Durrant-Whyte	Se mostraron métodos teóricos de estimación para problemas de localización y mapeo, se generan cuestionamientos computacionales para ser aplicados.	Robótica y Automatización, San Francisco USA[17]
1987-1988	Smith y Cheesman	Establecen una base estadística para la relaciones entre marcos y la manipulación geométrica en las regiones de incerteza. El elemento clave en los trabajos, fue mostrar que existe un elevado grado de correlación entre las estimaciones de la localización de diferentes marcos en un mapa y la correlación con sucesivas observaciones.	Publicación [33]

continúa en la siguiente página

Cuadro 3.2 continúa en la siguiente página

Año	Autores	Aporte	Congreso
1989	Crowley	El gran aporte se da en que una posible solución completa para el problema de localización y mapeo simultáneo exige un conjunto compuesto por el estado del robot y de las posiciones de los marcos observados. Y cada marco observado de todo el sistema debe ser actualizado.	Publicación [14]
1998	Thrun	Presentan métodos probabilísticos mapeo y filtro de Kalman basados en la localización.	Simposio Internacional sobre SLAM [12]
2000	Feder	Muestran una forma eficiente para SLAM en entornos de gran tamaño, muestran una forma eficiente para resolver SLAM en ambientes grandes.	Conferencia Internacional IEEE de Robótica y Automatización [5]
2002	Hugh Durrant-Whyte, John Leonard, Juan lento, Raja Chatila, Sebastian Thrun y Wolfram Burgard.Deciden unirse para resolver el problema SLAM, con un conjunto de 50 investigadores.	Escuela de Verano Estocolmo-Suecia [15].	
2003	David W. Murray	Introducen una nueva dimensión a la solución SLAM usando cámara. Mostró resultados utilizando visión activa y de una sola cámara.	Escuela de Verano Sydney-Autralia [28]

continúa en la siguiente página

Cuadro 3.2 continúa en la siguiente página

Año	Autores	Aporte	Congreso
2005	Armando J. Sousa , Paulo Jose Costa, A. Paulo Moreira, A. S. Carvalho	Une dos tipos de odometría (mécanica y visual), utilizando un filtro de partículas para estimar las velocidades lineales y angulares del robot utilizando la predicción de la odometría mécanica y una fase de actualización en la odometría visual.	Publicación [34]
2010	Sabry F. El-Hakim, Pierre Boulanger	Creación de un sistema diseñado para generar un modelo indoor de un ambiente, de forma precisa y en 3D. Para túneles de minas y construcciones, provisto de intensidad de imágenes y combinación.	Publicación [16]
2016	Saeedi, Sajad, Trentini, Seto	Revisa los sistemas de múltiples robots más modernos, con un enfoque principal en SLAM de múltiples robots	Publicación [31]
2017	Shichao Yang, Sebastian Scherer	Presentan un algoritmo de odometría que combina puntos y bordes. Funciona mejor en ambientes sin textura y también es más robusto a los cambios de iluminación y al movimiento rápido de convergencia.	Publicación [41]

3.1.2.1. Diferentes enfoques para SLAM

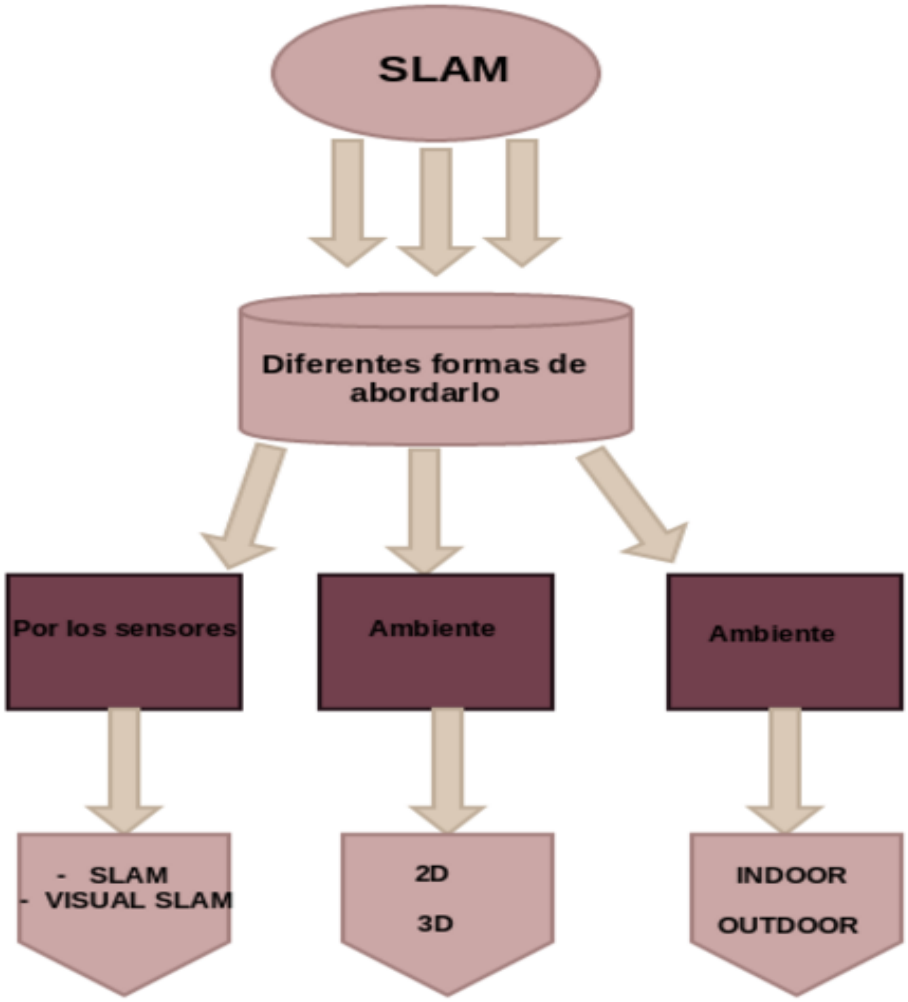


Figura 3.1: Diferentes formas de abordar SLAM (Fuente: Creación propia)

Se ha iniciado con la historia y trabajos de investigación para finalizar con un resumen del enfoque presentado en este trabajo. El principal problema de abordaje para SLAM, se encuentra en trabajar sin cámaras o referencias de imágenes aplicadas en sensores de diferentes tipos, sensores láser, sensores de profundidad, sensores RGB-D ASUS. Para VISUAL SLAM se aplica visión a partir del año 2003 con diferentes tipos de cámaras desde la monocular, estereoscópica, time *offlight*, profundidad, binocular, monócroma, alta resolución, omnidireccional, fie wire. Inclusive para ambas técnicas podemos hablar que para SLAM Y VSLAM existen técnicas híbridas en su mayoría para asegurar que los resultados son los mejores.

3.1.2.2. Comparación de técnicas aplicadas versus sensores

A continuación se muestran, las técnicas más utilizadas; una de las principales, son los filtros de Kalman, los cuales ayudan a estimar un vector de estado que contiene la configuración del robot y de los parámetros que caracterizan a los errores sistemáticos en la odometría [5].

Cuando hablamos de espacios, para 2D hacemos referencia al espacio en un eje cartesiano ya sea para graficar y para poder crear un diagrama en x y y [18].
Ahí se aplican técnicas matriciales, que retornan valores y un ángulo como la odometría [15].

Al referirnos a los tipos de ambientes, como en la Figura 3.1 debemos tomar en cuenta. Los tipos de errores no sistemáticos, que son causados por la interacción del robot con características impredecibles del medio, cuan resbaloso, cuantas grietas existen que hace que el robot se mueva y no genere mapas adecuados. Los errores sistemáticos, son específicos del robot y no dependen del medio, se basan en las distancias entre las ruedas, los problemas de los motores y estructuras robóticas. Cuando hablamos de ambientes outdoor nos referimos a parques y áreas con personas en ambientes externos. Para los ambientes indoor, se incluye ambientes dentro de espacios que habitamos, donde existen obstáculos de muebles, personas, diferentes tipos de ambientes o la luz, por ejemplo [23].

3.1.3. Sensores utilizados en SLAM

El hardware de un robot es sumamente importante; para realizar SLAM, existe la necesidad de que los robots tengan un dispositivo de medición de rango. Los robots que han sido considerados para este trabajo son robots indoor de ruedas. El enfoque, principalmente es sobre implementación del Software y no explora de forma profunda los modelos robóticos como humanoides, o vehículos autónomos bajo el agua y aviones autónomos.

- Láser: El dispositivo de medición de rango utilizado con mayor frecuencia en trabajos de investigación modernos es generalmente un escáner láser. Son muy precisos, eficientes y la salida no requiere mucho cálculo para procesar. El lado negativo es que también son muy caros. Los costos del escáner SICK son 5000 USD. Los problemas con escáneres láser se dan en ciertas superficies incluyendo vidrio, donde se dan muy malas lecturas (salida de datos). También, los escáneres láser no pueden usarse bajo el agua ya que el agua rompe la luz y el rango es drásticamente reducido [37].

- Sonares: Los sonares se utilizaron intensivamente hace algunos años y bastantes trabajos los aplican hoy en día. Son de menor costo en comparación a los escáneres láser que los desplazaron. Sus mediciones no son muy buenas en comparación con el láser y a menudo dan lecturas imprecisas. Bajo el agua, sin embargo, son la mejor opción y se asemejan a la forma en que los delfines navegan. El tipo usado es a menudo un sonar Polaroid. Originalmente desarrollado para medir la distancia al tomar fotos en cámaras Polaroid [9].

- Cámaras: Tradicionalmente el uso de visión computacional ha sido muy intensa y con errores dependientes de humo y la luz. Dado un cuarto sin luz, un sistema de visión no podría ser utilizado. En años recientes, han habido muchos avances en este campo. Usualmente aplican sistemas estereoscópicos, triclops que miden la distancia. Al utilizar visión, simula como las personas observamos el mundo a nuestro alrededor. También existe mucha más información en una fotografía comparada con un escaneó de láser o de sonar. El manejo de las fotografías solía ser el cuello de botella, porque

los datos necesitaban ser procesados, pero con los avances en algoritmos y poder computacional esto ha dejado de ser un problema [19].

Capítulo 4

MPTE-SLAM

La propuesta de este trabajo, llamada Mapas Provisionales Topológicos Esparsas (MPTE-SLAM), busca solucionar las desventajas en cuanto a mapas de ocupación y mapas topológicos aprovechando los aspectos positivos de cada uno de ellos. Se propone, el uso de una matriz dispersa que almacena en cada nodo un mapa de ocupación. Los nodos del tipo matriz son direccionados; en base al recorrido del robot por medio de los valores obtenidos de la odometría.

Los valores obtenidos por los sensores de profundidad en concatenación con la odometría logran orientar y localizar la posición de un objeto en un espacio del ambiente, tanto por la derecha, izquierda, adelante y atrás. Estos objetos son almacenados en una matriz imitando a los mapas ocupacionales pero tratando los valores con una función de actualización. La función incrementa el valor si se encuentra con el objeto y lo disminuye si no lo encuentra. Si la distancia del objeto, sobrepasa el tamaño del nodo matriz de ocupación se crea otra nueva matriz hacia la dirección del objeto. Obteniendo también el tamaño que debe tener el nodo matriz y el tamaño de la celda en relación al mundo.La propuesta tiene diferentes partes, que han sido divididas por bloques (Figura 4.1), presentados a continuación:

- En el primer bloque se da inicio a la matriz dispersa, generando las filas y las columnas, al dar inicio al programa. Para poder crear los nodos matriz, en el bloque dos.

- Con el bloque dos, se retorna la posición actual del robot y hacia dónde se dirige. El cálculo depende de los valores retornados de la odometría, la cual brinda la posición en 2D. En cuanto a la orientación del robot, también se trabaja con el ángulo de la odometría. Entonces, se conoce la posición del robot y hacia dónde se dirige. Los valores que retornan son números enteros que van a ser generados constantemente, para la posición x, y y radianes para la orientación. En este punto, es donde la parte de probabilidad de SLAM debe incluirse.

- El bloque 3 retorna, distancias de objetos con los sensores de profundidad del robot en el mundo. Los valores deben ser actualizados para generar mapas con probabilidad alta de que retornen el ambiente. Aunque, el mundo para el robot cambie

constantemente, estos estados son almacenados previamente, actualmente y a futuro. Generando posiciones en los nodos matriz que van siendo creados por medio de los valores retornados en la orientación, si percibe los objetos hacia la derecha, arriba o abajo. Todos estos valores son almacenados. Cada recorrido del robot, debe ser actualizado, dando inicio al bloque 4.

- En la etapa del bloque 4, las creencias de los nodos matrices son actualizados. Si en el recorrido del robot, aparece un nuevo objeto debe almacenarlo pero cuando desaparece, se reduce la creencia de su existencia. Si el sensor de profundidad de cualquiera de los puntos del robot percibe un elemento que no puede ser contenido en la submatriz por su tamaño, creamos otra submatriz o varias hasta que contengan la posición correcta de la lectura, esta etapa pertenece al bloque 5.

- Se inicia el bloque 5 como etapa final, al actualizar creencias en submatrices vecinas. Se mejora la ventaja de las matrices de ocupación, porque se ahorra memoria, al crear solamente lo que va existiendo para el robot y también se actualiza. Las submatrices de comportamiento para arriba, abajo, la derecha, izquierda crean en la estructura nuevas submatrices hacia donde nos dirijamos. La actualización de las creencias en submatrices vecinas genera un ciclo, porque la posición va ir incrementando a medida que avanza el robot y debe actualizar las lecturas. Luego se vuelve a generar una posición actual según la odometría y volvemos a repetir el proceso, de crear observaciones por medio de los sensores de profundidad. Se modifican las submatrices volviendo al ciclo por la retroalimentación hasta que se detenga, el programa actualizando los valores de probabilidad.

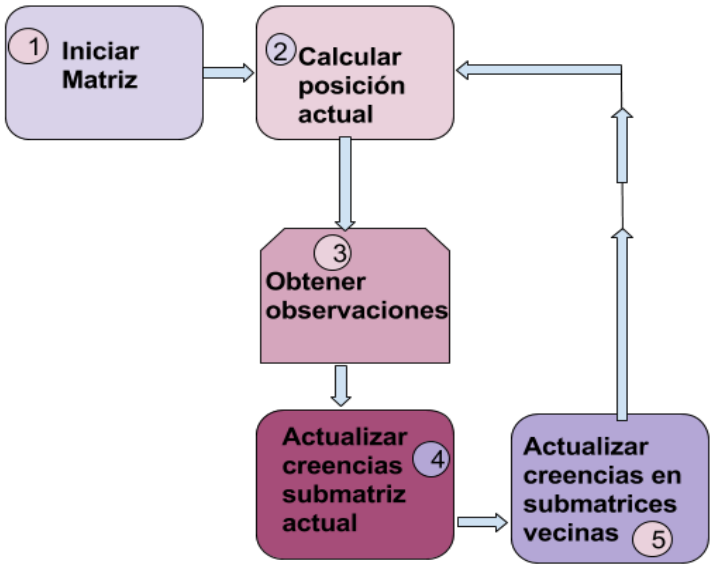


Figura 4.1: Pipeline de la propuesta (Fuente: Creación propia)

4.1. Descripción de Estructuras y Métodos Utilizados

4.1.1. Matriz dispersa

Se propone el uso de matrices dispersas, donde para un nodo matriz se debe poder crecer(agregar nuevos nodos) para sus 4 vecinos los cuales son adelante, atrás, derecha e izquierda. Lo anterior permitirá un crecimiento correlacionado a la navegación de un robot móvil. La definición del formato de la matriz es la siguiente sobre cómo debe recibir un tipo de dato matriz. Tiene que recibir el tamaño establecido para el mapa, filas columnas y una variable *len* que permite establecer un número interno de capacidad. Cada nodo es una matriz, que representa un mapa cuadriculado del ambiente recorrido con una variable por cada cuadrado. Los nodos, van conteniendo la información de las funciones que se presentan a continuación, para establecer la visión obtenida del robot en el mundo. En la Figura 4.2 se define, el comportamiento de la matriz dispersa MPTE-SLAM.

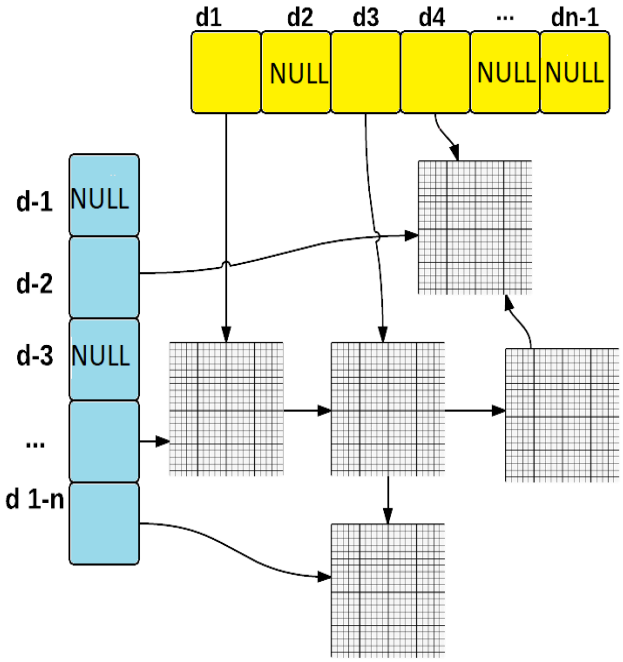


Figura 4.2: Estructura para crear la matriz dispersa (Fuente: Creación propia)

Para mostrar el comportamiento de la propuesta se realizó la comparación, entre la creación de una matriz estática y una con el comportamiento MPTE-SLAM. No es necesario comprar con el almacenamiento en forma de grafos ya que el comportamiento

matemático de una matriz dispersa y un grafo es el mismo (ver sección 2.3). Teniendo definido MPTE-SLAM, se puede comparar con una matriz estática, y porque la propuesta es mejor que el mapa de ocupación tradicional (estática).

En la Figura 4.3 se muestra, una matriz estática y una MPTE-SLAM, donde se incluye la misma distancia, el recorrido y la posición. Las líneas negras, representan los muros de un ejemplo de recorrido dos pasillos y un pasillo largo por donde el robot camina. El punto importante aquí, se encuentra en la lectura del sensor derecho que sigue leyendo a la derecha el fondo del pasillo. La lectura va a ser retornada como posición de lectura de obstáculo pero no podría ser almacenada en la matriz estática por su tamaño, devolviendo un error. Mientras que en MPTE-SLAM sigue generando lecturas a su derecha. Los sensores tienen 6 metros, de distancia generando lecturas en posiciones que no tienen donde almacenarse.

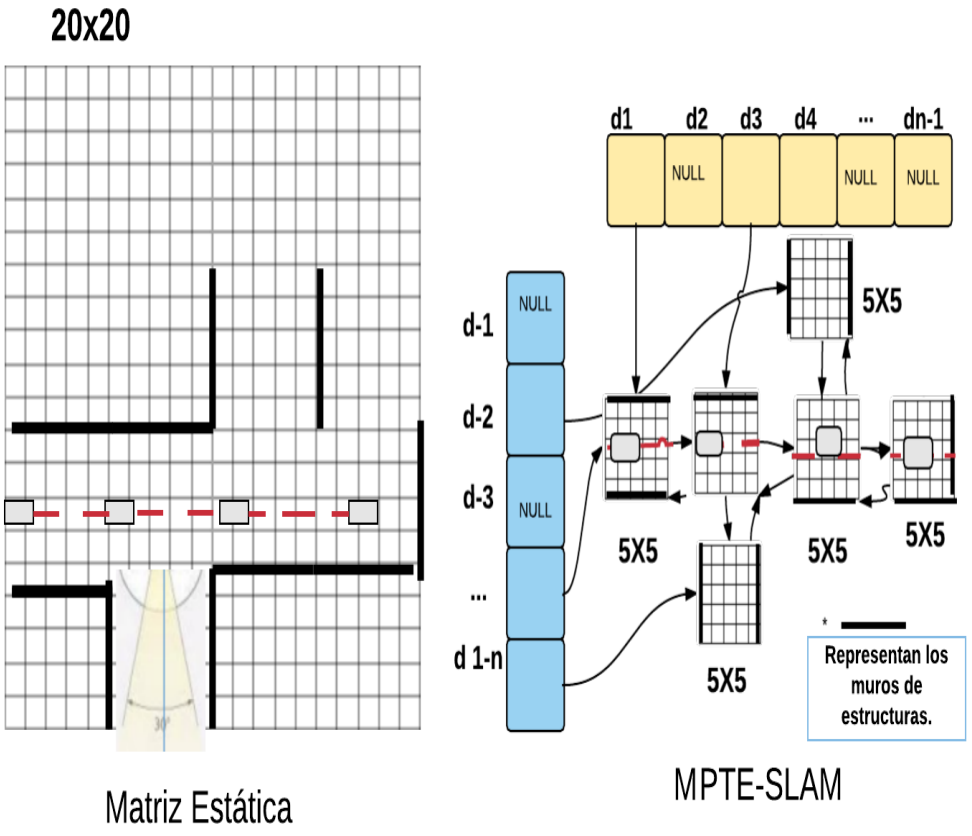


Figura 4.3: Comparación de MPTE y matrices estáticas(Fuente: Creación propia)

La Figura 4.4 al tener el comportamiento de los nodos de MPTE-SLAM, pueden construir el camino hacia los lados. Si, el sensor de profundidad derecho, lee datos de mayor distancia, crea estos nodos almacenando la verdadera posición del objeto (comparando una matriz estática con una MPTE-SLAM). El sensor retorna lecturas que no pueden ser

almacenadas, por tamaño en la matriz estática. Debido a que el mapa no necesariamente crece, por las posiciones de forma ordenada sino por la forma que recorre. En MPTE-SLAM sigue construyendo el mapa, por el crecimiento hacia los lados. El tamaño de los nodos es de 5 x 5 y aún con el pequeño tamaño de almacenamiento en sus nodos, construye el pasillo que se encuentra la derecha. Ahorrando a su vez el espacio que genera la matriz estática, por no contar con un crecimiento inteligente.

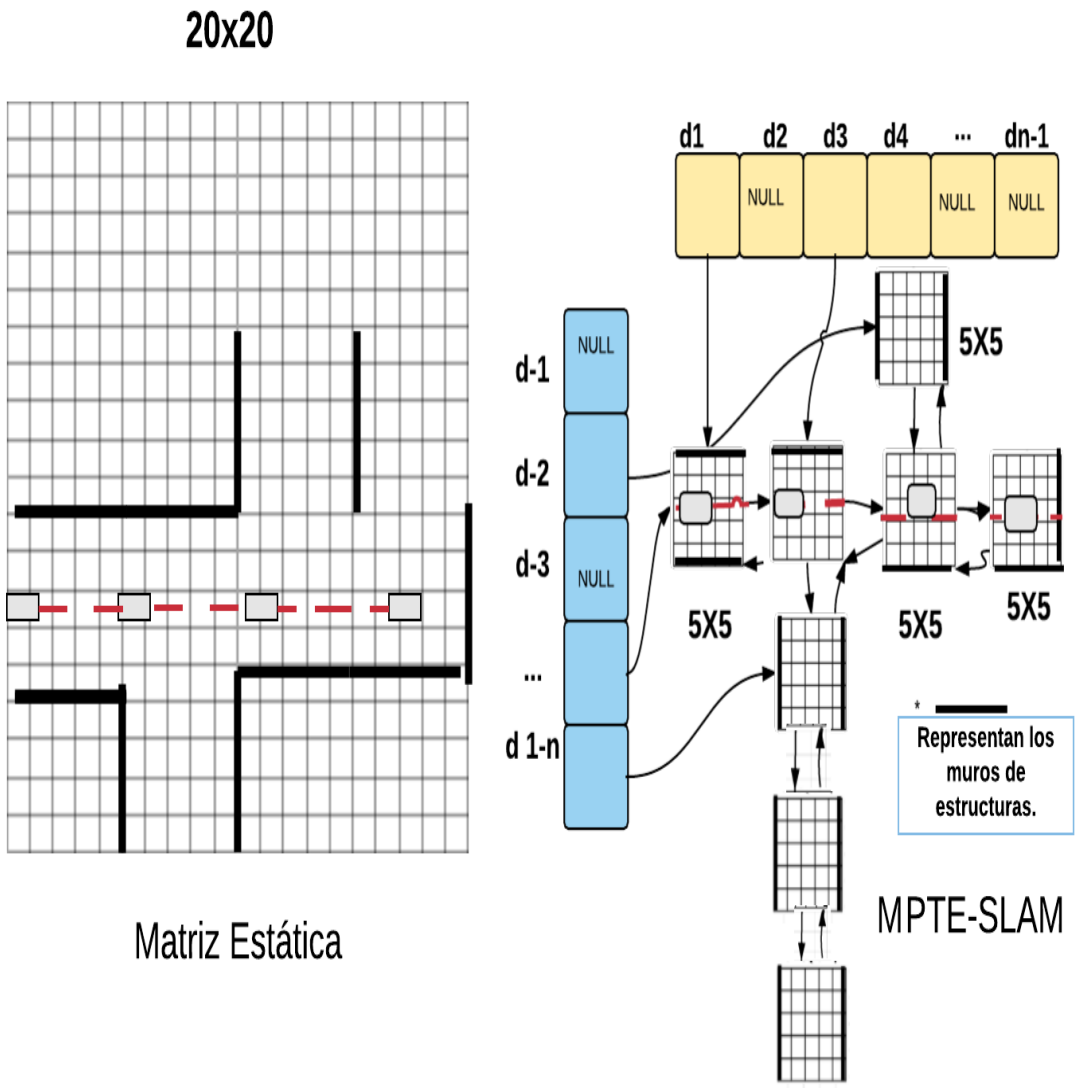


Figura 4.4: Comparación de MPTE y matrices estáticas, en lecturas a 6m (Fuente: Creación propia)

En la Figura 4.5 tenemos que el robot, parte desde 0,0 pero se ha decidido retroceder. La matriz estática no puede trabajar con posiciones negativas que si son retornadas por la odometría. Mientras que la propuesta MPTE-SLAM tiene un comportamiento hacia la izquierda y puede continuar creciendo.

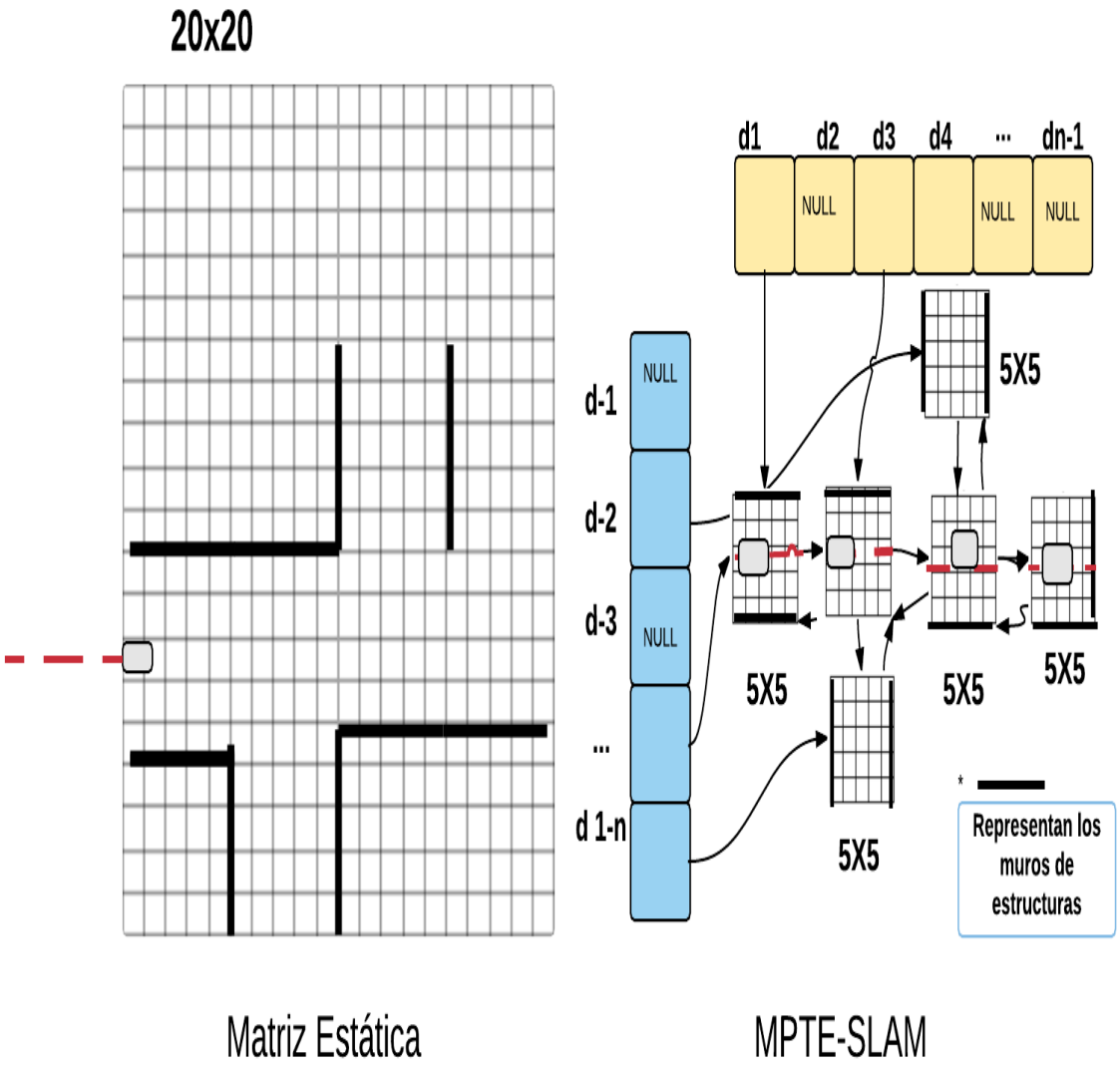


Figura 4.5: Comparación de MPTE y matrices estáticas con robot retrocediendo(Fuente: Creación propia)

4.1.2. Lista Doblemente Enlazada

En cuanto a la orientación para el crecimiento estructural de la matriz se tiene que trabajar con una lista doblemente enlazada, primero un nodo inicial y un nodo final. Debido, a que si el crecimiento va hacia la derecha aún se debe contar con las posiciones pasadas del mapa podremos ver gráficamente los direccionamientos de las actualizaciones en tiempo real hacia todos los lados. Las listas doblemente enlazadas son estructuras de datos semejantes a las listas enlazadas simples pero que pueden retroceder. La asignación de memoria se inicializa al momento de la ejecución. La conexión entre nodos de esta lista se realiza por dos punteros (uno que apunta hacia el elemento anterior y otro que apunta hacia el elemento siguiente). En la Figura 4.6 se muestra el nodo, que es una matriz de tamaño estático.

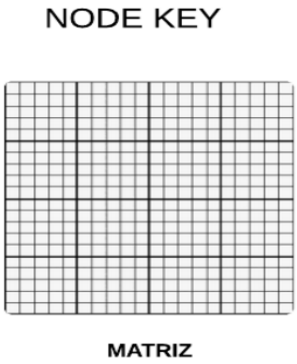


Figura 4.6: Nodo de matriz dispersa (Fuente: Creación propia)

Direccionamiento de los Nodos:

- El puntero anterior del primer elemento debe apuntar hacia NULL (el inicio de la lista).
- El puntero siguiente del último elemento debe apuntar hacia NULL (el fin de la lista).

Para acceder a un elemento, la lista puede ser recorrida en ambos sentidos: comenzando por el inicio, el puntero siguiente permite el desplazamiento hacia el próximo elemento; comenzando por el final, el puntero anterior permite el desplazamiento hacia el elemento anterior. Resumiendo, el desplazamiento se hace en ambas direcciones, desde el primer al último elemento o del último al primer elemento. La Figura 4.7 con el comportamiento de que cada nodo, del tipo matriz, ya cuentan con el desplazamiento, apunta hacia adelante, a la izquierda, derecha y atrás permitiendo el crecimiento de la estructura. Por ejemplo, si la lectura es 60, pero la matriz tiene un tamaño de 30 debe crear hacia su derecha dos nodos matrices.

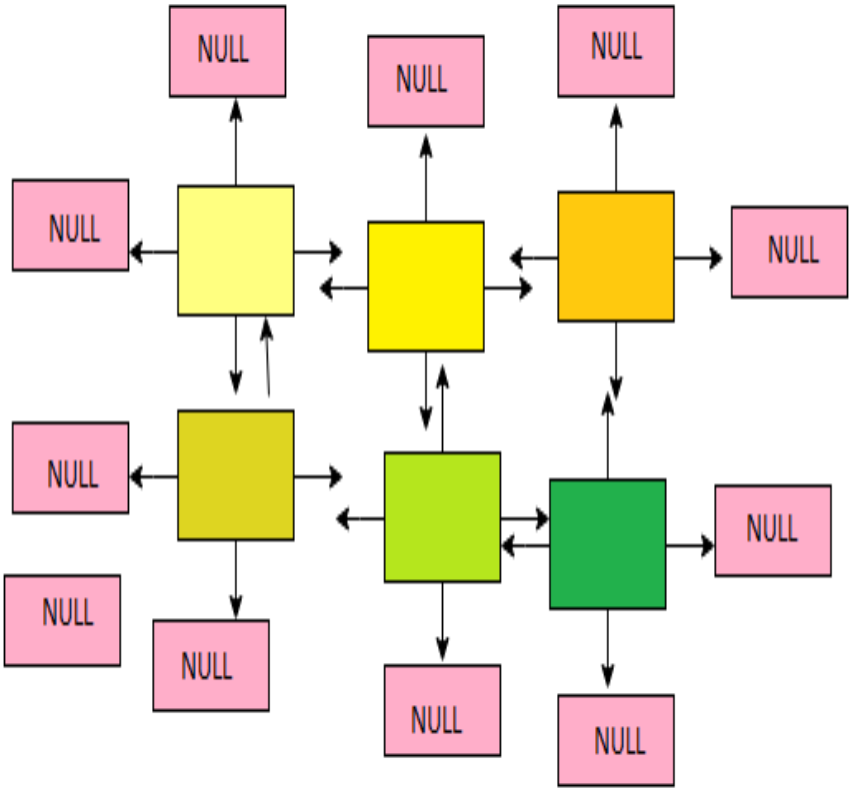


Figura 4.7: Estructura de lista enlazada con comportamiento disperso (Fuente: Creación propia)

4.1.2.1. Actualizaciones

Cada vez que guardamos un valor dentro de alguno de los nodos, los cuales son mapas topológicos, a medida que recorremos un espacio determinado. Ponemos un valor de 1, el cual es el valor es el esperado. Si no encuentra nada en esa misma posición, se va generando un error que se incrementa. La fórmula (4.1) retorna el error en comparación con el esperado y el obtenido, para no darle mucho peso al error. La fórmula (4.2) retorna el nuevo valor donde el alpha también es 0.02 que nos permite tener un acercamiento al mapa real, en cada recorrido. Al momento de modificar se toma la posición donde ha sido obtenida la lectura del objeto, el x y y, se hace la resta del $alpha * 1$ restandole ese valor a la posición de la matriz de ocupación interna en el nodo. Asegurando así que el valor aumente si sigue obteniendo en esa posición ese valor o disminuyéndolo de lo contrario.

$$error = esperado - obtenido \tag{4.1}$$

$$nuevovalor = valoranterior - alpha * error \tag{4.2}$$

4.1.3. Mapa

4.1.3.1. Dibujando el mapa

Una parte importante de la programación de la matriz, luego de solucionar los subproblemas es el método para graficarla. Se ha convertido en tonalidades de grises

retornándola en una matriz final, de un tamaño estático dependiente del recorrido. Los blancos refiere a lo observado en el ambiente, pero que no presenta obstáculos en el ambiente, grises a lo que no se consideró obstáculo constante y negro a lo que se tiene como realmente un posible obstáculo. En la imagen 4.8 se muestra el proceso de la propuesta de una forma gráfica y se explica cada parte separada por letras.

- A) Desde la conexión por medio de wifi entre el robot y la computadora con ROS.
- B) El retorno de los 3 sensores de forma secuencial que permitan trabajar en conjunto.
- C) La estructura de matriz dispersa, que guarda nodos del tipo matriz, retornan mapas ocupacionales; conectados entre sí por medio de una lista doblemente enlazada.
- D) Las actualizaciones se hacen durante cada lectura, aproximando el valor en el nodo matriz, que mayor número de veces se repite en esa posición de la matriz. Por ejemplo si la posición 5,5 tiene algo por varias veces incrementa el valor, mientras que lo reduce si deja de percibirlo en diferentes lecturas en esa posición.
- E) Logrando el objetivo final, el mapa o retorno del ambiente.

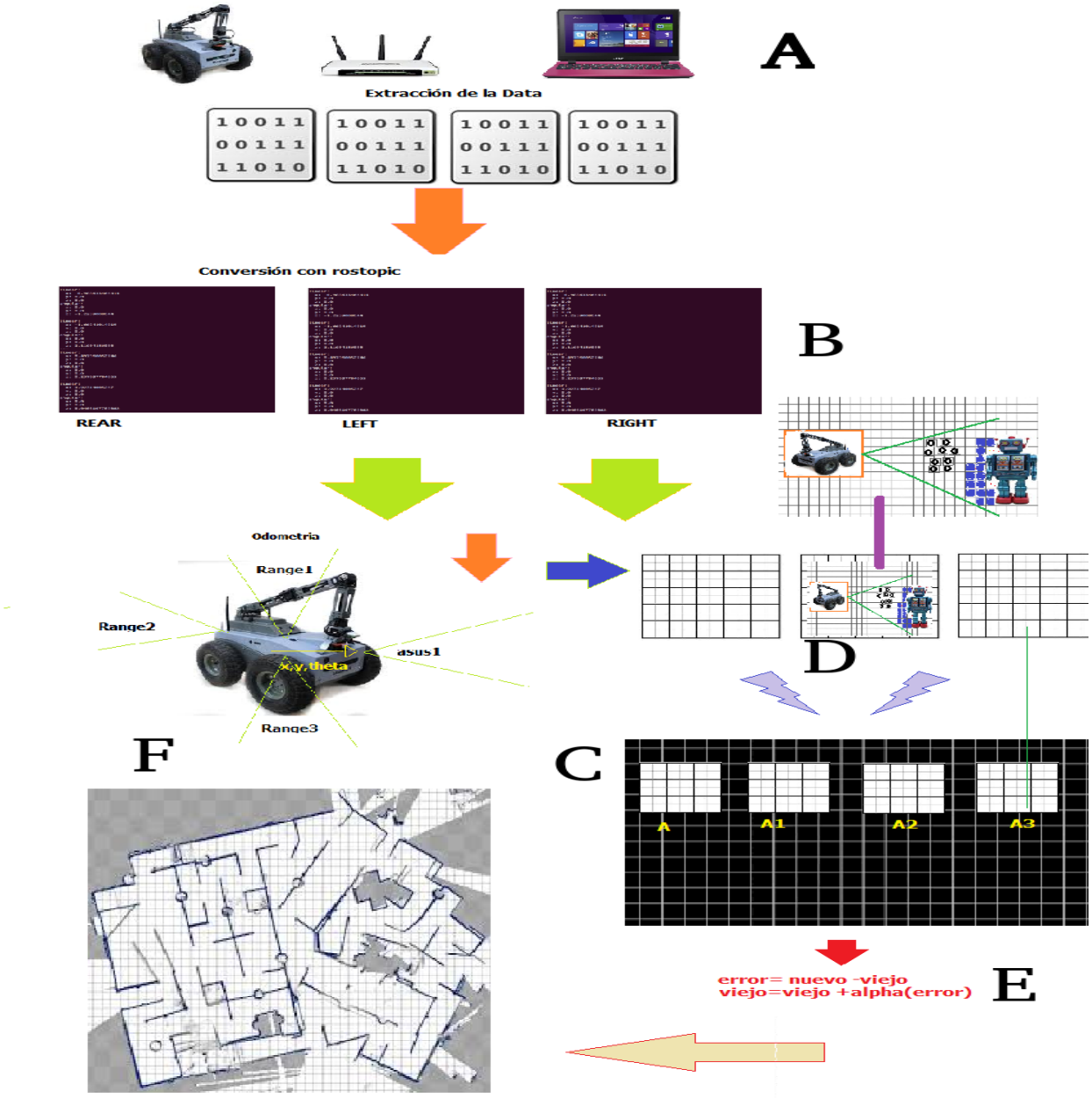


Figura 4.8: Unión de lo realizado en propuesta (Fuente: Creación propia)

Capítulo 5

Implementación

5.1. Características del robot

Komodo es un robot compuesto de tres módulos separados que pueden trabajar de forma independiente y son interconectados a través de una interfaz mecánica y eléctrica. Los tres módulos contienen lo siguiente, el Rover module tiene el chasis de las ruedas de los motores y las unidades de manejo, un controlador, una batería de conducción y un control de unidad remoto.

- El módulo sensor: Asus XTION, la cámara RGB, IMU (3 ejes de acelerómetro, 3 ejes de giro, 3 ejes de magnetómetro), un receptor de GPS, y 3 ultrasonidos y sensores de rango IR.
- El módulo Rover puede ser operado de forma remota controlada por el vehículo. Y los módulos de sensores pueden ser operados independientemente como una unidad de detección.

5.1.1. Módulo Rover

- Cuatro ruedas inflables de manejo diferencial.
- Dos motores de engranajes de 250 vatios.
- Encoders de alta resolución (512 *Pulses Per Revolution* (PPR)) para control de bucle cerrado.
- Peso: 25 kg (con batería).
- Bateria: LiFePO4 24V (26.4V) 12Ah (Toma de carga disponible en el módulo).
- Cuatro canales, 2.4 GHz R/C receptores (Dos canales están disponibles para uso adicional).
- Carga útil: 20 kg.
- Dos grupos de ruedas para terrenos internos (8") y de fuera (10").

5.2. ROS

Robot Operating System (ROS) es un *framework* sensible que permite programar *software* para robots. Es una colección de herramientas, y convencionalidades que simplifican las tareas de creación compleja y robusta para desempeño de robots. Su creación se debe a que el software para *robots* es bastante robusto, complejo en su desarrollo y difícil para estandarizar. Desde la perspectiva del robot, problemas que son triviales para las personas varían entre instancias de tareas y ambientes. Trabajar con estas variaciones es complicado. Como resultado ROS fue creado para la colaboración del desarrollo de software para la robótica. Por ejemplo, si un laboratorio tiene expertos en el mapeo de ambientes indoor, su código se comparte, para la investigación en cualquier otro robot.

5.2.1. Manejo de ROS

Es importante recalcar que la conexión entre el robot y la computadora debe guardar informacioón para ser tratada.

1. La conexión por medio de wifi entre el robot y la computadora, que deben pertenecer a una misma red.
2. Se debe lanzar ROS para poder inicializar la conexión entre ambas partes.

   En esta parte la conexión también se basa en *Internet Provider Security* (IPS). Una vez lanzado podemos decir que el ROS java hydro ha sido comunicado entre ambos de forma satisfactoria.

   ```
   ROSlaunch ric_base_station komodo_base_station.launch
   ```

A Partir de aquí ya se ha lanzado ROS para iniciar el recibimiento de los valores entre el la computadora del robot sus sensores y la computadora con el programa. Se deben habilitar las partes que deseamos conectar en nuestro robot. Que son estas:

```
eloya@eloya-laptop:$ ./read_sensors.sh
/komodo_1/arm_cam_node/image_raw/compressed/parameter_descriptions
/komodo_1/cmd_vel
/komodo_1/Rangers/Left_URF
/komodo_1/Rangers/Right_URF
/komodo_1/Rangers/Rear_URF
/komodo_1/elbow1_controller/state
/komodo_1/elbow2_controller/state
/komodo_1/left_finger_controller/state
/komodo_1/right_finger_controller/state
/komodo_1/shoulder_controller/state
/komodo_1/wrist_controller/state
/komodo_1/imu_pub
/komodo_1/odom_pub
```

Los siguientes puntos permiten levantar la información de estos sensores. El envío de la información también está sujeto a la conexión del brazo y la computadora del robot. En nuestro caso necesitamos la información de los sensores de profundidad. $LeftURF$, $RightURF$, $RearURF$, que retorna lo que percibe el robot a medida que va recorriendo el nuevo espacio. Esta información dentro de nuestra matriz nos permitirá generar obstáculos para ir generando un mapa más preciso.

3. Para finalizar utilizamos el $ROStopiclist-v$ para empezar a observar los parámetros con los que cuenta el robot, el retorno de la información de los sensores.

5.3. Descripción de los sensores:

5.3.1. Características de los sensores del robot

La propuesta en este trabajo, es un nuevo enfoque de matrices de ocupación usando una matriz dispersa, de submatrices que almacenen información de lecturas en sensores. Donde las submatrices involucradas almacenan los obstáculos y los no-obstáculos que retorne el ambiente.
Komodo es un robot compuesto de tres módulos separados que pueden trabajar de forma independiente y son interconectados a través de una interfaz mecánica y eléctrica. Con ultrasonic range sensors (derecha, izquierda, atrás), *Inertial Measurement Unit* (IMU) 3-axis accelerometer, gyro y magnetometro. Con dos motores de 250W alta resolción de encoders de 512 PPR.

5.3.2. Odometría

Gracias a los encoders en los motores del Komodo podemos determinar la odometría. Los valores odométricos como ya se explicó en el marco teórico nos permiten el retorno del recorrido de nuestro robot, en un espacio determinado y un tiempo determinado si manejamos las elipses de error formadas. Para esta parte en especial necesitamos el retorno de tres valores específicos que ya se plantearon en el marco teórico [x, y, θ]. Estos valores nos permiten tener también un punto referencial y la orientación del robot. Así localizamos al objeto en una dirección determinada (derecha o izquierda).

5.3.3. Obtención de los objetos y obstáculos del ambiente

Debemos estimar en donde se encuentra el obstáculo, pero lo fundamental. Se encuentra en lograr que en tiempo real, se pueda percibir los objetos en la posición correcta, que sería simular la visión. Esto se obtiene gracias a los sensores de profundidad que en conjunto nos dan una perspectiva por encima del ambiente. El principal problema, fue definir el ángulo de visión del sensor de profundidad. El cual, se definió entre 29 y 30 grados porque ese es el campo de apertura del sensor.

Las lecturas sobre profundidad con objetos, comprometen solamente al sensor de profundidad de la posición de esa lectura en el sonar, los cuales están al lado izquierdo derecho y en la parte posterior. En la Figura 5.2 se muestra como se obtuvo no solo los θ necesarios y su correspondencia en ángulos. Los valores de retorno de la odometría al final son x, y, θ. Pero θ se ha obtenido de convertir los cuaterniones. Un cuaternión, está formado por cuatro componentes $(q0, q1, q2, q3)$ que representan las coordenadas del cuaternión en una base (e, i, j, k). Los cuaterniones proporcionan una técnica de medición alternativa que no sufren del bloqueo de ejes. Los ángulos de Euler están limitados por un fenómeno llamado bloqueo de ejes, lo que les impide medir la orientación del robot cuando el ángulo de inclinación se aproxima a +/-90 grados. Los cuaterniones se resumen también como un vector de cuatro elementos que puede ser utilizado para codificar cualquier rotación en un sistema de coordenadas 3D, recordemos que el espacio de configuración de un robot está en 3D. Técnicamente, los cuaterniones se componen de un elemento real y tres elementos complejos y pueden ser utilizados para mucho más que solo rotaciones. El frame interno es el eje coordenado fijo a la tierra, esta coordenada es definida en eje x apuntando al norte, el eje y apunta al este y el eje z apunta hacia abajo como mostramos en la Figura 5.1.

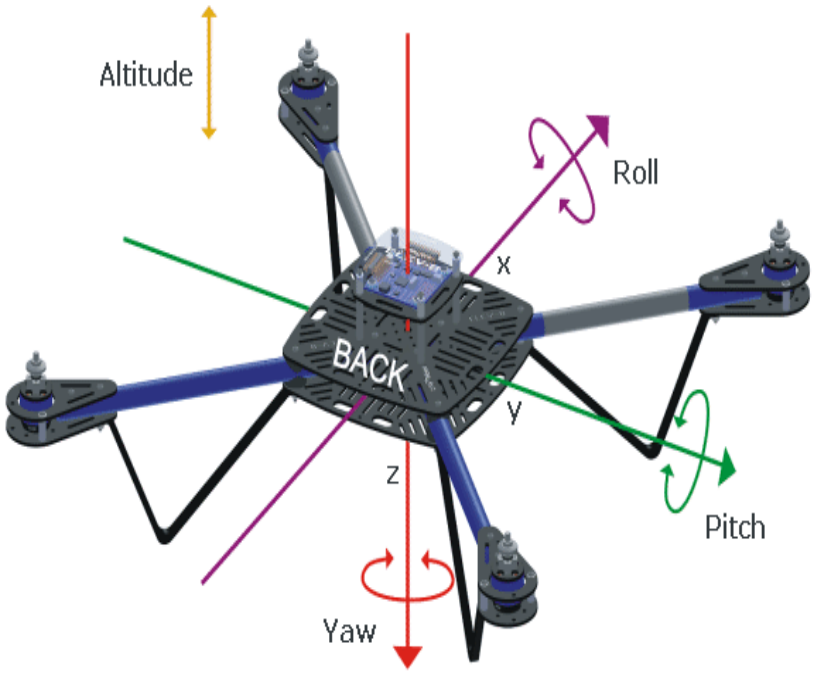


Figura 5.1: Ángulos de Euler con la similitud de Tait-Bryan [8]

El espacio de trabajo utilizado en este espacio, es en 2D por lo tanto tomaremos solamente la orientación en x y y. Obteniendo el ángulo en grados de la rotación de cuaterniones, se incluye la conversión en grados del cuaternión normalizado. Convirtiendo todo en el ángulo de orientación, para la odometría. Es importante tomar en cuenta que *arctan* y *arcsin* son funciones implementadas en lenguajes de computación, debido a que solamente produce resultados de $-\pi/2$ y $\pi/2$ para tres rotaciones entre $-\pi/2$ y $\pi/2$, con todas las posibles orientaciones. Para generar todas las orientaciones se necesita reemplazar la función arctan en los programas por atan2, como se muestra, en la fórmula (5.1), estas conversiones nos permiten obtener yaw, pitch y roll. Para generar todas las orientaciones se ha definido la fórmula (5.2) y la fórmula (5.3).

$$\begin{pmatrix} \phi \\ \theta \\ \Psi \end{pmatrix} = \begin{pmatrix} arctan2(q_0 * q_1 + q_2]q_3/1 - 2(q_1^2 + q_2^2) \\ arcsin2(q_0 * q_2 - q_3 * q1) \\ arctan2(q_0 * q_3 + q_1 * q_2)/1 - 2(q_2^2 + q_3^2) \end{pmatrix} \tag{5.1}$$

$$arctan(2, 0 * (y_1 * z_1 + w_1 * x_1), w_1 * w_1 - x_1 * x_1 - y_1 * y_1 + z_1 * z_1) \tag{5.2}$$

$$atan2(2, 0 * (y_1 * z_1 + w_1 * x_1), w_1 * w_1 - x_1 * x_1 - y_1 * y_1 + z_1 * z_1) \tag{5.3}$$

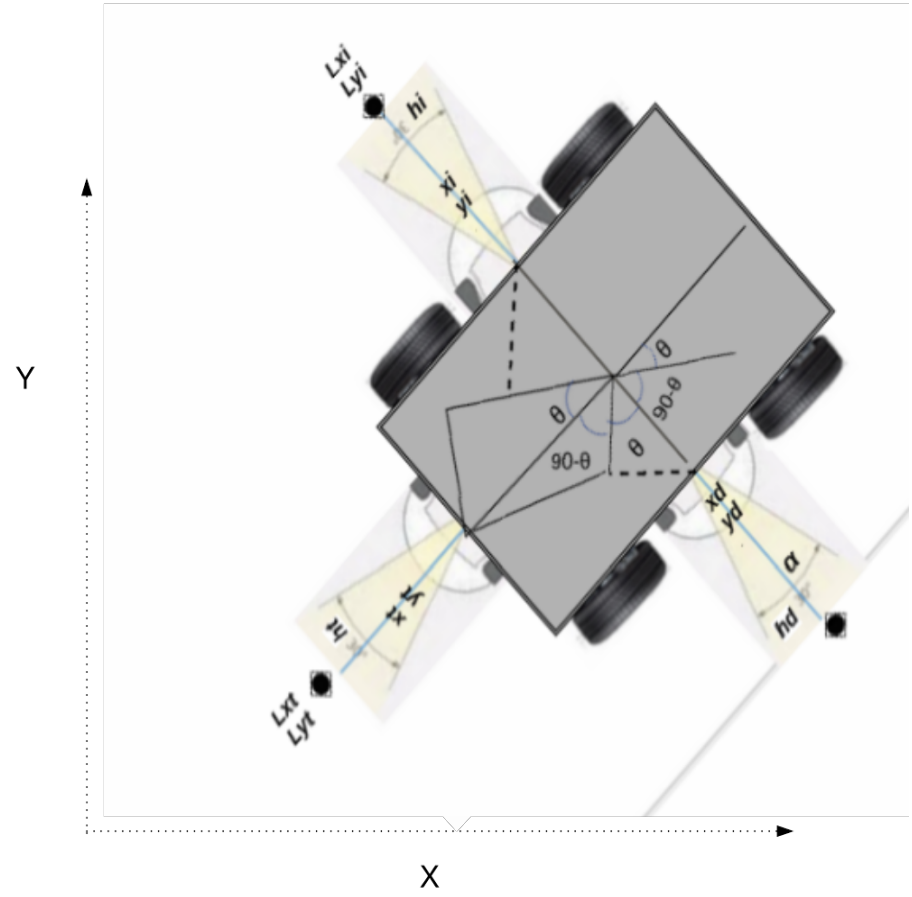


Figura 5.2: Obtención de ángulos y sensores con odometría (Fuente: Creación propia)

Aplicando luego el ángulo de rotación convertido, el resultado ahora es el θ de la odometría, definimos por lo tanto lo siguiente:

1. En cuanto a los ángulos de la Figura 5.2 se aplica la correspondencia de los ángulos basándonos en que son triángulos rectángulos. Conocemos el área que suma 90 grados y a partir de eso nos guiamos a 180 grados, encontrando los valores para cada caso.

2. Los valores agregados a las sumas o restas de cada valor según el punto en x o y, dependen de la posición en ese momento de los valores que retorna la odometría. La suma y la resta dependen del cuadrante en el que se encuentren en el diagrama cartesiano, para asignarle un signo.

$$\theta d = \theta - \pi/2 \tag{5.4}$$

$$\theta i = \theta + \pi/2 \tag{5.5}$$

$$\theta t = \theta - \pi \tag{5.6}$$

3. Para el sensor de la derecha, hemos llamado, a las lecturas en la Figura 5.2 con la letra d.

$$yd = y - \Delta yd \tag{5.7}$$

$$\Delta yd = (cos\theta)(distanciasensorderecha/2) \tag{5.8}$$

$$xd = x + \Delta xd \tag{5.9}$$

$$\Delta xd = (sen\theta)(distanciasensorderecha/2) \tag{5.10}$$

4. Para el sensor trasero(rear), hemos llamado a las lecturas en la Figura 5.2 con la letra t.

$$yt = y - \Delta yt \tag{5.11}$$

$$\Delta yt = (sen\theta)(distanciasensortrasero/2) \tag{5.12}$$

$$xt = x - \Delta xd \tag{5.13}$$

$$\Delta xt = (cos\theta)(distanciasensorderecha/2) \tag{5.14}$$

5. Para el sensor izquierdo , lo hemos llamado en la Figura 5.2 con la letra i.

$$yi = y + \Delta yi \tag{5.15}$$

$$\Delta yi = (cos(\pi/2) - \theta)(distanciasensortrasero/2) \tag{5.16}$$

$$xi = x - \Delta xi \tag{5.17}$$

$$\Delta xi = (sen(\pi/2) - \theta)(distanciasensorderecha/2) \tag{5.18}$$

6. Ahora debemos encontrar (como un radar de barco), los objetos percibidos por el sensor, agregando los valores obtenidos previamente para no solo encontrar la orientación del objeto sino también la distancia, al objeto. Tomamos de referencia desde el -15 al 15, 75 al 105 para el izquierdo y del -105 al -75 derecho, que en conjunto forman 30 grados si se r, en este caso el sensor de profundidad trasero es tomado como referencia de las fórmulas debido a que es lo mismo para los otros sensores. Los valores de los ángulos convertidos a radianes en esa parte, son los α. Lxd y Lyd representan al sensor derecho, Lxi y Lyi representan al sensor izquierdo, Lxt y Lyt al sensor de profundidad trasero.

7. En las fórmulas de 5.19 hasta 5.24 el valor de hd representa el valor obtenido por medio del sensor de profundidad para cualquiera de los casos. Hacemos esta notación para las fórmulas siguientes que presenten este valor. Por otro lado el valor de distancia se refiere a la forma en que vamos a discretizar las celdas del mapa.

$$Lxd = (xd + sen(\theta + \alpha) * (hdd))/valordedistancia \tag{5.19}$$

$$Lyd = (yd - cos(\theta + \alpha) * (hdd))/valordedistancia \tag{5.20}$$

$$Lxi = (xi + sen(\theta + \alpha) * (hdi))/valordedistancia \tag{5.21}$$

$$Lyi = (yi - cos(\theta + \alpha) * (hdi))/valordedistancia \tag{5.22}$$

$$Lxt = (xt + sen(\theta + \alpha) * (hdt))/valordedistancia \tag{5.23}$$

$$Lyt = (yt - cos(\theta + \alpha) * (hdt))/valordedistancia \tag{5.24}$$

5.4. Captura de la información

5.4.1. Sistema operativo del Komodo

La idea principal sobre la aplicación de ROS se encuentra, en que al ser instalado en un sistema robótico (computadora) permite tener un desempeño de computadora a computadora en nuestro caso, con conexión wifi. ROS es un framework que se utiliza ampliamente en la robótica. Puede ser aplicado en otro tipo de robots, con modificaciones en el código, en el caso del sensor Kinect se trabaja con el mismo paquete de OpenNi. Es un *framework open-source*, con un sistema meta operativo capaz de ser compilado en diferentes plataformas de robots. Es utilizado por múltiples laboratorios de investigación, brinda mucha funcionalidad en diferentes niveles. En un inicio, corre bajo plataformas basadas en Unix. Con una conexión de red del tipo *peer-to-peer*, inicializa su trabajo de procesos independientes diferentes y hosts. Ros es capaz de utilizar múltiples lenguajes: La comunicación está basada en *Extensible Markup Language Remote Procedure Call* (XML-RPC), soporta $c++$, Python, octave, LISP, Java, pueden definirse tipos de datos, mensajes de prioridad de procesos. Periféricos que soportan ROS:

- Cámaras
- Sensores Ultrasónicos, tacto, torque.
- IMU/*Global Positioning System* (GPS)
- Reconocimiento de audio y discurso.
- *Radio-Frequency Identification* (RFID)
- Arduino

5.4.2. Lenguaje Escogido

Para lograr la comunicación entre el código y ROS (como framework) se escogió, de lenguaje java y de entorno Eclipse. Eclipse nos permite trabajar la interfaz para la transformación de la información y su almacenamiento. La conexión entre el código y las lecturas se logra, a través de la conexión por medio de sus cabeceras.

5.4.3. Captura de la información

Es importante recalcar que la conexión entre el robot y la computadora debe guardar información para ser tratada. Este punto nos permite tener los datos de todos los sensores con los que estamos trabajando.

1. La conexión por medio de wifi entre el robot y la computadora debe pertenecer a una misma red.

2. Se debe lanzar ROS para poder inicializar la conexión entre ambas partes.

   En esta parte la conexión también se basa en IPS. Una vez lanzado podemos decir que el ROS java hydro ha sido comunicado entre ambos de forma satisfactoria.

   ```
   ROSlaunch ric_base_station komodo_base_station.launch
   ```

   A Partir de esta parte, ya se ha lanzado ROS para iniciar el recibimiento de los valores entre la computadora del robot, sus sensores y la computadora con el programa. Se deben habilitar las partes que deseamos conectar en nuestro robot, las cuales son:

```
eloya@eloya-laptop:~/Documents/sensores/sensors$ ./read_sensors.sh
/komodo_1/arm_cam_node/image_raw/compressed/parameter_descriptions
/komodo_1/cmd_vel
/komodo_1/Rangers/Left_URF
/komodo_1/Rangers/Right_URF
/komodo_1/Rangers/Rear_URF
/komodo_1/elbow1_controller/state
/komodo_1/elbow2_controller/state
/komodo_1/left_finger_controller/state
/komodo_1/right_finger_controller/state
/komodo_1/shoulder_controller/state
/komodo_1/wrist_controller/state
/komodo_1/imu_pub
/komodo_1/odom_pub
```

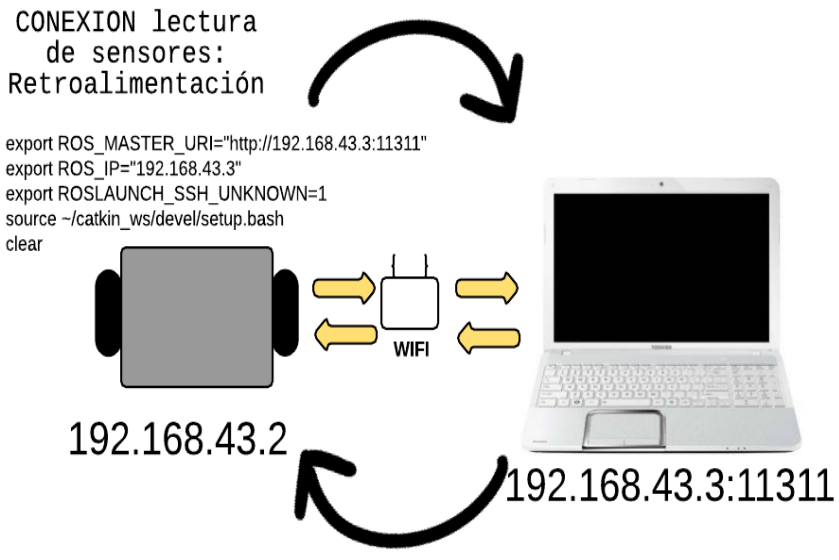


Figura 5.3: Conexión para retroalimentación (Fuente: Creación propia)

5.4.3.1. Retroalimentación

Por medio de ROS se obtiene las lecturas de los sensores que están conectados a la computadora del robot y pasan, a la maestra la cual lanza el programa. Para poder trabajar con los datos en tiempo real, se ha provisto de un sistema de comunicación, que a través de un router permite la conexión entre el robot y la computadora. Con sus propias IPS y una máscara de red, como se muestra en la Figura 5.3.

Los siguientes puntos permiten levantar la información de estos sensores. El envío de la información está sujeta a la computadora del robot. En nuestro caso necesitamos la información de los sensores de profundidad. Para obtener la información de esta conexión, necesitamos llamar diferentes tópicos en el código para obtener las lecturas.

```
lecture.Load("komodo_1.odom_pub");
reader_rear.Load("/komodo_1/Rangers/Rear_URF");
reader_right.Load("/komodo_1/Rangers/Right_URF");
reader_left.Load("/komodo_1/Rangers/Left_URF");
```

Left URF, Right URF, Rear URF, que nos avisa que es lo que percibe nuestro robot a medida que va recorriendo el nuevo espacio. Por medio de la declaración de range, nos retorna el rango del valor obtenido para almacenarlo y se almacena en una variable para retornarlo.

```
h_rear = reader_rear.GetDouble("range");
h_right = reader_right.GetDouble("range");
h_left=reader_left.GetDouble("range");
```

5.4.4. Implementación para estructura de MPTE-SLAM

En cuanto a la implementación de la estructura, se presentan a continuación los algoritmos, que definen los comportamientos y el almacenamiento de la misma.

5.4.5. SparseMatrixDataNode

Entre SparseMatrixDataNode que define el comportamiento y también la construcción. En IsInRows nos muestra que esta en las filas, en isInColumns que se encuentra en las columnas, set establece valores y get los retorna. En el Algoritmo 1 se inicia la definición de lo que el nodo del tipo matriz tiene internamente como constructor.

Algorithm 1 SparseMatrixDataNode

```
1: procedure SparseMatrixDataNode(sM, x_, y_)
2:     integer x ← x_
3:     integer y ← y_
4:     integer sizeMat ← sM
5:     C3[sizeMat][sizeMat]
6:     top ← null
7:     bottom ← null
8:     right ← null
9:     left ← null
10: end procedure
```

En el Algoritmo 2 muestra la función isInRow, que retorna en que parte de las filas se encuentra, la posición de la lectura de la matriz. Este comportamiento en el caso que retorne 0, significa que el valor ingresado en x y y, se encuentra en esa posición, para los otros retornos, siendo menor que x debe estar arriba, si x_ es mayor entonces se encuentra abajo y si retorna 5, está dentro de la fila.

Algorithm 2 isInRows

```
1: procedure isInRows( x_, y_)
2:     if (( x_>=x and x_<x+sizeMat) and ( y_>=y
3: and y_<y+sizeMat)) then return 0;
4:     else if x_<x then
5:         return 1;
6:     else if x_>x then
7:         return 2;
8:     else if x_>=x then
9:         return 5;
10:    end if
11: end procedure
```

En el Algoritmo 3 se presenta el comportamiento, para cuando se desee encontrar la posición en columnas. En return 0, se encuentra ahí, en return 3, significa que está a la izquierda, en return 4 se encuentra a la derecha finalizando en return 5, que se encuentra en la columna.

Algorithm 3 isInColumns

```
1: procedure isInColumns( x_, y_)
2:     if (( x_>=x and x_<x+sizeMat) and ( y_>=y
3: and y_<y+sizeMat)) then return 0;
4:     else if y_<y then
5:         return 3
6:     else if y_>y then
7:         return 4
8:     else if y_>=y then
9:         return 5
10:    end if
11: end procedure
```

En el Algoritmo 4 set, busca que los valores no sobrepasen el tamaño de la matriz y que se almacene el valor de la lectura en la posición específica.

Algorithm 4 set

```
1: procedure set( x_, y_, value)
2:     if (( x_>=x and x_<x+sizeMat) and ( y_>=y
3: and y_<y+sizeMat)) then C3[ x_-x][ y_-y]] = value
4:         return true;
5:     end if
6:     return false;
7: end procedure
```

En el Algoritmo 5 se incluye la posición y se obtiene el valor de la matriz, si se encuentra, mientras que, sino retorna un error grande que valide que ahí se encuentra el error.

Algorithm 5 get

```
1: procedure get( x_, y_, value)
2:     if (( x_>=x and x_<x+sizeMat) and ( y_>=y
3: and y_<y+sizeMat)) then
4:         return C3[ x_-x][ y_-y]];
5:     end if
6:     return -999999;
7: end procedure
```

5.4.6. DoubleLinkedList

En esta parte, se incluyen los constructores, en putAtHead, establece en la cabeza de la lista doblemente enlazada, y putAtTail coloca al elemento al final de la lista. En el Algoritmo 6 DoubleLinkedListNode, se define primero lo que tiene dentro, el nodo, de la estructura de lista doblemente enlazada, el comportamiento de la lista doblemente enlazada, define que el nodo tiene una conexión a previo siguiente y el nodo Inicial. En cuanto a la construcción, se define que recibe el nodo de la matriz con un booleano de orientación, que se inicializa.

Algorithm 6 DoubleLinkedListNode

```
1: procedure DoubleLinkedListNode
2:     private DoubleLinkedListNode previous = null
3:     private DoubleLinkedListNode next = null
4:     private SparseMatrixDataNode nodoInicial=null
5: end procedure
6: procedure DoubleLinkedListNode (SparseMatrixDataNode nodoMatriz, boo-
   lean orientation)
7:     orientation = orientation
8:     nodoInicial = nodoMatriz
9: end procedure
```

En el Algoritmo 7, permite agregar información por columnas, recibe el nodo actual, la posición y la sub matriz. El valor que se agrega, como debe agregarse en columnas, se conecta el siguiente al final, creciendo hacia abajo. También almacenamos el valor de la columna en a donde ir, el cual representa la posición. En caso de ser 1, igualamos el de abajo con el siguiente y la submatriz que contiene el nodo matriz. Hacia abajo, se conectara el siguiente a ese nodo y la parte de arriba ahora es el valor actual. En el segundo caso se inserta e la columna, el siguiente nodo en al posición x, y, con la submatriz.

Algorithm 7 addInColumn

```
1: procedure addInColumn(SparseMatrixDataNode actual, int x, int y, SparseMa-
   trixDataNode subMatrix)
2:     SparseMatrixDataNode nextNode= actual.getBottom();
3:     int dondeIr = nextNode.isInColumns (x,y)
4:     Switch (dondeIr)
5:     case1:
6:         SubMatrix.setTop(actual)
7:         SubMatrix.setBottom(nextNode)
8:         actual.setBottom(SubMatrix)
9:         nextNode.setTop(SubMatrix)
10:        break
11:    case2:
12:         addInColumn(nextNode,x,y,subMatrix)
13: end procedure
```

En el Algoritmo 8 insertando en fila, agrega valores como nodo, en la posición co-

rrespondiente en 2D. Debe ser conectado a la derecha, por eso el siguiente nodo, debe estar en la posición derecha. Si el valor es el caso 3, el valor actual está a la izquierda, el siguiente a la derecha y actual se establece a la derecha. En el caso 4, sino se inserta en la fila, el siguiente nodo, ósea una matriz.

Algorithm 8 addInRow

```
1: procedure addInRow(SparseMatrixDataNode actual, int x, int y, SparseMatrixDa-
   taNode subMatrix)
2:     SparseMatrixDataNode nextNode= actual.getRight();
3:     int dondeIr = nextNode.isInRows (x,y)
4:     Switch (dondeIr)
5:     case 3:
6:           SubMatrix.setLeft(actual)
7:           SubMatrix.setRight(nextNode)
8:           actual.setRight(subMatrix)
9:           nextNode.setLeftt(subMatrix)
10:          break
11:    case 4:
12:           insertInRow(nextNode,x,y,subMatrix)
13: end procedure
```

En el Algoritmo 9 insertInColumn, recibe la posición y la submatriz. Establece que dondeIr, es igual al nodo inicial de la columna en la posición x y y, pero para las columnas. En el caso 1, se establece el fondo con el nodo inicial. Colocando el nodo de tipo matriz, en la parte de conexión de fondo y hacia arriba, simulando el comportamiento en columna.

Algorithm 9 insertInColumn

```
1:  procedure INSERTINCOLUMN(int x, int y, SparseMatrixDataNode subMatrix)
2:      int dondeIr = nodoInicial.isInColumns(x,y)
3:      Switch (dondeIr)
4:      case 1:
5:              SubMatrix.setBottom(nodoInicial)
6:              SubMatrix.setTop(subMatrix)
7:              nodoInicial=subMatrix
8:              break
9:      case 2:
10:              insertInColumn(nodoInicial ,x,y,subMatrix)
11: end procedure
```

En el Algoritmo 10 insertInRow, recibe parámetros como la posición en 2D y también el nodo submatriz. En dondeIr, se establece como valor, si se encuentra en las filas. En el caso 3, la submatriz se debe comparar para estar en filas a la derecha y la izquierda. El nodo inicial es la submatriz y se establece el comportamiento de la izquierda.

Algorithm 10 insertInRow

```
1:  procedure INSERTINROW(int x, int y, SparseMatrixDataNode subMatrix)
2:      int dondeIr = nodoInicial.isInRows(x,y)
3:      Switch (dondeIr)
4:      case 3:
5:              SubMatrix.setRight(nodoInicial)
6:              SubMatrix.setLeft(subMatrix)
7:              nodoInicial=subMatrix
8:              break
9:      case 4:
10:              insertInRow(nodoInicial ,x,y,subMatrix)
11: end procedure
```

En el Algoritmo 11 DoubleLinkedList, representa el comportamiento de la lista doblemente enlazada. En este algoritmo parte de la clase como constructor. Definimos que tiene el nodo del tipo matriz, un inicio y un final.

Algorithm 11 DoubleLinkedList

```
1: procedure DoubleLinkedList
2:     public boolean orientation
3:     public EliDoubleLinkedListNode first
4:     public EliDoubleLinkedListNode last
5: end procedure
```

En el Algoritmo 12 DoubleLinkedList, recibe los valores por defecto, donde first es nulo, last es nulo y la orientación que es un booleano se define como falso.

Algorithm 12 DoubleLinkedList

```
1: procedure DoubleLinkedList
2:     integer first ← null
3:     integer last ← null
4:     orientation ← false
5: end procedure
```

En el Algoritmo 13 DoubleLinkedList, establece los parámetros al recibir el booleano con la orientación.

Algorithm 13 DoubleLinkedList

```
1: procedure DoubleLinkedList(boolean orientation)
2:     integerfirst ← null
3:     integer last ← null
4:     orientation ← orientation
5: end procedure
```

En el Algoritmo 14 coloca en la lista doblemente enlazada, en la parte de adelante de la estructura, el contenido. Recibe el nodo del tipo matriz. Si es el único elemento, igualamos el primero al último como posición (con el nodo). Sino, establece el nodo con conexión al siguiente nodo, del nodo que se encontraban allí, finalizando con el nodo como el primero.

Algorithm 14 DoubleLinkedListNode putAtHead

```
1: procedure DoubleLinkedListNode putAtHead(SparseMatrixDataNode node-
   Matrix)
2:     DoubleLinkedListNode node=
3:         new DoubleLinkedListNode (nodeMatrix,orientation)
4:     if (first == null) then
5:         first=last =node
6:     else
7:         first.setPrevious(node)
8:         node.setNext(first)
9:         first=node
10: end if
11: return node
12: end procedure
```

En el Algoritmo 15 putAtTail, recibe el nodo del tipo matriz. Si no existe contenido, el nodo es igual a vacío entonces establecemos el primero como último. Sino, establece, el anterior de ese nodo, al nodo anterior que ya existe y establece el contenido antes del último y el último va a ser el insertado (actual).

Algorithm 15 DoubleLinkedListNode putAtTail

```
1: procedure DoubleLinkedListNode putAtTail(SparseMatrixDataNode node-
   Matrix)
2:     DoubleLinkedListNode node=
3:         new DoubleLinkedListNode (nodeMatrix,orientation)
4:     if (last == null) then
5:         first=last =node
6:     else
7:         last.setPrevious(node)
8:         node.setPrevious(last)
9:         last=node
10: end if
11:         return node
12: end procedure
```

5.4.7. MPTE

En el Algoritmo 16 MPTE, recibe los siguientes valores, para dar forma a la estructura. Recibe dos tipos de listas doblemente enlazadas, para filas y columnas, se inicializan con valores nulos. Teniendo valores máximos para x, para y, valores mínimos para x y mínimos para y. Las cantidades en x y las cantidades en y de nodos matrices. Las filas y las columnas como valores falsos y verdaderos. Estos algoritmos definen el comportamiento de la estructura principal, addUp inserta hacia arriba, addDown hacia abajo, addLeft a la izquierda y representan algunos comportamientos de las funciones. SearchUp, busca

arriba de la estructura al igual que searchUp II, como representación del comportamiento de las búsquedas. Finalizando con set que establece los valores.

Algorithm 16 MPTE

```
1: procedure MPTE
2:     double alpha ← 0,1
3:     DoubleLinkedList rows
4:     DoubleLinkedList columns
5:     DoubleLinkedListNodebaseRows ← null
6:     DoubleLinkedListNodebaseColumns ← null
7:     max_x ← null
8:     max_y ← null
9:     min_x ← null
10:    min_y ← null
11:    amount_x ← null
12:    amount_y ← null
13:    rows= new DoubleLinkedList(false)
14:    columns= new DoubleLinkedList(true)
15:
16: end procedure
```

En el Algoritmo 17 al igualar dondeIr, con la comparación booleana nos retorna 0, si el valor se encuentra ahí. En el caso de que sea 1, donde se encuentra arriba, cuando el caso es 3 está a la izquierda y 4 a la derecha. Para el caso 2, está abajo pero hay que insertarlo porque no existe. Si entra al else, hay que insertarlo más arriba.

Algorithm 17 addUp

```
1: procedure ADDUP(DoubleLinkedListNode nodoActual, int x,int y,SparseMatrixDataNode subMatrix)
2:     DoubleLinkedListNode nodoSiguiente = nodoActual.getPrevious()
3:     int dondeIr
4:     if (nodoSiguiente!=null) then
5:         dondeIr = nodoSiguiente.getDataNode().isInRows(x, y)
6:         Switch (dondeIr)
7:         case 1:
8:               insertandoArriba(nodoSiguiente,x,y,SubMatrix)
9:               break
10:        case 2:
11:              DoubleLinkedListNode nuevo
12:               = new DoubleLinkedListNode(SubMatrix,true)
13:              setPrevious(nodoSiguiente)
14:              nuevo.setNext(nodoActual)
15:              nodoActual.setPrevious(nuevo);
16:              break
17:        case 5:
18:        nodoSiguiente.insertarEnFila(x,y,subMatrix)
19:    else
20:        DoubleLinkedListNode nuevo = new DoubleLinkedListNode(SubMatrix, true)
21:        nuevo.setNext(nodoActual)
22:        nodoActual.setPrevious(nuevo)
23:        minimo_x = x / size_Mat* size_Mat
24:        cantidad_x++
25: end if
26: end procedure
```

En el Algoritmo 18, se hace lo mismo que en el Algoritmo 17 pero cuando se desea agregarlo abajo.

Algorithm 18 addDown

```
1: procedure addDown(DoubleLinkedListNode nodoActual,int x,int y,SparseMatrixDataNode subMatrix)
2:     DoubleLinkedListNode nodoSiguiente = nodoActual.getNext()
3:     int dondeIr
4:     if (nodoSiguiente!=null) then
5:         dondeIr = nodoSiguiente.getDataNode().isInRows(x, y)
6:         Switch (dondeIr)
7:         case 1:
8:         DoubleLinkedListNode nuevo = new DoubleLinkedListNode(SubMatrix, true)
9:         nuevo.setNext(nodoActual)
10:        nodoActual.setPrevious(nuevo)
11:        minimo_x = x / size_Mat* size_Mat
12:        cantidad_x++
13:              break
14:        case 2:
15:              addDown(nextNode, x, y, subMatrix)
16:              break
17:        case 5:
18:        nodoSiguiente.insertarEnFila(x,y,subMatrix)
19:    else
20:        DoubleLinkedListNode nuevo = new DoubleLinkedListNode(SubMatrix, true)
21:        nuevo.setNext(nodoActual)
22:        nodoActual.setPrevious(nuevo)
23:        maximo_x = x / size_Mat* size_Mat-1
24:        cantidad_x++
25: end if
end procedure
```

En el Algoritmo 19 tiene el mismo comportamiento que el Algoritmo 17 pero hacia la izquierda.

Algorithm 19 addLeft

```
1: procedure addLeft(DoubleLinkedListNode nodoActual,int x,int y,SparseMatrixDataNode subMatrix)
2:     DoubleLinkedListNode nodoSiguiente = nodoActual.getPrevious()
3:     int dondeIr
4:     if (nodoSiguiente!=null) then
5:         dondeIr = nodoSiguiente.getDataNode().isInColumns(x, y)
6:         Switch (dondeIr)
7:         case 3: addLeft = nextNOde.getDataNode().isInColumns(x,y)
8:                 break
9:         case 4:
10:        DoubleLinkedListNode nuevo = new DoubleLinkedListNode(SubMatrix, true)
11:        nuevo.setPrevious(nodoSiguiente)
12:        nuevo.setNext(nodoActual)
13:        nodoSiguiente.setNext(nuevo)
14:        nodoSiguiente.setPrevious(nuevo)
15:        cantidad_x++
16:                break
17:        case 5:
18:        nodoSiguiente.insertarColumna(x,y,subMatrix)
19:    else
20:        DoubleLinkedListNode nuevo = new DoubleLinkedListNode(SubMatrix, true)
21:        nuevo.setNext(nodoActual)
22:        nodoActual.setPrevious(nuevo)
23:        minimo_y = x / size_Mat* size_Mat-1
24:        cantidad_x++
25: end if
26: end procedure
```

En el Algoritmo 20 recibe, el nodo actual y la posición en x y y de la matriz nodo. En el caso en que el valor de dondeIr sea 0 al recorrer las filas, significa que si se encuentra. El valor, 1 está arriba, 2 se encuentra abajo, 3 a la izquierda y 4 hacia la derecha. En el caso 5, se encuentra en esa columna, en else hay que agregarlo más arriba, y se vuelve a comparar los valores y también se agrega la submatriz en las columnas.

Algorithm 20 searchUp

```
1: procedure SEARCHUP(DoubleLinkedListNode nodoActual,int x,int y)
2:     DoubleLinkedListNode nodoSiguiente = nodoActual.getNext()
3:     int dondeIr
4:     if (nodoSiguiente!=null) then
5:         dondeIr = nodoSiguiente.getDataNode().isInRows(x, y)
6:         Switch (dondeIr)
7:         case 0:
8:               return nodoSiguiente.getDataNode();
9:         case 1:
10:              return searchUp(nextNode,x,y)
11:        case 2:
12:        SparseMatrixDataNode subMatriz =
13:        new EliSparseMatrixDataNode(size_Mat,(x /
14:        size_Mat * size_Mat,
15:        *y/size_Mat)* size_Mat
16:        nuevo.setPrevious(nodoSiguiente)
17:        nuevo.setNext(nodoActual)
18:        nextNode.setNext(nuevo)
19:        ActualNode.setPrevious(nuevo)
20:        cantidad_x++
21:        dondeIr = baseColumns.getDataNode().isInColumns(x, y);
22:        switch (dondeIr)
23:        case 3:
24:              insertandoIzquierda(baseColumns,x,y,subMatriz);
25:              break;
26:        case 4:
27:              insertandoDerecha(baseColumns,x,y,subMatriz);
28:             break;
29:        case 5:
30:              baseColumns.insertarEnColumna(x,y,subMatriz);
31:        return subMatriz
32:    end if
33: end procedure
```

Algorithm 21 searchUp Part II

```
1: procedure SEARCHUP(DoubleLinkedListNode nodoActual,int x,int y)
2:     Else
3:     SparseMatrixDataNode subMatriz = new
4:     SparseMatrixDataNode(size_Mat,x/size_Mat)* size_Mat),
5:     (y/size_Mat)* size_Mat)
6:     DoubleLinkedListNode nuevo = new DoubleLinkedListNode(subMatriz, true)
7:     nuevo.setNext(nodoActual)
8:     nodoActual.setPrevious(nuevo)
9:     cantidad_x++ dondeIr = baseColumns.getDataNode().isInColumns(x, y)
10:    Switch (dondeIr)
11:    case 3:
12:            addLeft(baseColumns,x,y,subMatrix)
13:    case 4:
14:            addRight(baseColumns,x,y,subMatrix)
15:    case 5:
16:    return subMatrix
17:    baseColumns.insertInColumn(x,y,subMatrix);
18:    minimo_x = x / size_Mat* size_Mat
19:    return subMatrix
20: end procedure
```

En el Algoritmo 22, set recibe los valores de la posición. Busca si existe en la matriz dispersa el x, y, en la lista doblemente enlazada donde se almacena los valores de las filas y las columnas. Se coloca la submatriz en la cabeza de la lista doblemente enlazada, y se almacenan los valores de máximos y mínimos para tener, los valores que deban ir creciendo hacia la derecha o la izquierda, arriba o abajo. Busca dónde puede agregarlo, primero busca en las filas y vuelve hacer la comparación que existen en los otros algoritmos como casos.

Algorithm 22 Set

```
1: procedure SET(int x,int y, double value)
2:     SparseMatrixDataNode subMatrix
3:     actual, resultante =0.0
4:     if (baseRows=null) then
5:         subMatrix = new SparseMatrixDataNode(size_Mat,0,0)
6:         baseRows = new LinkedListNode(subMatrix,true)
7:         baseColumns = new DoubleLinkedListNode(subMatrix,false)
8:         rows.putAtHead(subMatriz)
9:         columns.putAtHead(subMatriz);
10:         maximo_x = size_Mat
11:        maximo_y =  size_Mat
12:        minimo_x = 0;
13:        minimo_y = 0;
14:        cantidad_x = 1;
15:        cantidad_y = 1;
16:        Switch (dondeIr)
17:        case 0:
18:              return
19:        case 1:
20:              return searchUp(nextNode,x,y)
21:        case 2:
22:        SparseMatrixDataNode subMatriz =
23:        new EliSparseMatrixDataNode(size_Mat,(x /
24:        size_Mat * size_Mat,
25:        *y/size_Mat)* size_Mat
26:        nuevo.setPrevious(nodoSiguiente)
27:        nuevo.setNext(nodoActual)
28:        nextNode.setNext(nuevo)
29:        ActualNode.setPrevious(nuevo)
30:        cantidad_x++
31:        dondeIr = baseColumns.getDataNode().isInColumns(x, y);
32:        switch (dondeIr)
33:        case 3:
34:              insertandoIzquierda(baseColumns,x,y,subMatriz);
35:              break;
36:        case 4:
37:              insertandoDerecha(baseColumns,x,y,subMatriz);
38:              break;
39:        case 5:
40:              baseColumns.insertarEnColumna(x,y,subMatriz);
41:        return subMatriz
42:    end if
43: end procedure
```

5.4.8. Graficar

En esta sección, se muestra parte, de cómo se gráfica y como actualiza las lecturas constantemente, que trabajan con la librería de JPanel. MatrixUpdaterFinal, muestra cómo se actualiza el gráfico en ActualizarGraficar, y run la hebra que permite que grafique a medida que lee. En el Algoritmo 23 se muestra, como se retorna la gráfica, con la librería JPanel. Los valores de las lecturas en la función de profundidad, son almacenados en vectores que ahora se establecerán para las lecturas de los vectores, con las posiciones que se encuentran en la matriz dispersa.

Algorithm 23

```
1: procedure ActualizarGraficar(Graphics g)
2:     vector_odomet= rear_odometrico()
3:     for int i=0;i<vector.size();i++ do
4:         x=vector_.get(i).first_
5:         x=vector_odomet.get(i).second_
6:         g.setColor(Color.decode(color));
7:         g.drawLine(x,y,x,y)
8:     end for
9: end procedure
```

En el Algoritmo 24 se crean las clases principales, para invocar al graficar y crear la matriz dispersa.

Algorithm 24

```
1: procedure MatrixUpdaterfinal
2:     SparseMatrix mat_sparse = new EliSparseMatrix();
3:     JPanel Padre
4:     procedure Matrix_Updater_final((JPanel Padre_)
5:         Padre = Padre_;
6:     end procedure
7: end procedure
```

En el Algoritmo 25 por medio de la librería, se invoca, la clase Graficar con una hebra que dibuja a medida que recibe la data, actualizando las lecturas.

Algorithm 25

```
1: procedure run
2:     Graficar ex = new Graficar();
3:     ex.setVisible(true);
4:     Thread t = new Thread(new Matrix_Updater_final(dpnl));
5:     t.start();
6: end procedure
```

5.4.9. Implementación para funciones de profundidad

En el capítulo 4, se incluye la sección 4.1.3, y las fórmulas de trabajo, para los 3 sensores de profundidad con los que se trabaja. En el algoritmo 26, para los sensores de profundidad se toma en cuenta, la apertura del sensor de profundidad por lo tanto se incluye en la misma función las lecturas de todos los sensores. Para lograr que los sensores generen, la apertura de 30 grados desde que empiece a reconocer tanto, el grado de orientación en la odometría como el obstáculo dentro del radio del sensor, se incluye un for. El valor `deltha_prime`, va incrementando de grado en grado, los ángulos en la programación deben estar en radianes, el ángulo de orientación es `theta_prime`. Para el paso 8, se incluye lo mismo para la apertura del sensor. En cuanto al segundo for, este incluye las distancias desde el punto que existe del robot en conjunto con las fórmulas propuestas de la sección 4.1.3 del sensor de profundidad. En el otro caso, (el paso 16), se almacena lo que realmente se considera un objeto que existe con el valor de 1, que después se actualizará en otra función. Todo, lo que se ha obtenido se almacena en un pair, donde se incluye la posición en x, la posición en y y el valor.

Algorithm 26 Vector<Pair>Lecturas de profundidad

```
1: deltha_prime ← incrementa en un 1 grado
2: theta_prime ← ángulo de orientación
3: hip_rear_komodo ← distancia del punto trasero del robot
4: r1 ← 0
5: x ← posición odométrica en x
6: y ← posición odométrica en y
7: x_rear=x +coseno(180 + deltha_prime) mod(360)* hip_rear_komodo
8: for r1=tetha_prime-15;r<tetha_prime+15:r1=r1+deltha_prime  do
9:     for h_new;h_new<h_rear; h_new=h_new+deltha_h  do
10:        x_rear_total= (x_rear +coseno(180 + r1) mod(360) *(h_new))
11:        y_rear_total= (y_rear +sin(180 + r1) mod(360) *(h_new))
12:        if  x_rear_total <size_Mat_lectures and y_rear_total <size_Mat_lectures
13:            C3[x_rear_total][y_rear] = −1 then
14:        end if
15:    end for
16:    x_rear_total= (x_rear +coseno(180 + r1)) mod(360) *(h_new))
17:    y_rear_total= (y_rear +sin(180 + r1) mod(360) *(h_new))
18:    if  x_rear_total <size_Mat_lectures and y_rear_total <size_Mat_lectures
19:        C3[x_rear_total][y_rear] = 1 then
20:    end if
21: end for
22:
23: for i=0; i<size_Mat_Lectures;i++ do
24:    for j=0; j<size_Mat_Lectures ;j++ do;
25:        if C3[i][j]=1  then
26:            respuesta = new Pair(i,j, 1.0);
27:            vector.add(respuesta);
28:        else if C3[i][j]=-1 then
29:            respuesta = new Pair(i,j, 0.0);
30:            vector.add(respuesta);
31:        end if
32:    end for
33:
34: end for
       return vector
```

Capítulo 6

Pruebas y resultados

En este capítulo se muestran los resultados obtenidos durante el desarrollo de esta tesis, en la primera parte se muestran los resultados de pruebas sobre el sistema sensorial del robot, lo cual es importante para poder tener resultados coherentes en SLAM. La segunda parte muestra los resultados de mapeo, donde, es importante destacar que para cada mapa se muestran dos resultados, el primero son los resultados originales (en los cuales existían inexactitudes agregadas por el sistema sensorial, más no por la estructura que es la propuesta de la tesis) y el segundo son resultados con el sistema sensorial corregido.

6.1. Plan de Experimentos

La organización de los experimentos es la siguiente:

1. Funcionamiento de los sensores en la sección 6.2
2. Pruebas de distancias con inclusión del algoritmo graficar. Pruebas iniciales sin odometría y pruebas finales con inclusión de odometría, tiempo real y perfeccionamiento en mapas en la sección 6.3.
3. Pruebas de sección en mapas, comparaciones con mapas originales en la sección 6.4.
4. Pruebas de memoria, entre MPTE-SLAM y mapas de ocupación en la sección 6.5.
5. Pruebas de MPTE-SLAM para diferente tamaño en los nodos en la sección 6.6.

6.2. Pruebas de los sensores

En el resultado de las siguientes pruebas, se buscó probar los sensores del robot con los que se iba a trabajar. Los encoders de los motores y los sensores de profundidad.

6.2.1. Odometría en encoders

En esta sección se decidió mostrar el comportamiento de los sensores, frente a diferentes tareas. Se inicia con los motores para la odometría.

Se ha definido la medición por medio de las velocidades máximas, intermedias y medias lineales alcanzadas por el robot Komodo. Esto ha sido obtenido por medio del tópico cmd_{vel} que nos retorna la velocidad en tiempo real con dos valores por medio de la conexión wifi de datos: *linear*, *angular*. Incluyen pruebas de los encoders en los motores del robot y la odometría. Considerando la velocidad lineal:

- Lento: 1m/s - 3 m/s
- Medio: 4m/s - 6 m/s
- Rápido: 7m/s - 10m/s

1. Obedeciendo a las funciones que se le incluyeron, como evitar obstáculos y seguir recorriendo un área determinada, se tomó la medición desde un punto fijo establecido para un punto determinado, basándose en la odometría que como ya se explicó antes, se basa en movimientos de motores servo.
2. Comparando el valor de los errores sistemáticos calculados por la odometría como valor real obtenido, fue creado el error lineal.
3. Comparando el valor de orientación estimado por la odometría, con el valor medido fue obtenido el error angular.

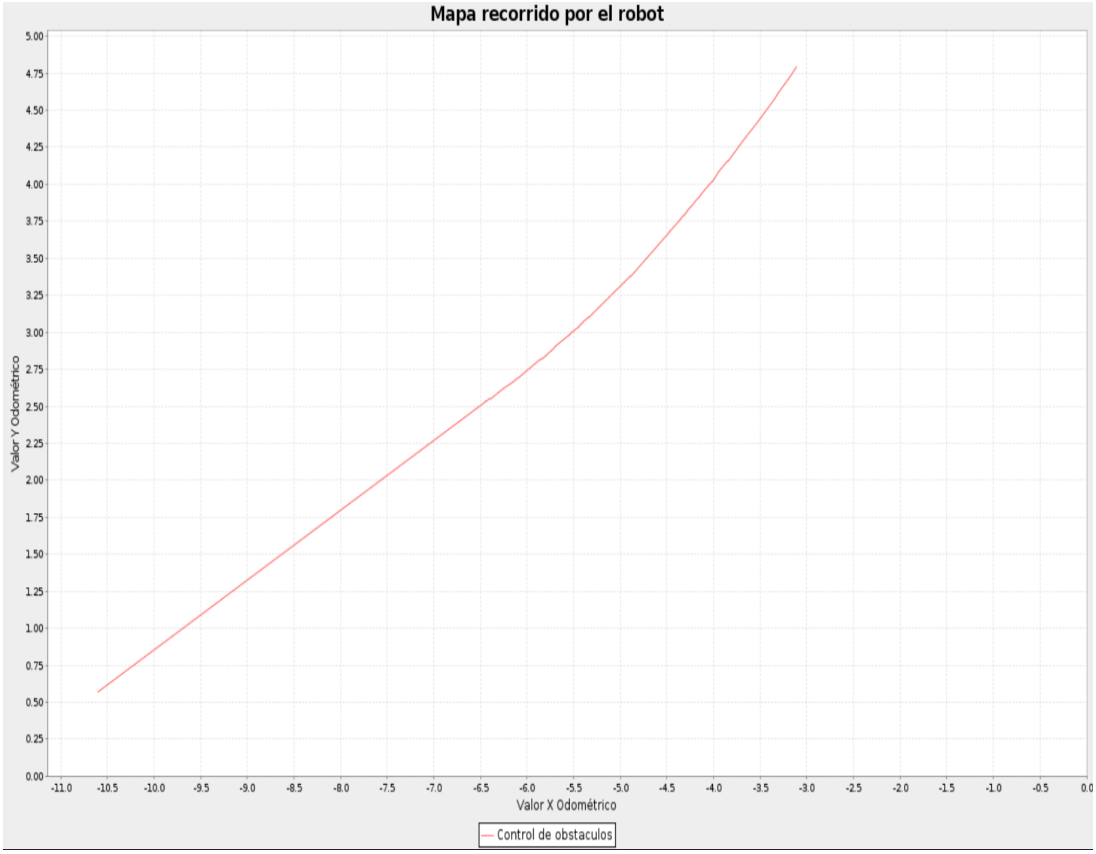


Figura 6.1: Retorno del mapa en línea recta (Fuente: Creación propia)

En la Figura 6.1 el robot avanza en línea recta, que representa el mejor caso aun así se muestra como los errores no sistemáticos afectan la trayectoria del robot dependiendo del suelo.

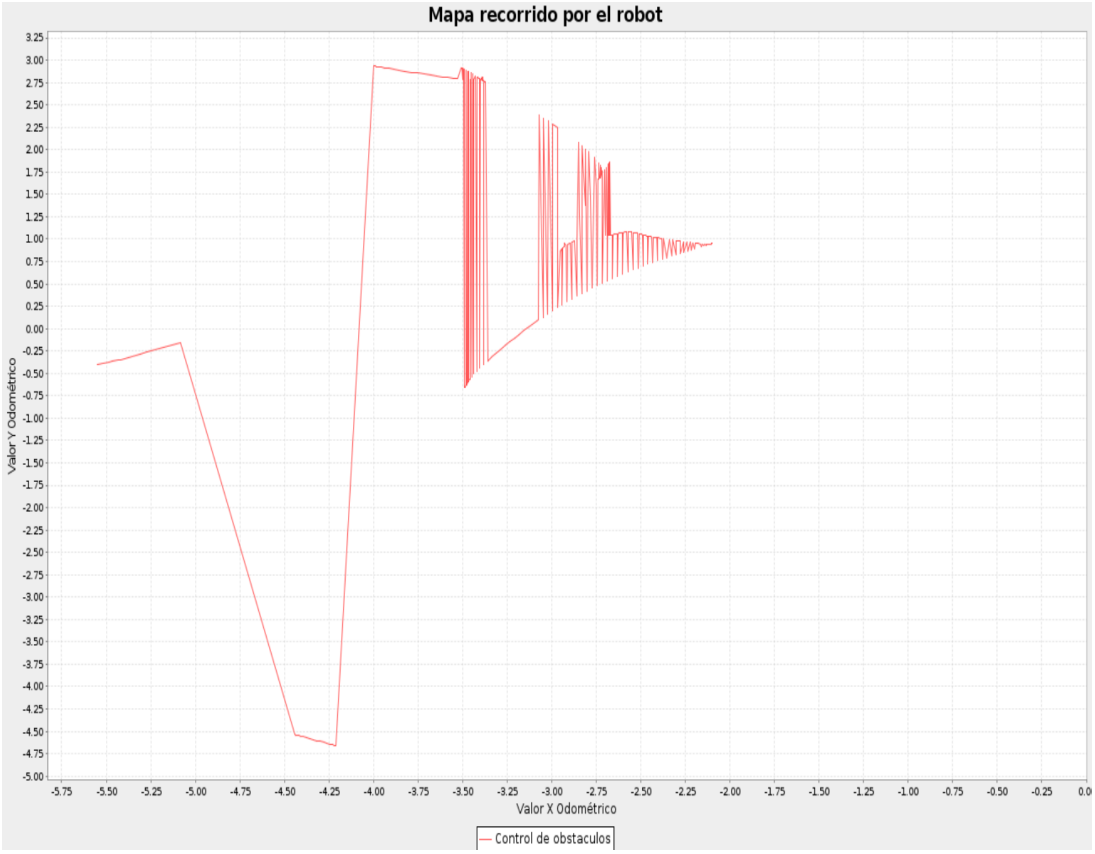


Figura 6.2: Retorno del mapa en recorrido largo (Fuente: Creación propia)

En la Figura 6.2 se muestra como el robot recorre en línea recta y logra realizar la tarea en forma adecuada. El problema al generar bastantes recorridos o giros se encuentra en la generación de una elipse de error a pesar de que el gráfico que muestra una semejanza con el camino. Lo interesante es que es capaz aunque de una forma torpe generar los giros que se simularon en la ruta, este resultado no incluye realmente la posición del robot pero si inicia en 0,0.

6.2.2. Sensores de profundidad

Esta sección forma los resultados, en la capacidad de distinguir obstáculos a distancias del sensor. No se tomaron velocidades pero si distancias, las cuales no son muy grandes debido a que pertenecen a los obstáculos que pueden encontrarse cerca del robot y si la función es capaz de colocarlos de manera acertada en su radio de visión. Todo esto, permite comprobar que no solo su funcionamiento es correcto, sino que podemos calcular la distancia de visión del robot por medio de la medición de objetos. Mientras recorre un ambiente con una velocidad promedio de 4 a 5 m/s y estos obstáculos se alejan del robot, todo en tiempo real.

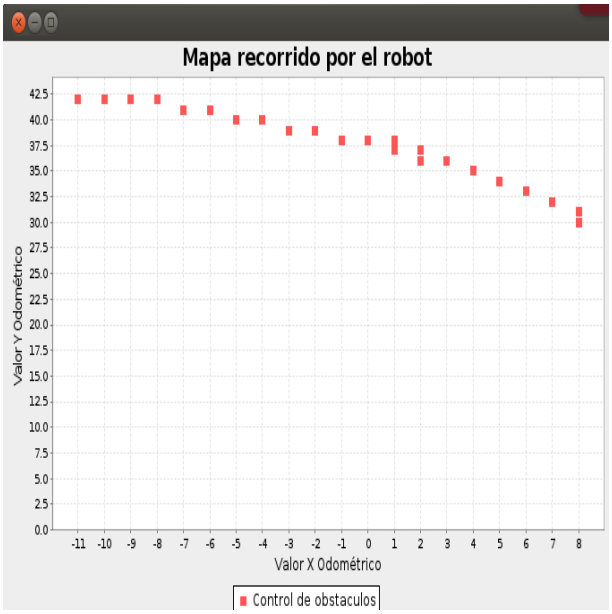


Figura 6.3: Retorno de imagen en la distancia de 50 cm del sensor de profundidad (Fuente: Creación propia)

En esta parte de los resultados, el arco de profundidad solo ha sido probado de forma unitaria, sin incluir el ángulo de orientación. En la Figura 6.3, el arco representa la apertura del sensor y el punto inicial es 0 en x y 0 en y.

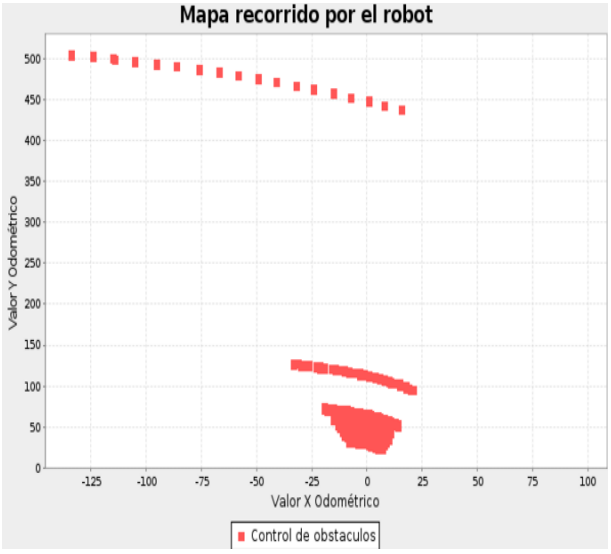
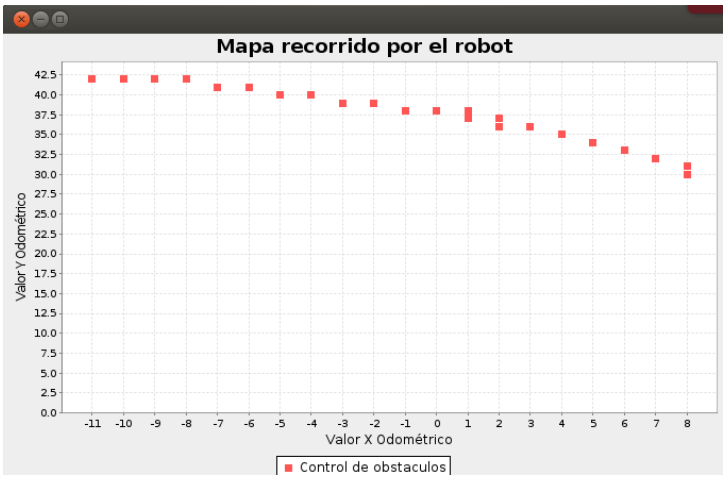
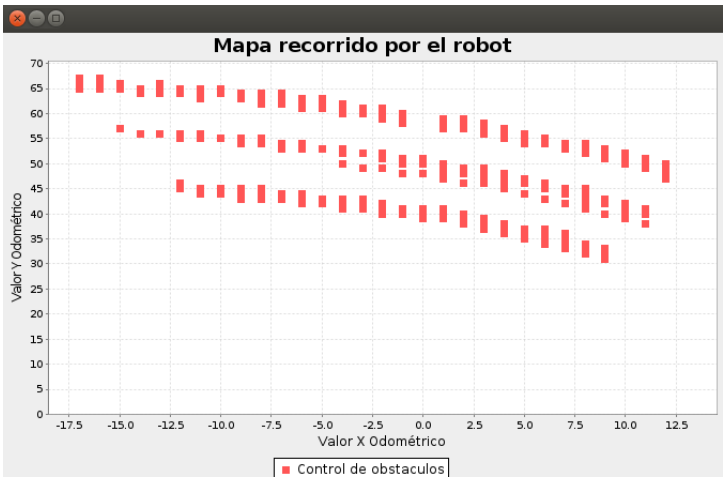


Figura 6.4: Retorno de imagen en la distancia de 3 cm del sensor de profundidad (Fuente: Creación propia)

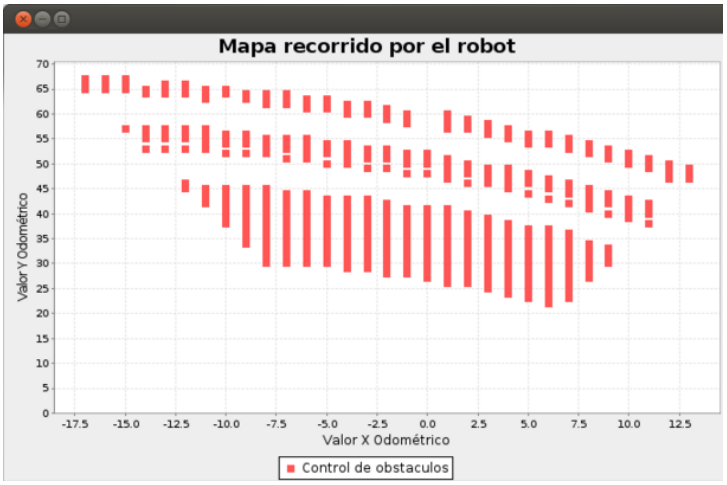
En la figura 6.4 la gráfica, muestra cómo el objeto es acercado y el dibujo presenta la actualización.



(a)

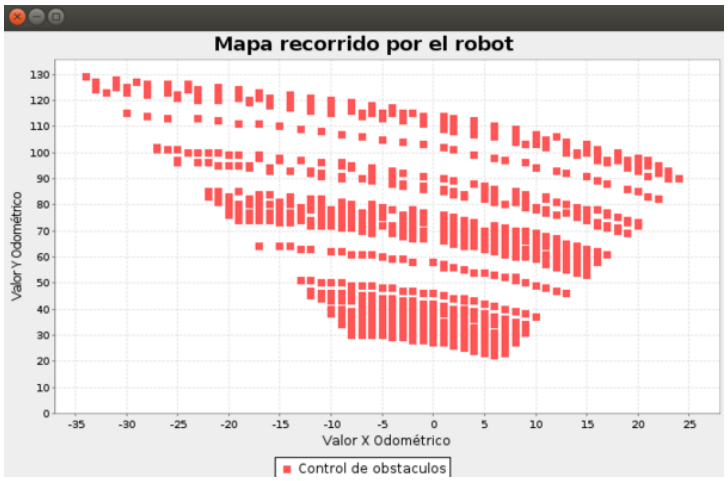


(b)

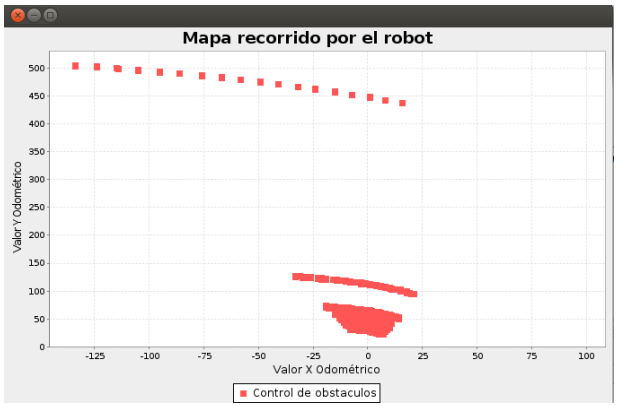


(c)

Figura 6.5: Actualización en computadora de sonar a:
(a) 30 cm
(b) 20 cm
(c) 15 cm (Fuente: Creación propia)



(a)



(b)

Figura 6.6: Actualización del sensor a:
(a) 10 cm (Fuente: Creación propia)
(b) 3 cm (Fuente: Creación propia)

En la Figura 6.5 se muestra, la pantalla de la computadora a medida que se busca el retorno de los datos obtenidos por, los sensores de profundidad. En (a) empiezan a acercarse los puntos, en (b) ya toman forma como conjunto, para que en (c) ya estén todos los puntos centrados porque el objeto ha sido aproximado, cada vez más. Se realiza todo esto, para que se grafique el arco de profundidad del sensor de profundidad según el Algoritmo 26. Como refieren las imágenes, cada una es una distancia más cercana al sensor de profundidad trasero que en tiempo real, permite el retorno de las distancias por la computadora del robot, a la computadora que trata los datos. En la Figura 6.6 (a) la actualización a una distancia de 10 cm, (b) muestra al sensor trasero a 3 cm, para graficar las actualizaciones en tiempo real. Tanto en la Figura 6.5 y en 6.6 el sensor de profundidad, no inicia en 0 para **x** y 0 para **y**. Debido a que aún no se tiene, la orientación de los sensores sino una prueba, la curva se muestra en 0,0 debido a que no se ha incluido la posición en 2D.

En el caso de obtener la precisión del sensor de profundidad. Se realizó la medición para cada caso 100 veces. A las distancias definidas en 3 cm, 10 cm, 15 cm, 20 cm y 30 cm. Esto permite probar que la medición a menor distancia, genera mayor imprecisión, mientras la medición a 10 cm. En el caso de la precisión del sensor de profundidad este es de +/- 1 cm. La Figura 6.7 permite observar que la desviación de mayor valor, el cual es de 0.75 cm siendo este el peor caso y que cumple que el error es de 1 cm (estando estacionado).

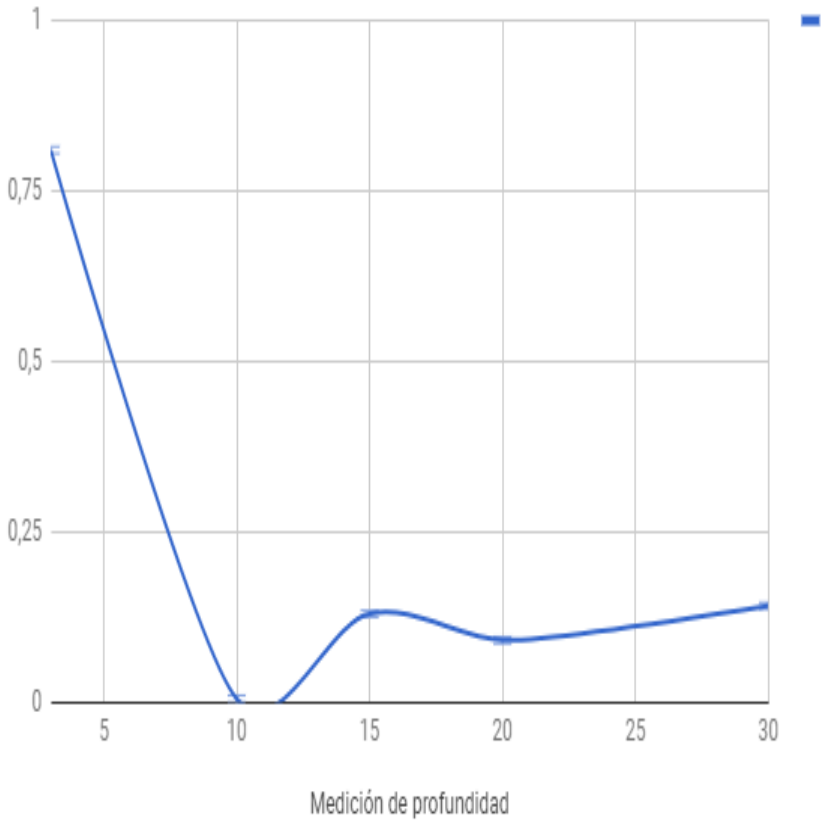
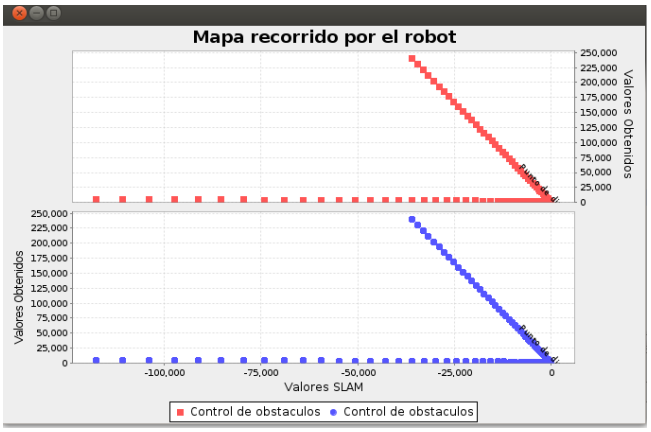


Figura 6.7: Error de medición entre distancia y sensor de profundidad (Fuente: Creación propia)

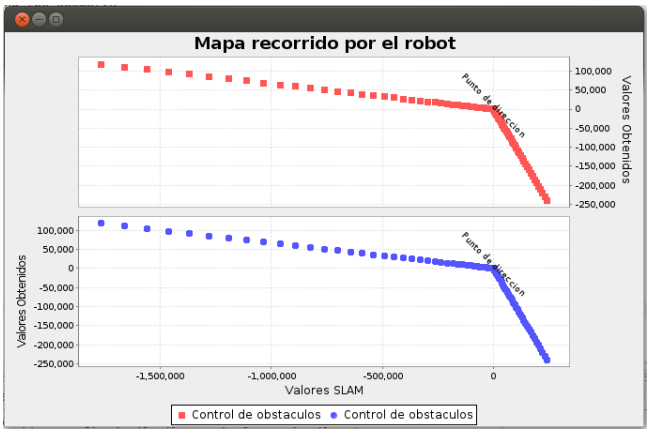
6.2.3. Ángulo de orientación por encoders

Los siguientes resultados representan la función que muestra la dirección a la cual el robot se dirige (puede ser tanto hacia la derecha o izquierda). Se comprobó, que la odometría en cuanto a giros, genera a mayor velocidad, gráficos imprecisos. Para los resultados en cuanto a la orientación se tomaron las velocidades iniciales ya propuestas y se hizo un total de 20 pruebas para cada caso. La orientación no presentó ningún error debido a que el ángulo con el que se retorna la orientación, proviene de la odometría y se modifica de una manera óptima en los resultados obtenidos de los datos.

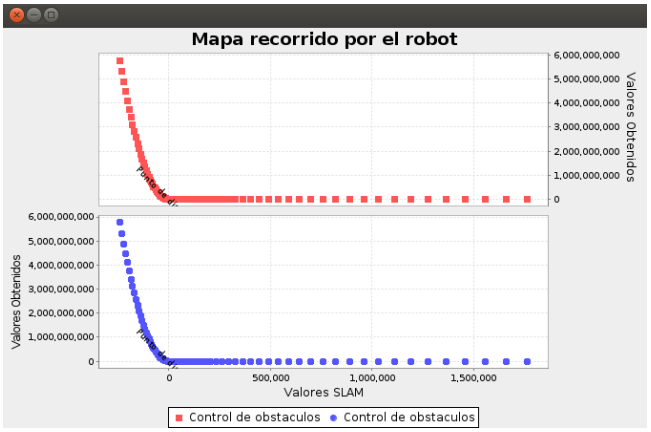
En la Figura 6.8, se aprecia las pruebas en dibujos, de los giros presentados por los 3 sensores, según la posición hacia dónde se dirigían. Esta prueba, se realizó para probar que el ángulo de orientación del robot estaba realizando mediciones de forma adecuada. La Figura (a) el robot gira hacia abajo y a la derecha, en (b) gira hacia arriba y en (c) gira hacia abajo izquierda cuyo punto de intersección en todas las imágenes, se muestra como unos puntos.



(a)



(b)



(c)

Figura 6.8: Giro del robot:
(a) Hacia abajo por derecha
(b) Hacia arriba a la derecha
(c) Hacia abajo, izquierda (Fuente: Creación propia)

6.3. Resultados de reconocimiento del ambiente

6.3.1. Pruebas de 3 sensores con robot detenido

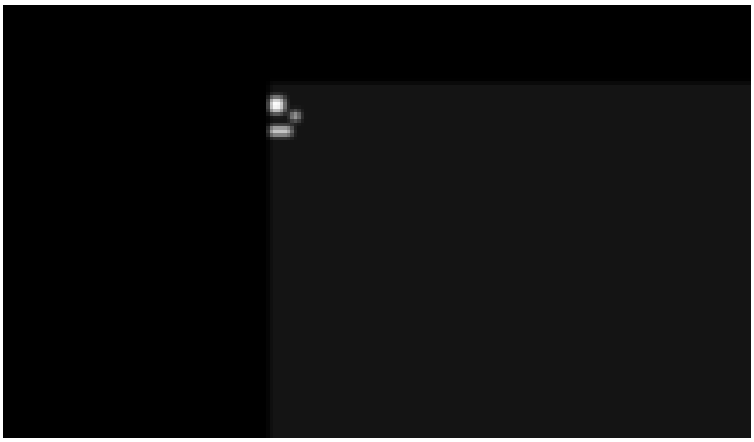
En los resultados presentados en este capítulo, se muestra, la mejora en cuanto a la visualización de la imagen final. Por medio de un ajuste en las fórmulas que se presentaron en el Capítulo 4. A su vez para discretizar mejor el mundo se hizo una división entre el valor de lo que valdría cada celda en el mundo real. En las siguientes imágenes se muestra que debido a que los resultados anteriores de los mapas eran algo pequeños, se debía modificar ciertas fórmulas y lograr que las imágenes mejorarán por medio de las pruebas de la captura de objetos (con el robot detenido). Se decide tomar estas pruebas, para mostrar que el robot detenido tiene lecturas correctas del ambiente, al recorrer una pequeña distancia se genera almacenamiento en la estructura.

(a) (b)

Figura 6.9: Pequeña estructura para prueba de sonares
(a) Robot dentro de la pequeña estructura, pero tomada de lado. (b) Robot dentro de la pequeña estructura, pero tomada al frente (Fuente: Creación propia).

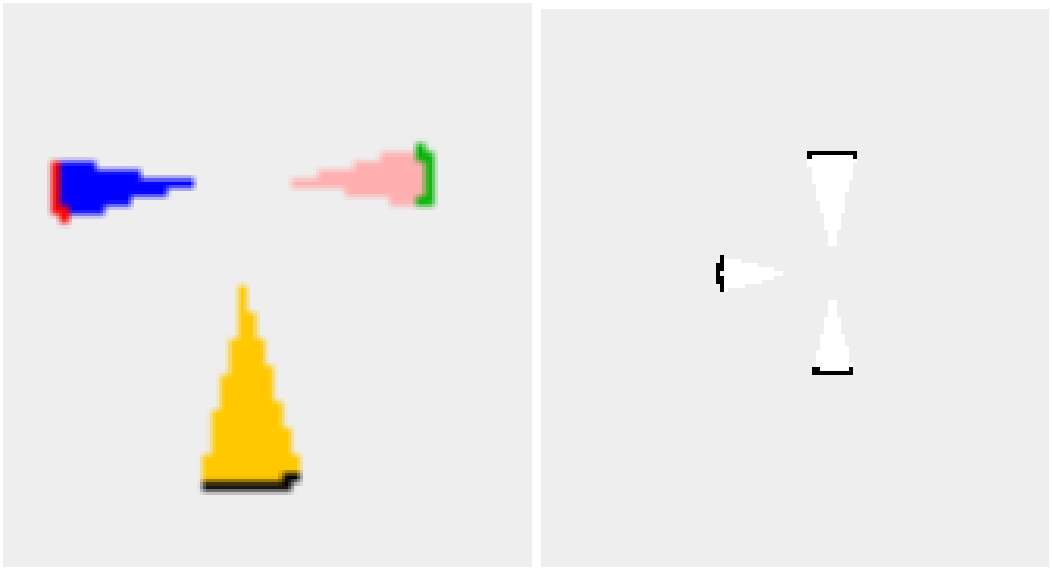
Como se observa en la Figura 6.9, en este caso se realizó la prueba de obtención de lecturas, con los tres sensores y el robot estacionado. Estos resultados, nos permiten observar, si el robot realmente funciona con lo establecido para la orientación y si el cálculo es correcto.

En la Figura 6.10, se muestra el desarrollo y la transición de todas las lecturas realizadas, con los mismos obstáculos. En (a) la primera lectura de los tres sensores con el robot estacionado, muestra que si se puede diferenciar tres lecturas pero no del todo correctas. La medición por parte de algunos sensores, son más grandes o más pequeños. En (b) la distancia, según la fórmula (4.3) que aún no había sido del todo modificada pero si, la fórmula de orientación permite generar lecturas mejoradas y mostrar que realmente, si se distingue entre los 3 sensores pero aun generando error en el retorno de las vistas según la fórmula de (4.3). En cuanto a (c) para mostrar, si realmente los tres sensores eran correctos, se decidió modificar los colores, para que cada uno muestre cuales son las lecturas (independientes de cada sensor). Una vez corregidas las lecturas por los cuaterniones y la conversión a Euler, se decide colocar los colores blancos para lectura, negro para el objeto que realmente se encuentre allí, por la función en la fórmula (4.25). En cuanto a (d) la imagen al final, se definió de la siguiente manera, mientras el objeto tenga un color más oscuro por la fórmula de actualización (4.25) y la fórmula (4.26).



(a)

(b)



(c) (d)

Figura 6.10: Lecturas de tres sensores de profundidad:
(a) Primer resultado de las lecturas del robot
(b) Segundo resultado generado
(c) Tercer resultado del mismo espacio
(d) Gráfico final (Fuente: Creación propia)

(a)

Figura 6.11: Lecturas en diferentes distancias y posiciones con odometría (Fuente: Creación propia)

En la Figura 6.11 se incluye, la odometría como punto referencial del movimiento del robot, el cual va a ser definido como el camino realizado. Como se puede observar en (a) el sensor, tiene lecturas en ambientes iguales, pero en la segunda fotografía, el sensor rear está sintiendo un obstáculo más lejos que los sensores derecha e izquierda. En cuanto a la tercera medición, el sensor derecho, está actualizando sus lecturas y también no se encuentra en movimiento, por esta razón el centro sólo se genera como un punto medio entre todos los sensores.

6.3.2. Pasillo pequeño

La siguiente prueba, debe mostrar lecturas en pequeñas estructuras, y que se almacene una habitación muy pequeña. Esta prueba, se realiza luego de las lecturas de los tres sensores estacionados porque el robot ahora incluye movimiento. Las pruebas se componen, de dos etapas, a continuación:

1. **Primera Etapa**: En esta etapa, se muestran las pruebas en una etapa inicial, antes de corregir la fórmula (4.4) hasta la fórmula (4.18).

(a)

Figura 6.12: Inicio de las lecturas en pasillo pequeño con agujero en el fondo (Fuente: Creación propia)

(a)

Figura 6.13: Retorno de las lecturas del robot, girando a la izquierda del pasillo (Fuente: Creación propia)

(a)

Figura 6.14: Fisuras en la pared, que genera cercanía y lejanía en el sensor (Fuente: Creación propia)

(a)

Figura 6.15: Imagen del pequeño pasillo, al fondo se logra capturar la forma de las sillas (Fuente: Creación propia)

(a)

Figura 6.16: Toma de las lecturas del robot en una distancia mayor, aún girando en el pasillo (Fuente: Creación propia)

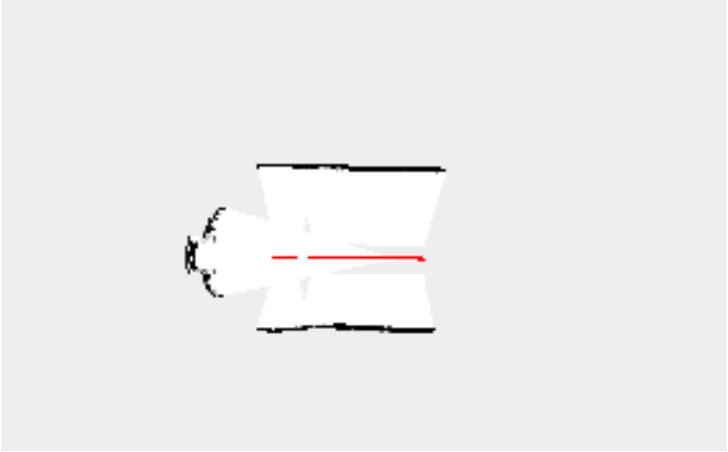
En esta prueba el robot avanzó y retrocedió constantemente, la Figura 6.12, muestra las diferentes características por donde se hicieron las pruebas, en el pasillo. En la Figura 6.13 (a) debe retornar el fondo, con diferentes formas. En Figura 6.14 (a) el robot ha recorrido una mayor distancia pero las paredes de la prueba no son del todo lisas. Como se muestra en Figura 6.15 existen pequeñas aberturas que permiten leer las patas de las sillas, debido a que el sensor de profundidad, tiene un rango de 6 metros y en Figura 6.16 (a) se encuentra el mejor resultado obtenido, las distancias entre la profundidad aún no son los adecuados.

2. **Segunda etapa:** Los resultados finales para el pequeño pasillo, incluyen ángulos de Euler, odometría en la generación de caminos.

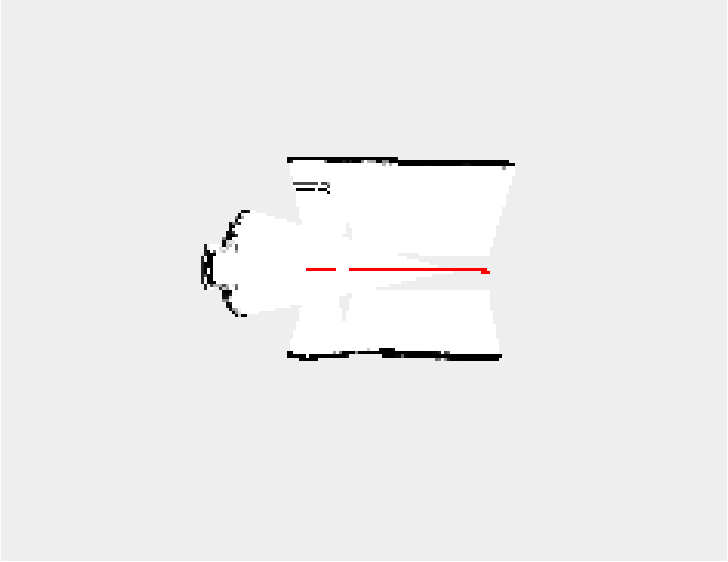
(a)

Figura 6.17: Mapa de pequeño pasillo con odometría (Fuente: Creación propia)

En la Figura 6.17, en el pequeño pasillo, solo se realizaron dos pasadas por el ambiente, la línea roja es el camino realizado, el cual solo fue hacia adelante. Este pasillo pequeño puede ser recorrido por el robot, al fondo se tienen tablas de madera.



(a)



(b)

Figura 6.18: Robot dentro de la pequeña estructura, pero tomada al frente
(a) Prueba I
(b) Prueba II (Fuente: Creación propia)

Como resultado del mapa en cuanto al pequeño pasillo, lo que se ha logrado es una mejora significativa en cuanto a la lectura del ambiente. Por otro lado, existen unas puntas grises en la parte izquierda e inferior como resultado de la apertura definida del sensor. Existe una dislocación inicial que se muestra al ingresar al pasillo, que es retornado, por la odometría como un punto más grueso debido al dislocamiento, tanto en la Figura 6.18 (a) como en (b) se presenta este dislocamiento. En (a) no se presenta a la derecha, una perturbación negra que puede representar la madera a la derecha en el ambiente real de Figura 6.17, como se muestra en (b). La pequeña estructura de prueba, no cuenta con una prueba de memoria por lo pequeño del ambiente.

6.3.3. Pasillo largo

- **Primera parte, donde se muestra el pasillo largo con puerta abierta pero sin modificaciones en las fórmulas**

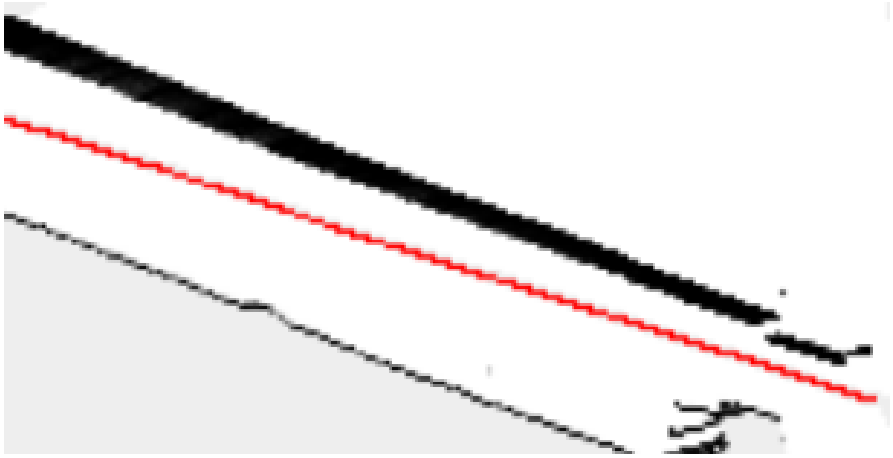
(a)

(b)

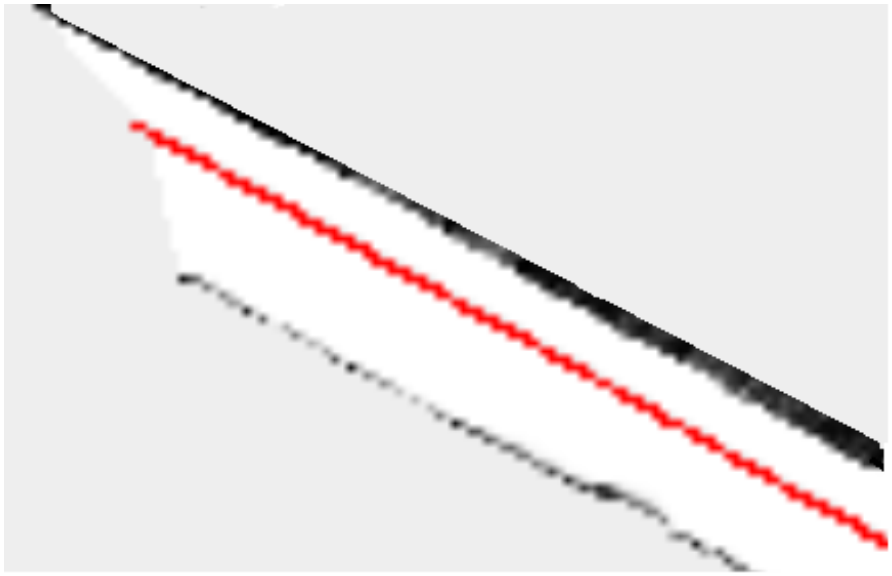
Figura 6.19: Mapa de pasillo largo con odometría, avanzando y retrocediendo (a) Mapa retornado por el robot (b) Mapa real (Fuente: Creación propia)

En la Figura 6.19 (a) se muestra la gráfica, realizada por el robot por medio de sus tres sensores. A pesar de que aún no se agregaron, las modificaciones de cuaterniones para ángulos de Euler. En (b) la imagen del pasillo permite notar, que una puerta está abierta.

- **Segunda parte, se incluye odometría, tiempo real y un mayor tramo para las lecturas del pasillo**



(a)



(b)

Figura 6.20: Mapa de pasillo en una única pasada, con odometría en rojo
(a) Resultado I
(b) Resultado II (Fuente: Creación propia)

En la figura 6.20, se vuelve a tomar dos pruebas dentro del pasillo, está prueba tiene algunas características particulares. Los sensores de profundidad tienen un retorno diferente, dependiendo del material de las paredes pero generando la misma curva en la parte derecha inferior de la imagen mostrada en (a). En la mayoría de casos esto se debe a la odometría como se muestra en (b) y el problema no solo, es el dislocamiento de las ruedas sino también de los sensores de profundidad. La Figura 6.21, muestra la cantidad de memoria utilizada de 6.4 GB por el mapa de pasillo en un nodo de tamaño 100 x 100.

(a)

Figura 6.21: Memoria utilizada en pasillo largo (Fuente: Creación propia)

6.3.4. Prueba de patio

- **Primera parte de resultados de patio**
  Los resultados obtenidos retornaban una imagen similar al mundo que tenía alrededor, en conjuntos con los pasillos y los objetos del mundo. En teoría estaba finalizada la parte de la obtención del mundo, pero en la práctica ocurrió un problema. Cuando el robot tenía giros amplios este fallaba retornando líneas superpuestas por donde debía graficar. En el momento que se pasaba los valores de odometría tanto los de posición como los de orientación; el ángulo de θ de la odometría, la cual brinda la orientación no estaba siendo el adecuado para trabajar.

(a)

Figura 6.22: Pasillo donde se tomaron las pruebas (Fuente: Creación propia)

(a)

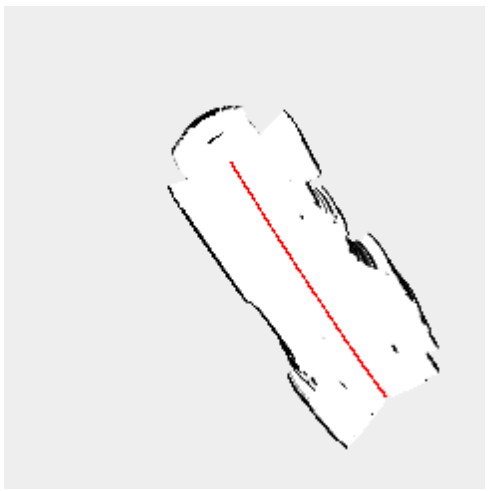
Figura 6.23: Mapa de pasillo en una única pasada (Fuente: Creación propia)

La Figura 6.22 muestra, el ambiente en donde se realizaron las pruebas en esta primera parte. En la Figura 6.23 se puede observar que simplemente por la generación de errores en odometría el robot avanza, generando errores. En la búsqueda de una solución se concluye que era la lectura del ángulo de orientación, teniendo que modificar la fórmula.

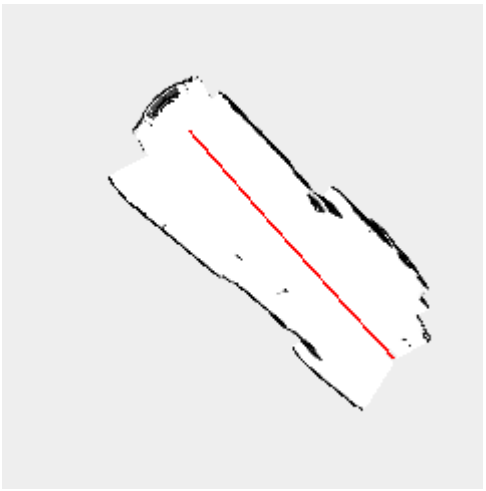
- **Segunda parte de resultados de patio, con resultados mejorados**

En la toma de resultados con las fórmulas mejoradas y la inclusión de odometría se puede observar una mejora notoria. La Figura 6.24, tiene tres tomas diferentes de la misma estructura. Para (a) se muestra que reconoce a la izquierda un obstáculo, las gradas, de la Figura 6.22 a lado de la planta. En la Figura 6.25 se muestra la cantidad de memoria de 6.56 GB en el nodo 100 x 100, utilizada en el patio.

(a)

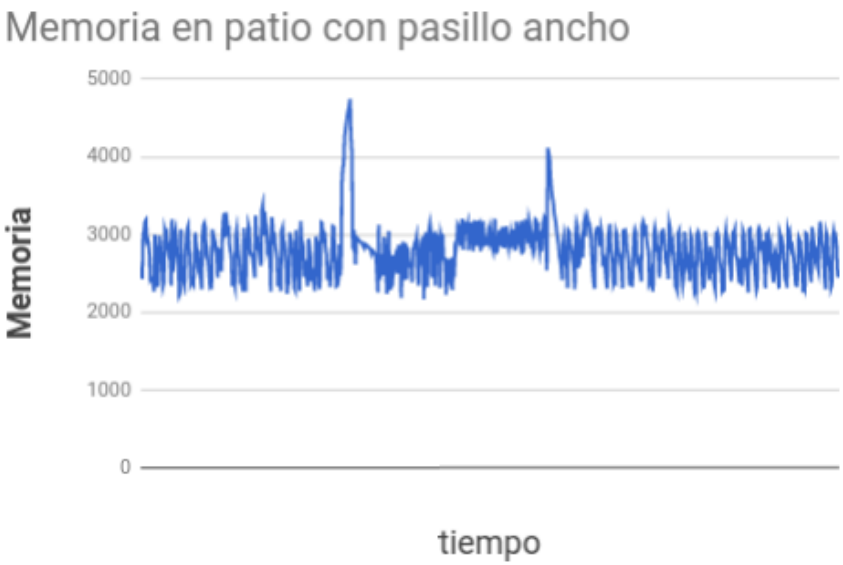


(b)



(c)

Figura 6.24: Pruebas de mapas finales:
(a) Primera generación de mapa de patio en una única pasada, con odometría en rojo
(b) Segunda generación de mapa de patio en una única pasada, con odometría en rojo
(c) Tercera generación de mapa de patio en una única pasada, con odometría en rojo
(Fuente: Creación propia)



(a)

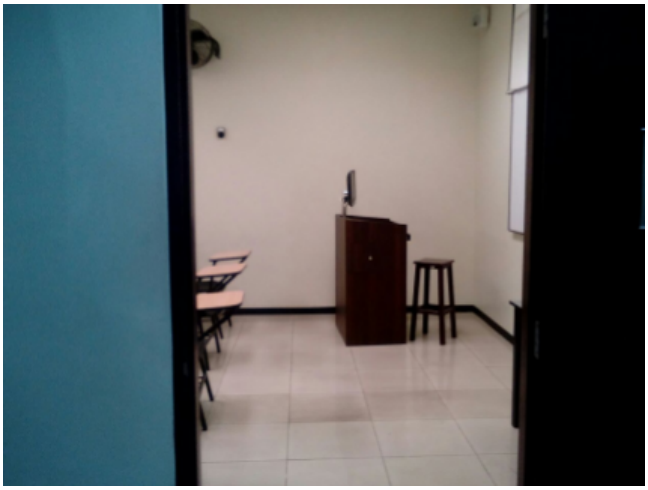
Figura 6.25: Memoria utilizada en patio (Fuente: Creación propia)

6.3.5. Prueba de unión de estructuras

- **Primera parte de pruebas de unión de estructuras**
  Para la obtención de resultados en cuanto a la construcción de mapas más grandes, la estructura y el tamaño que debe ser almacenado, se tomó el pasillo y clases por donde debía entrar y almacenar la información el robot. En esta primera evaluación, las habitaciones pequeñas no se graficaban de manera adecuada.

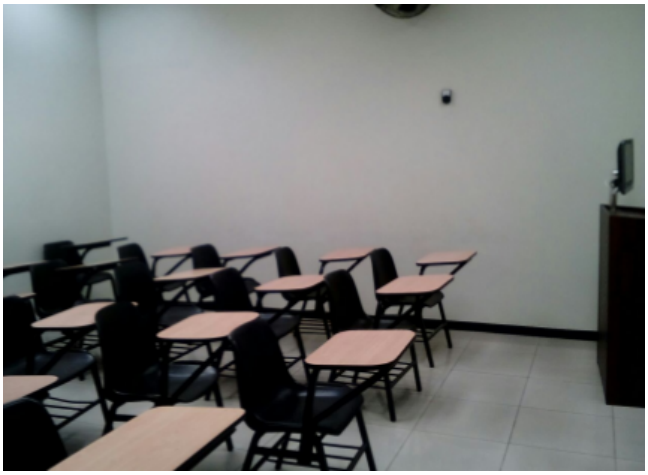
En la Figura 6.26 se muestra en (a) el pasillo que recorrió el robot, a su vez en (b) se muestra una clase, el robot sólo recorrió hasta el podio, sin ingresar entre las carpetas.

(a)



(b)

Figura 6.26: Recorrido de robot:
(a) Pasillo de ambiente
(b) Salón luego de recorrer ambiente (Fuente: Creación propia)



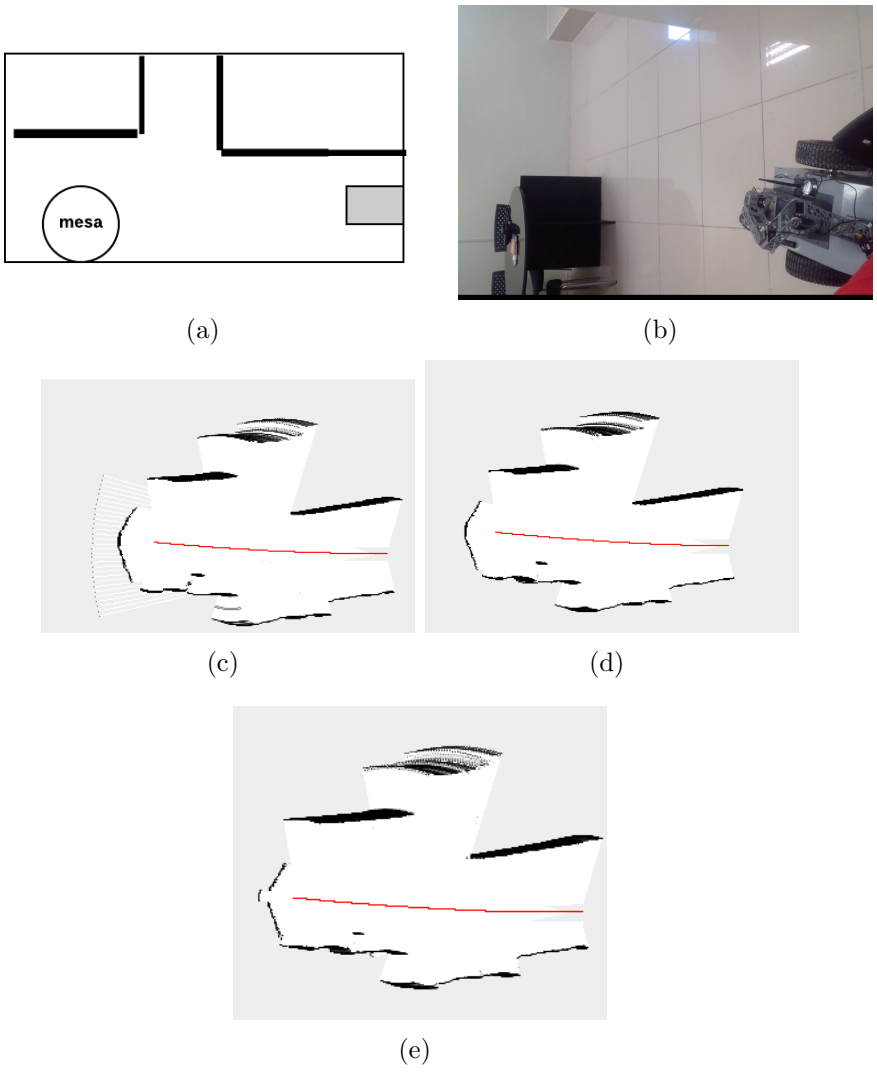
(a)

(b)

Figura 6.27: Ambiente recorrido por el robot:
(a) Aula en la parte interna
(b) Aula retornada por el robot y pasillo (Fuente: Creación propia)

En la Figura 6.27 se incluyó, la fotografía interna del aula, mientras que en (b) el resultado no muestra, un resultado similar a lo mostrado como ambiente de prueba.

- **Segunda parte de pruebas de unión de estructuras**
  En la segunda parte de las pruebas, se busca unir los caminos para incluir giros de la odometría como se mostrará más adelante en el documento. La inclusión del manejo por medio del comando *rqt*, permite manipular el robot desde la computadora, toma velocidad al ser lanzado por lo tanto genera un error más grande en la odometría por el dislocamiento.



(a) (b)

(c) (d)

(e)

Figura 6.28: Toma de pasillo corto y ancho con pequeño cuarto:
(a) Primera imagen de estructura en el plano
(b) Segunda imagen de estructura en el mundo real
(c) Tercera imagen resultante de toma I
(d) Cuarta imagen resultante de toma II
(e) Quinta imagen resultante de toma III (Fuente: Creación propia)

La Figura 6.28 se realizó en una pequeña habitación debido a que los resultados anteriores, presentaban errores notorios en parte, por la división del espacio, que no retornaba la habitación o un pequeño espacio similar al plano real. A su vez. Por ejemplo, (a) contiene el dibujo de la habitación pequeña (b) tiene la imagen real (c), (d) y (e) retornan los resultados por el robot. Se generaron mapas similares. La generación de un mapa, más grande en comparación de la habitación es el resultado de la división de la posición tanto en x y y por un valor menor. En la Figura 6.29 se muestra la cantidad de 2.7 GB en memoria en la pequeña habitación, para nodos de 100 x 100.

(a)

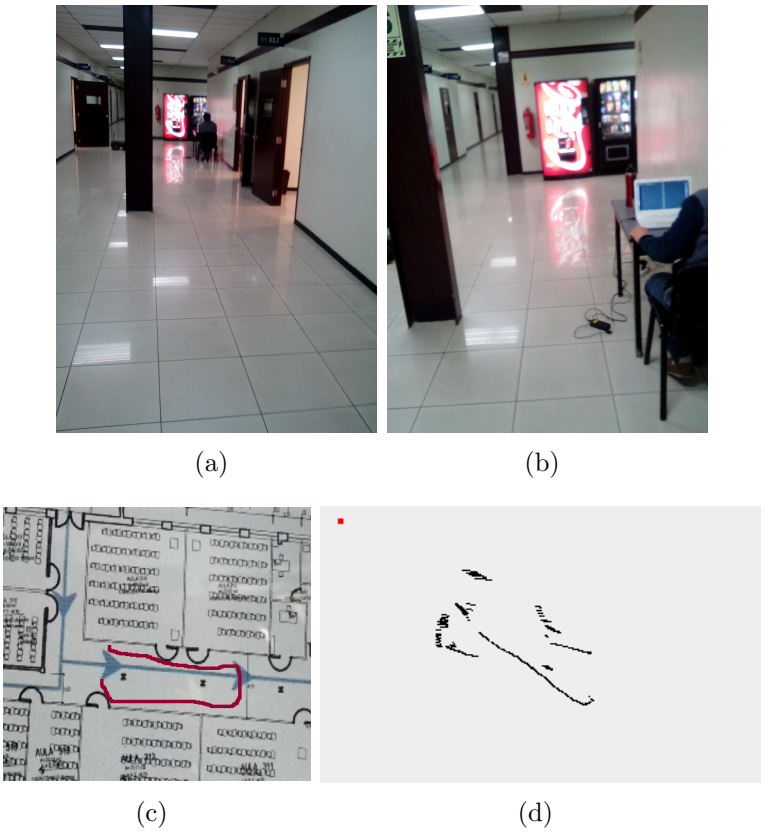
Figura 6.29: Memoria utilizada en pequeña habitación (Fuente: Creación propia)

6.3.6. Pruebas comparativas de mapas

Este tipo de pruebas, representan diferentes ambientes, que han sido recorridos de nuevo, por el robot con la fórmula mejorada y la inclusión de la odometría en la segunda parte de pruebas, la primera parte aún incluye los primeros resultados.

- **Primera parte de las pruebas en pasillo con columnas**

  Las siguientes pruebas realizadas, se llevaron a cabo en un pasillo con dos columnas de metal. Para poder observar si realmente se les considera un obstáculo, en el mapa realizado por el robot.



(a) (b)

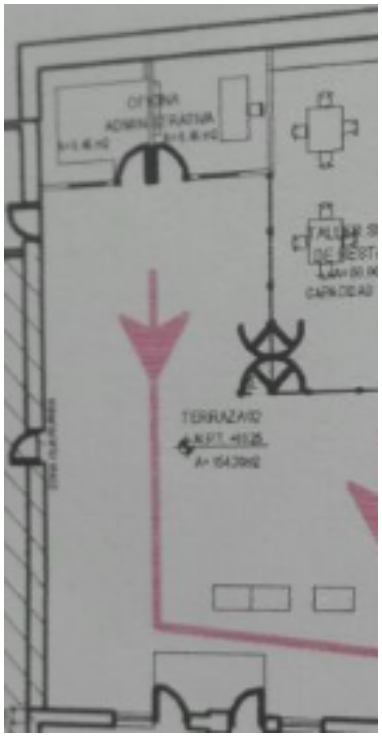
(c) (d)

Figura 6.30: Estructura de pasillo con columnas:
(a) Fotografía real del ambiente con columnas
(b) Ambiente de pasillo cercano a columnas
(c) Resultado en plano del camino realizado
(d) Resultado en plano del camino realizado por el robot (Fuente: Creación propia)

En la Figura 6.30, se muestra una primera prueba, en un ambiente con columnas en el recorrido, para mostrar de qué manera lo rastrean los sensores del robot. En (a) se nota la distancia que mapeara el robot. En (b) al finalizando el recorrido y el grosor de la columna (c) ha sido obtenida del plano y el camino trazado. En (d) se muestra,el resultado final que el robot ha realizado.

- **Segunda parte de pruebas con ambientes largos**
  Las siguientes pruebas realizadas, son la corrección para de la primera parte de las pruebas, en donde al tener perturbaciones la fórmula, generaba un error al colocar en la profundidad (la verdadera posición del muro). A la izquierda se muestra que el material del muro, es fibra de vidrio, material que afecta las lecturas del sensor pero que coloca correctamente la profundidad, se incluye el ruido de las personas.

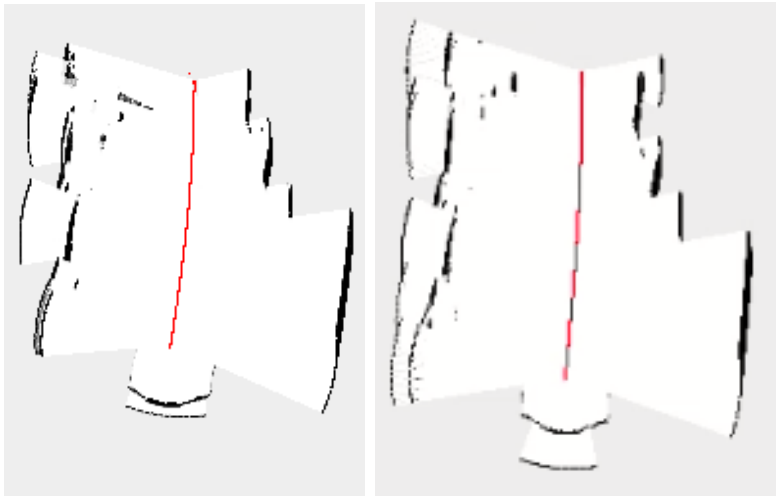
(a)



(b)

Figura 6.31: Recorrido de patio:
(a) Mapa del ambiente recorrido, con personas y plantas
(b) Plano del ambiente recorrido (Fuente: Creación propia)

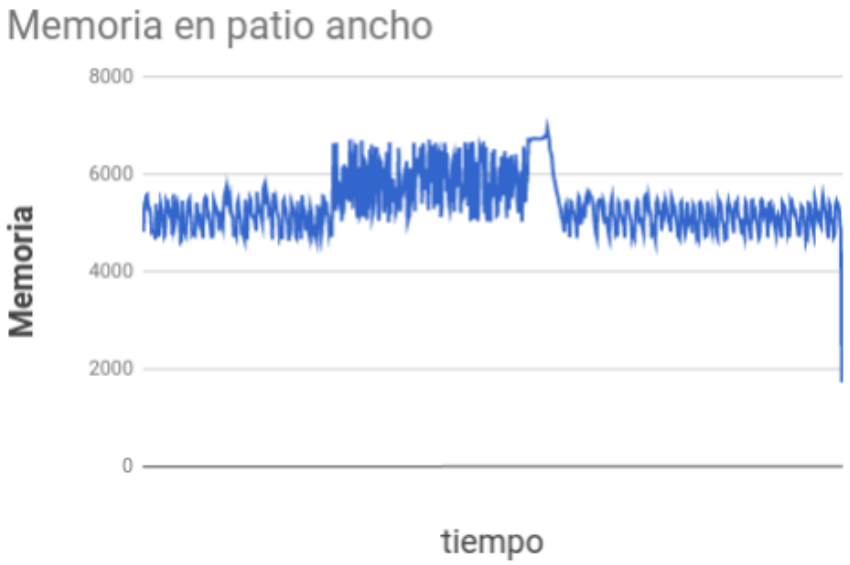
La Figura 6.31, tiene diferentes objetos dentro, las mesas, las plantas y las personas que recorren el ambiente que recorre el robot. En (a) la muestra real del ambiente, y (b) el plano recorrido por el robot.



(a) (b)

Figura 6.32: Resultado de mapa de patio:
(a) Resultado I
(b) Resultado II (Fuente: Creación propia)

En la Figura 6.32, se muestra una segunda prueba, en un ambiente como patio a la derecha. En (a) se muestra la distancia que mapea el robot al lado derecho con el mapa. En (b) al finalizar se muestra el fondo con la grada, en ambas. Con el ambiente, de la Figura 6.31 a) y el plano b). El material en a) a la izquierda es fibra de vidrio, a diferencia de concreto a la derecha. Para la Figura 6.33 se muestra la cantidad de memoria de 6.9 GB, utilizada en nodos de 100 x 100.



(a)

Figura 6.33: Memoria utilizada en patio ancho (Fuente: Creación propia)

6.3.7. Comparación de celdas por cada ambiente

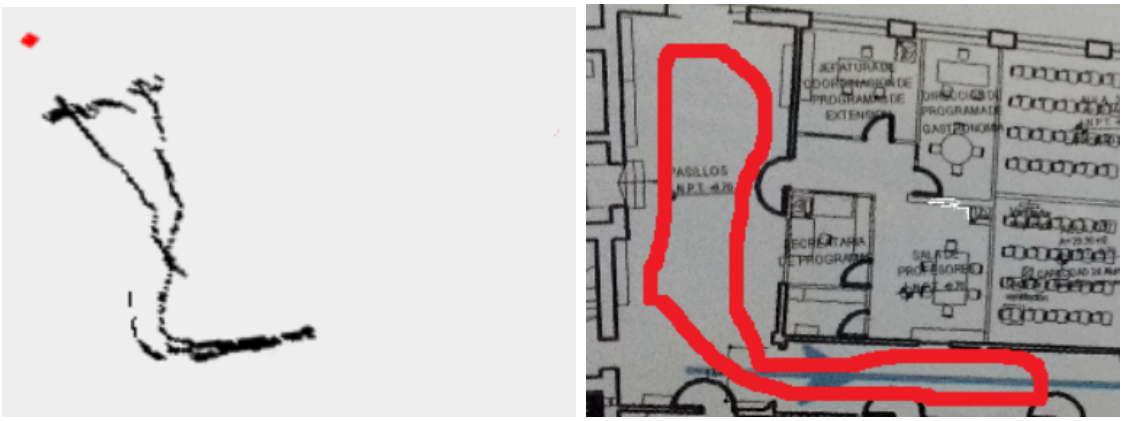
- **Primera parte de las pruebas, patio y pasillo por derecha**

Las siguientes pruebas realizadas, se llevaron a cabo en un pasillo y un patio ancho. Como se ha mencionado en otros resultados, los giros generan error de dislocamiento que no están exentos en los resultados.

(a) (b)

Figura 6.34: Ambientes recorridos:
(a) Pasillo ancho con gradas a la izquierda
(b) Pasillo largo (Fuente: Creación propia)

En la Figura 6.34 se muestra el lugar donde se tomaron las pruebas, (a) muestra el pasillo largo y ancho y (b) muestra el pasillo.

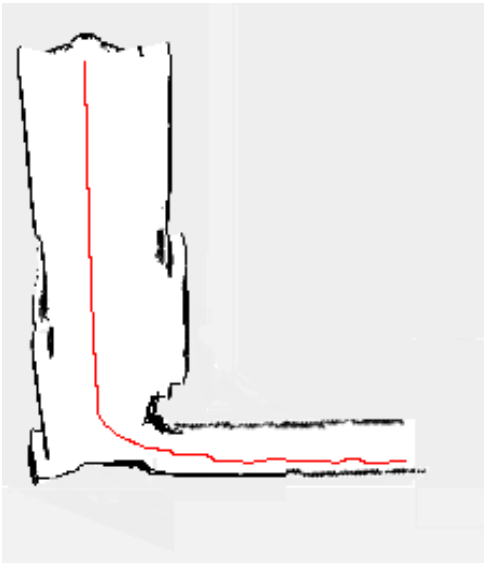


(a) (b)

Figura 6.35: Mapa resultante vs mapa con camino en plano:
(a) Mapa resultante de 6.34
(b) Generación de camino en el plano (Fuente: Creación propia)

Los siguientes resultados en la Figura 6.35, pertenecen al mapeo del robot, el cual tiene similitud con el mundo real. Pero en el giro, pierde el ángulo de orientación de la odometría.

- **Segunda parte de las pruebas, patio y pasillo por derecha**
  En esta segunda parte, de pruebas de mapas comparativos, se incluye el siguiente resultado que muestra una mejora, tanto en la gráfica como al momento de dar giros con odometría. La prueba realizada es una de las primeras tomadas, el robot se encontraba con mejor calidad en los giros. En la Figura 6.36 se muestra las curvas realizadas por el robot, pero con generación del camino y espacio. En la Figura 6.37 se muestra el uso de memoria de 6.5 GB utilizada en un mapa L.



(a)

Figura 6.36: Generación de camino de dos ambientes (Fuente: Creación propia)

(a)

Figura 6.37: Memoria utilizada en mapa en L (Fuente: Creación propia)

6.4. Estadísticas por medio del cálculo de sección:

Primera parte de comparación de mapas con el plano real
Como previamente se presentaron, ejemplos de mapas de mayor tamaño, se necesitaban resultados en diferentes ambientes. Para establecer si realmente el robot obtiene el mundo que recorre. En la sección de este trabajo se ha hecho una distancia de 0,5 metros. En la matriz estática se define cada cuadrado del mismo tamaño al hacer la división entre los valores obtenidos. Por ejemplo si retorna en 2D, x= 0.45 sería entre la distancia 0.5, un valor de 0,9 que redondeando nos da 1. Eso significa que el robot está en la posición x=1. Dentro de la matrix en x, sería 1, ahora se realizó el mismo proceso para y. Solo que ya estamos incluyendo la orientación y la posición por medio de la odometría con modificaciones realizadas. En el trabajo final se puede mostrar que mientras la distancia sea menor, el valor del gráfico va aumentando. En este caso queríamos comparar el mundo real con las imágenes y por ese motivo se tomaron los valores, de 0.5 de medición.

En la Figura 6.38 se realiza, la medición de cuanto equivale cada baldosa del piso, comparado con el valor establecido de 0.5 metros en el programa como se muestra en el gráfico. Partiendo desde este punto, se cuadrícula de forma comparativa, mostrando las diferencias y las similitudes. Por otro lado antes de iniciar con el muestreo y las comparaciones entre mapas, debemos recordar que el mundo que ha recorrido el robot ha sido cuadriculado, por medio de las baldosas. Si ocupa un total de 5 cuadraditos debemos multiplicarle el 0.5 para hacer la conversión a metros. Teniendo un total de 5 cuadraditos debemos multiplicarle por 0.5, que es la medición por baldosa, tendríamos la multiplicación de 2.5 metros. La velocidad constante de 7 metros por segundo.

(a)

(b)

Figura 6.38: Medición de baldosa en su totalidad:
(a) Mostrando medición de baldosa
(b) Mostrando valor en la medición (Fuente: Creación propia)

6.4.0.1. Comparación de cuadrícula de mapas resultantes en primera parte

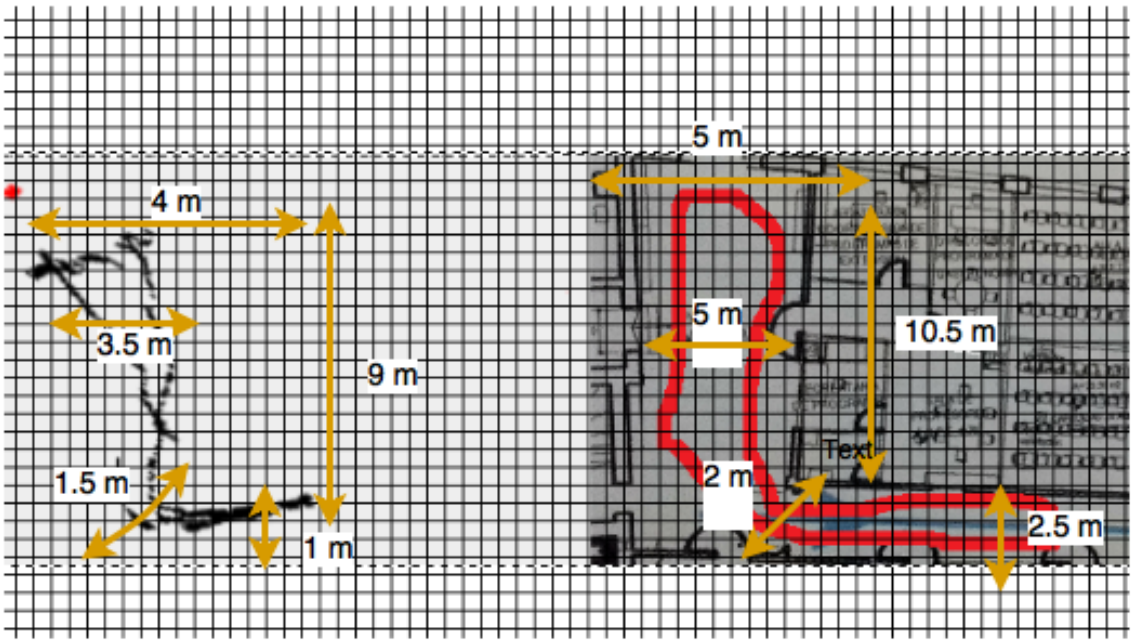
En esta sección se determinan diferentes resultados de los mapas realizados por el robot. Los primeros son en casos de curvas, el robot produce un error más grande, con un error de precisión de +- (0.5) metros en el caso de líneas rectas y de +- (1 a 1.5) metros en curvas (por la resta entre valores). Los valores fueron obtenidos entre la comparación de los mapas versus el mapa retornado presentes en las Figuras 6.28, 6.30, 6.35. La resta de las diferencias, concluye lo expuesto en este trabajo, a estos resultados agregamos la dificultad que presenta el robot al hacer giros y también su tamaño en los ambientes. Para la comparación de resultados finales del error porcentual presentado antes, se utilizó la fórmula 5.3 donde se obtiene el valor de error. El valor bibliográfico sería el valor del mapa real, restándole el valor experimental sobre el valor bibliográfico de nuevo multiplicándolo por cien, para obtener un porcentaje.

$$\frac{ValorBibliografico - ValorExperimental}{ValorBibliografico} * 100 \tag{6.1}$$

La fórmula 6.1, forma parte de la búsqueda de similitud entre los mapas realizados como primera parte de pruebas. Para encontrar la similitud en las pruebas obtenidas con los planos reales.

6.4.1. Patio y pasillo por derecha

En los resultados para patio y pasillo por derecha, se tomó la Fórmula 6.1 para dividir el plano y el mapa obtenido.



(a)

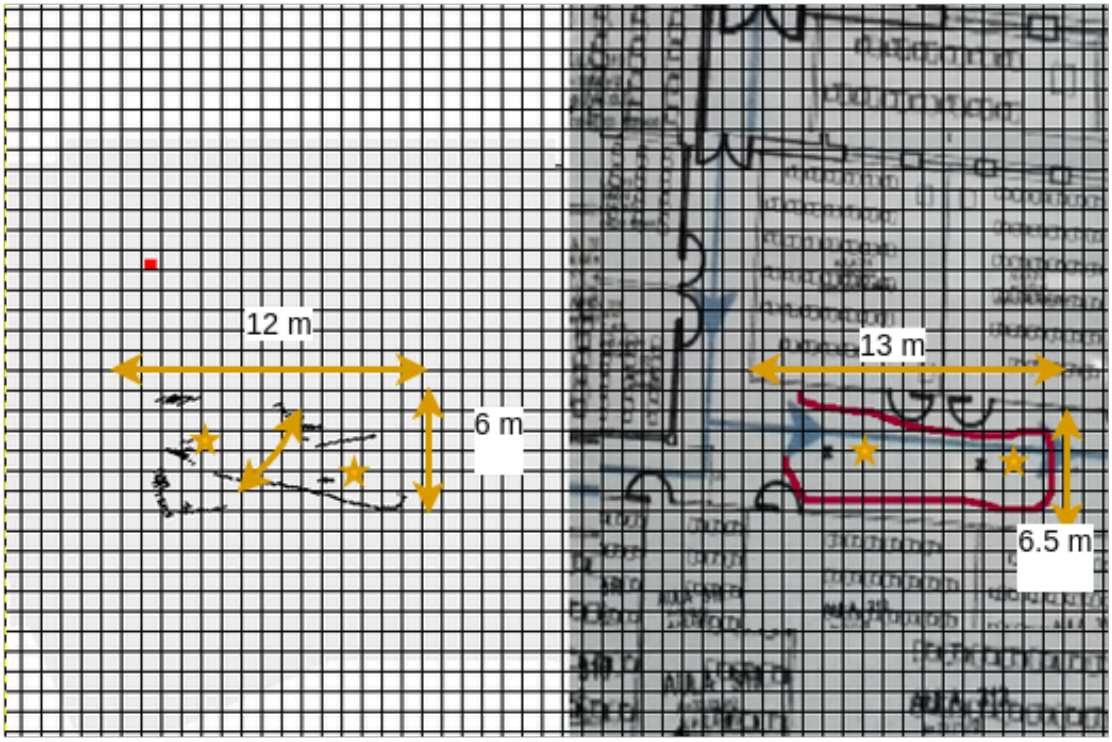
Figura 6.39: Patio y pasillo por derecha I (Fuente: Creación propia)

Cuadro 6.1: Mapa de patio y pasillo derecho I

Comparación del total de celdas por cada ambiente I			
	Mapa real	Mapa obtenido	Error de Precisión
Muro arriba	10 cuadraditos x 0.5 m= 5m	8 x 0.5 = 4 m	20 %
Altura	21 cuadraditos x 0.5 = 10.5 m	18 cuadraditos x 0.5 = 9 m	14.28 %
Ancho de curva	4 cuadraditos x 0.5 = 2 m	3 cuadraditos x 0.5 = 1.5 m	25 %
Grosor pasillo	5 cuadraditos x 0.5 = 2.5 m	2 x 0.5 = 1 m	60 %

En la Figura 6.39 las comparaciones a través de los cuadraditos del plano. Se puede verificar las distancias por el conteo de cuadraditos con el mapa real. En el Cuadro 6.1 se define la comparación entre el mapa real y el mapa obtenido en el mapa de patio y pasillo derecho. Se definió las partes importantes según la estructura y el recorrido final del robot.

6.4.2. Ambiente con columnas en el medio



(a)

Figura 6.40: Comparación de cuadrícula de ambiente con columnas en el medio III (Fuente: Creación propia)

Cuadro 6.2: Mapa de patio y pasillo izquierdo II

Comparación del total de celdas por cada ambiente II			
	Mapa real	Map obtenido	Error de Precisión
Muro arriba	10 cuadraditos x 0.5 m= 5m	8 x 0.5 = 4 m	20 %
Largo de ambiente	21 cuadraditos x 0.5 = 10.5 m	18 cuadraditos x 0.5 = 4 m	14.29 %
Grosor de máquinas	4 cuadraditos x 0.5 = 2 m	3 cuadraditos x 0.5 = 1.5 m	25 %
Grosor pasillo	5 cuadraditos x 0.5 = 2.5 m	2 x 0.5 = 1 m	60 %

.

En la Figura 6.40 las comparaciones a través de los cuadraditos del mapa, con la otra imagen. Se puede verificar las distancias por el conteo de cuadraditos con el mapa real. En el Cuadro 6.2 se define la comparación entre el mapa real y el mapa obtenido en el mapa de patio y pasillo derecho. Se definió las partes importantes según la estructura y el recorrido final del robot.

Segunda parte de comparación de mapas con el plano real:

En mapeamiento para robótica, realmente no existe una comparación entre los mapas y el plano. Si el resultado es similar a la estructura con odometría y detección de obstáculos, este resultado es suficiente por mostrar la distancia, la proporción de profundidad y el camino del robot. En la primera parte de resultados se realizó la comparación porque realmente existían errores, a pesar de que había similitud, porcentual. A continuación se presentan las comparaciones entre los resultados, que han sido ya explicados con el plano que han recorrido. En la Figura 6.41 se incluyen los mapas mejorados (a la izquierda el plano y retorno del robot a la derecha). Al considerarlos mejorados, nos referimos a la colocación de lo que va percibiendo el robot del mundo con la profundidad adecuada y el tramo recorrido por la odometría. En (a) se comprueban las características de profundidad con el mapa y el generado por el robot. En (b) las aperturas del espacio también es percibido por el robot, en la orientación adecuada. Para la (c) el robot muestra que en su recorrido, pierde estabilidad por la odometría (en rojo), pero genera un mapa preciso del ambiente que recorrió.

(a) (b)

(c)

Figura 6.41: Comparación de mapas y resultados:
(a) Mapa de patio con abertura
(b) Patio largo
(c) Patio largo con pasillo (Fuente: Creación propia)

6.5. Pruebas de memoria

En esta sección ya hemos comprobado que se genera un mapa del espacio. Existen errores que no permiten que la generación de un mapa sea preciso pero se ha logrado obtener la percepción de los 3 sensores del robot en el mundo. Las especificaciones de la computadora son las siguientes:

- Memoria: 8 GB
- Procesador: Intel Core i5
- Graficos: Gallium 0.4 on NVC1
- Tipo sistema operativo: 64 Bit
- Disco: 160.3 GB

Como la propuesta en este trabajo es probar, que la nueva estructura versus las utilizadas en la robótica realmente cumplen con el objetivo de que la matriz dispersa en este trabajo, como los autores refieren según lo investigado y presentado en el Capítulo 2, en representación con grafos, tiene el mismo comportamiento que un grafo (que representa a un mapa topológico). Por este motivo, falta la comparación entre la matriz estática y la estructura MPTE-SLAM para comprobar por otro lado, que esta estructura en cuanto al uso de memoria es mejor.

```
while true
do
        free -m >> memory.txt
        sleep 0.25
done
```

Para las pruebas de memoria, se trabajó con el siguiente script, en Python. Para realizar la siguiente prueba se llevó a cabo lo siguiente:

1. Se ejecuta en una terminal memory.sh durante 15 segundos.
2. Eso crea un archivo de texto en donde se van a guardar los valores del uso de RAM cada 0.25 segundos.
3. Interrumpimos el programa y esperamos 15 segundos.
4. Apagamos la PC, y reiniciamos el paso 1; pero con la otra estructura, si es MPTE-SLAM o la matriz estática.

En los siguientes resultados de comportamiento de memoria, la Figura 6.42 muestra el resultado, en el mapeo de un ambiente similar, referenciado en la Figura 6.28. Esta prueba inicial se realizó para comprobar que se cumplía el ahorro en memoria. La matriz de ocupación con un tamaño de 1000 x 1000 y nodos de 10 x 10 de tamaño en MPTE-SLAM. En (a) el crecimiento de la memoria empezó a incrementarse rápido y paso de

forma rápida a bordear 3.1 gigabytes y seguía creciendo hasta que se detuvo el script. Mientras que en (b) se muestra el comportamiento de la estructura presentada, MPTE-SLAM. Donde el comportamiento es estable hasta que es detenido bordeando luego de actualizar hasta 2.7 gigabytes, pero se estableció en 2.5 gigabytes de nuevo hasta que se detuvo.

(a)

(b)

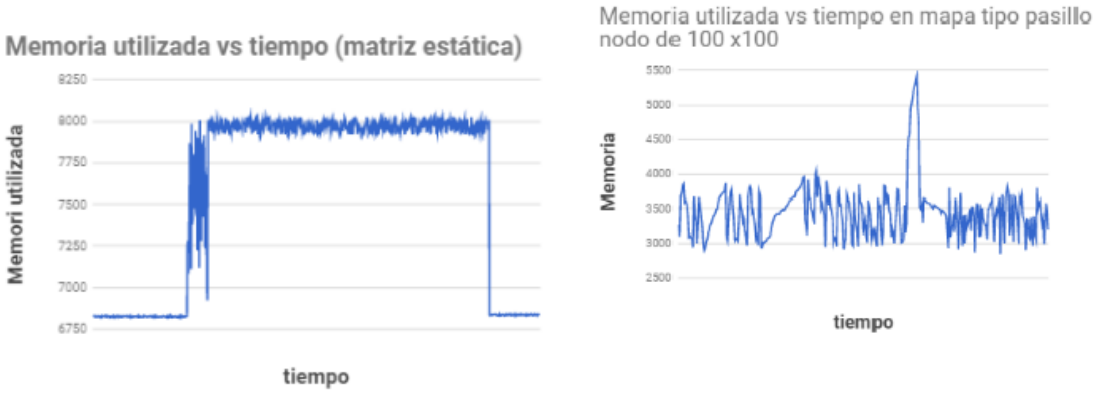
Figura 6.42: Uso de memoria en pequeña habitación:
a) Matriz estática (Fuente: Creación propia)
b) Matriz MPTE (Fuente: Creación propia)

En la Figura 6.43 se muestra la memoria aplicada en una matriz de ocupación versus MPTE-SLAM con nodos de tamaño 100x100. En (a) mostramos la memoria utilizada, que crece constantemente, haciendo lento el retorno de mapas. El tamaño (en este caso 3500 x 3500) las 8 GB de memoria son alcanzadas. En (b) MPTE-SLAM trabaja con 6.5 GB en el pico más alto.
Para la Figura 6.44 la estructura recorrida es un pasillo, (a) representa la memoria utilizada, haciendo un proceso lento de retorno del mapa. La cantidad de 5 GB es utilizada, en (b) los picos dependen de las actualizaciones que se realizan.

En el siguiente Cuadro 6.3 se hicieron las comparaciones según los mapas que muestran la memoria utilizada en cada uno, tenemos dos tipos de mapas en pasillo que es un caso bueno para ambas estructuras, porque permiten separar memoria para las matrices de ocupación y en las MPTE-SLAM el ahorro de memoria. Para el mapa en forma de L,

(a) (b)

Figura 6.43: Uso de memoria en mapa en L:
a) Matriz estática (Fuente: Creación propia)
b) Matriz MPTE (Fuente: Creación propia)



(a) (b)

Figura 6.44: Uso de memoria en mapa de pasillo:
a) Matriz estática (Fuente: Creación propia)
b) Matriz MPTE (Fuente: Creación propia)

Cuadro 6.3: Comparación de uso de memoria entre matriz de ocupación y MPTE-SLAM

Tamaño de nodos	Memoria en mapa en L	Memoria en mapa pasillo largo
10	4.9 GB	4.7 GB
20	5.0 GB	5.1 GB
30	5.7 GB	5.7 GB
100	6.5 GB 0	6.4 GB

también se han realizado estas pruebas en diferentes tamaños de nodos. Todos los mapas han sido comparados previamente en las Figuras 6.42, 6.43 y 6.44.

Al finalizar las pruebas de memoria en diferentes mapas, podemos concluir que la memoria utilizada en la estructura MPTE-SLAM, realmente utiliza menor cantidad de memoria, facilitando el retorno de la imagen sin que se vuelva lento el retorno del mapa.

6.6. MPTE-SLAM con nodos de diferentes tamaños

Para los siguientes resultados se tomó de ejemplo el mapa del patio largo, la estructura que tiene el mejor resultado para las matrices de ocupación. Por carecer de aberturas, porque no hay pasillos y genera una matriz total por bordes. Se tomaron diferentes tamaños de nodos matrices para MPTE-SLAM. Se inició, con un nodo de tamaño 10 por 10, la segunda prueba con un nodo de tamaño 20 x 20, la tercera de 30 x 30 y la última con una matriz de 100 x 100 como nodo. Esta estructura en particular, presenta un tamaño diferente en la parte superior de mayor grosor mientras que en la parte inferior, es más estrecho. Entonces para la división del mapa tenemos que el patio mide 5 m de ancho en el mundo real y 11 m, de largo. Por la división del espacio obtenemos que el mapa a graficar, mide 2750 m de largo y 1250 m de ancho. Cada espacio del patio, se ha dividido por losetas de 0.5 m, cada una, pero en el código se ha decidido definir el espacio de división a 0.002 para generar un ambiente más grande. El tiempo ha sido el mismo, para todos los recorridos, lo que se modifica es el tamaño en los nodos de matrices, (como se ha explicado). En la Figura 6.45 se muestra el ambiente que se ha recorrido con el robot para las siguientes pruebas.

(a)

Figura 6.45: Mapa de patio grande (Fuente: Creación propia)

- **10 x 10**

(a)

Figura 6.46: Uso de memoria para matriz de 10 x 10 (Fuente: Creación propia).

En la Figura 6.46 se muestra en (a) cuánta memoria se utilizó, para lograr el mapa con nodos de tamaño 10 x 10. Al dividir el espacio de 1250 de ancho se necesita 125 matrices de 10 x 10, 2750 entre 10 es 275 de ancho.

- **20 x 20**

(a)

Figura 6.47: Uso de memoria para matriz de 20 x 20 (Fuente: Creación propia)

En la Figura 6.47 se muestra cuánta memoria se utilizó, para lograr el mapa con nodos de tamaño 20 x 20. Al dividir el espacio de 1250 de ancho, que se necesita 63 matrices de 20 x 20, y 2750 entre 20 es de 138 de largo.

- **30 x 30**

(a)

Figura 6.48: Uso de memoria para matriz de 30 x 30 (Fuente: Creación propia).

En la Figura 6.48 Se muestra cuánta memoria se utilizó, para lograr el mapa con nodos de tamaño 30 x 30. Al dividir el espacio de 1250 de ancho se necesita 42 matrices de 30 x 30 y 2750 entre 30 es de 92 de ancho.

- **100 x 100**

(a)

Figura 6.49: Uso de memoria para matriz de 100 x 100 (Fuente: Creación propia)

Cuadro 6.4: Memoria empleada en MPTE-SLAM

Tamaño	Memoria utilizada	Memoria Libre	Número de submatrices por sección
10x 10	4.300 GB	3.600 GB	23
20 x 20	4.650 GB	3.200GB	11
30 x 30	5.200 GB	2.200 GB	8
100 x 100	7.500 GB	1000 MB	3

En la Figura 6.49 se muestra cuánta memoria se utilizó, para lograr el mapa con nodos de tamaño 100 x 100. Al dividir el espacio de 1250 de ancho, se necesita 13 matrices de 100 x 100 ancho, 2750 entre 100 es de 28 de largo.

En el Cuadro 6.4 se muestra cuánta memoria se desperdicia en el mapa del patio. Los valores que debe actualizar constantemente en una matriz de ocupación son en total 3500 x 3500, debido a que está ligado al ejemplo donde la mayor posición calculada es 2750 pero por las aperturas del sensor, siempre obtendrá lecturas más lejanas por las aperturas iniciales en el mapa. Las posiciones para MPTE-SLAM solo suman en el peor caso 300, porque recordemos que los valores en cada recorrido pueden modificarse por lecturas nuevas en un mismo ambiente. Aún así, el ahorro de memoria en comparación de MPTE-SLAM con las matrices de ocupación, es bastante. Las actualizaciones deben darse en toda la matriz de ocupación pero en MPTE-SLAM solo ingresa a ese nodo en específico.

Para finalizar la comparación de resultados de memoria, podemos mostrar diferentes aspectos en comparación con las matrices estáticas, versus la estructura planteada en este trabajo. Como en el Cuadro 6.4, la cantidad de memoria utilizada es bastante, a medida que las submatrices deben ser de mayor tamaño. Para número de submatrices, nos referimos a los nodos matrices levantados en ese momento en memoria por el inicio de recorrido. Mostrando que el número, siempre es menor al tamaño de una matriz estática, que cumpla con el almacenamiento de toda la matriz.

El plano aplicado para los resultados, representa una estructura maximizada y el peor de los casos para la aplicación en referencia al comportamiento de crecimiento planteado, entre una estructura de matriz estática y una del tipo MPTE-SLAM. Pero aún este caso sigue siendo mejor, debido a que la apertura de los sonares en cualquiera de las tomas, tiene una generación de aperturas, ninguna estructura real, no presenta un ingreso cerrado como estructura y la apertura de los sensores siempre es de 6 m.

Ahora imaginemos que el robot inicia su recorrido en este plano. El robot, siempre captará algo a su derecha si se encuentra frente al pasillo, generando un crecimiento de hasta 6 m entre la división del espacio en este caso 0.002 daría 3000, dependiendo de la división, todo el tamaño que tendría que aumentar la matriz estática ya no podría ser aplicada en la manera en que trabajan los mapas ocupacionales. En Figura 6.42 (b) representa el crecimiento mantenido que ocurre en la memoria mientras que en (a) el crecimiento va aumentando, entonces si tenemos mayor cantidad de datos, como hemos obtenido por resultados, no sería óptimo.

La característica principal con la que podemos concluir, en base a los siguientes resultados son que la matriz estática, con un almacenamiento para los datos en mapas

como los presentados por las pruebas generarían muchas celdas vacías. Probando que según el tramo del robot, la actualización de los valores tiene un mejor desempeño versus las matrices estáticas.

Capítulo 7

Conclusiones y Trabajos Futuros

1. La estructura MPTE-SLAM es una solución, para el problema de mapeo y localización simultáneo.

2. El resultado de un comportamiento como MPTE-SLAM, hacia todas las direcciones. Permite trabajar, en estructuras con divisiones complejas, tanto pasillos largos o subdivisiones. Las matrices de ocupación, van a generar un mapa, pero no con la profundidad correcta si se encuentra en las últimas posiciones de la matriz, no retornando información importante. Recordando que todo resultado de mapeamiento debe ser en tiempo real.

   Esta estructura también, se acopla al recorrido del robot, si iniciamos la odometría en la posición 0,0. El robot puede retroceder generando una estructura hacia la izquierda y no generando error por los valores negativos.

3. Para demostrar el ahorro en uso de memoria y la optimización del mismo. Los Cuadros 6.4 y 6.3 comprueban que la cantidad de posiciones desperdiciadas con una matriz de ocupación es muy alta, en comparación de MPTE-SLAM. Por cálculo de actualizaciones se obtiene el mismo comportamiento porque cada una de esas posiciones debe actualizarse constantemente, porque el robot puede regresar a posiciones previas. Agregando que constantemente debemos actualizar el gráfico, en la computadora por la retroalimentación.

7.1. Problemas encontrados

La mayoría de problemas encontrados en este trabajo se basan en la diferencia entre lo programado y lo realizado en conjunto con el robot.

Por otro lado debe ser necesario contar con un buen router para la comunicación wifi entre el robot komodo y la computadora que tratará los valores . A mayor recorrido por el robot, aumenta la probabilidad de error en cuanto a las mediciones de los sensores, lo cual es ajeno a la propuesta de esta tesis. Es importante conocer los tópicos de *ros* para determinar porque a veces se puede perder la conexión o si no existe.

7.2. Recomendaciones

Para estandarizar los trabajos en conjunto entre software y hardware, se recomienda estandarizar la data con *ros*.

Verificar constantemente si los sensores están recibiendo información de la computadora del robot, por medio del tópico *rostopic/echo*.

El comando *rqt* (ROS) permite también, el manejo del robot por medio de la computadora, facilitando el recorrido a distancia.

7.3. Trabajos futuros

- La creación del mapa actual permitirá la navegación outdoor (ambiente estructurado) autónomo, evitando obstáculos a través de sus sensores o por una cámara de visión.
- Agregar detalles/especificaciones como el estado de las víctimas encontradas al realizar el mapa en tiempo real para la búsqueda y rescate de personas.
- Realizar el mapa en tiempo real con un dron, por medio de cuaterniones en su espacio de trabajo.

Bibliografía

[1] Barrios-Aranibar, D. (2005). Estratégias Baseadas em Aprendizado para Coordenação de uma Frota de Robôs em Tarefas Cooperativas. Master's thesis, Universidade Federal de Rio Grande do Norte, Natal, Brasil.

[2] Borenstein, J., Everett, H. R., and Feng, L. (1996a). *Navigating Mobile Robots: Systems and Techniques*. A. K. Peters, Ltd., Natick, MA, USA.

[3] Borenstein, J. and Feng, L. (1995). Correction of systematic odometry errors in mobile robots. In *Intelligent Robots and Systems 95.'Human Robot Interaction and Cooperative Robots', Proceedings. 1995 IEEE/RSJ International Conference on*, volume 3, pages 569–574. IEEE.

[4] Borenstein, J., Feng, L., and Borenstein, C. J. (1996b). Measurement and correction of systematic odometry errors in mobile robots. *IEEE Transactions on Robotics and Automation*, 12:869–880.

[5] C.Bezerra (2004). *Localización de um robot movil usando odometria y marcos naturaless, Dissertação de mestrado*. Universidade Federal do Rio Grande do Norte, Natal, RN.

[6] Choset, H. and Nagatani, K. (2001). Topological simultaneous localization and mapping: Toward exact localization without explicit localization. *IEEE Transactions on Robotics and Automation*, 17:125–137.

[7] Clarke, R. (1993). Asimov's laws of robotics: Implications for information technology-part i. *Computer*, 26(12):53–61.

[8] Dai, J. S. (2015). Euler–rodrigues formula variations, quaternion conjugation and intrinsic connections. *Mechanism and Machine Theory*, 92(Supplement C):144 – 152.

[9] Dakulovic, M., Sprunk, C., Spinello, L., Petrovic, I., and Burgard, W. (2013). Efficient navigation for anyshape holonomic mobile robots in dynamic environments. In *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*.

[10] Davis, T. A. and Hu, Y. (2011). The university of florida sparse matrix collection. *ACM Trans. Math. Softw.*, 38(1):1:1–1:25.

[11] de Santana, A. (2008). *Mapeamento com Sonar Usando Grade de Ocupação Baseado em Modelagem, PÓS-GRADUA ÇÃO EM ENGENHARIA ELÉTRICA*. Universidade Federal do Rio Grande do Norte, Natal, RN.

[12] Dellaert, F., Thorpe, C., and Thrun, S. (1998). Super-resolved texture tracking of planar surface patches. In *IEEE/RSJ International Conference on Intelligent Robotic Systems*.

[13] Dissanayake, M. W. M. G., Newman, P., Clark, S., Durrant-whyte, H. F., and Csorba, M. (2001). A solution to the simultaneous localization and map building problem. *IEEE Transactions on Robotics and Automation*, 17:229–241.

[14] Duff, I. S., Grimes, R. G., and Lewis, J. G. (1989). Sparse matrix test problems. *ACM Trans. Math. Softw.*, 15(1):1–14.

[15] Durrant-Whyte, H. and Bailey, T. (2006). Simultaneous localisation and mapping: Part i the essential algorithms. *IEEE ROBOTICS AND AUTOMATION MAGAZINE*, 2:2006.

[16] El-Hakim, S. and Boulanger, P. (1998). Mobile system for indoor 3-d mapping and creating virtual environments. CA Patent App. CA 2,215,690.

[17] Elfes, A. (1989). Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57.

[18] Eustice, R. M., Singh, H., and Leonard, J. J. (2006). Exactly sparse delayed-state filters for view-based SLAM. *IEEE Transactions on Robotics*, 22(6):1100–1114.

[19] Grisetti, G., Tipaldi, G., Stachniss, C., Burgard, W., and Nardi, D. (2007). Fast and accurate SLAM with Rao-Blackwellized particle filters. *Robotics and Autonomous Systems, Special issue on Simultaneous Localization and Map Building*, 55(1):30–38.

[20] Hayward, R. (2001). *The Tortoise and the Love-machine: Grey Walter and the Politics of Electroencephalography*. Sci. Context.

[21] Kakuma, D., Tsuichihara, S., Ricardez, G. A. G., Takamatsu, J., and Ogasawara, T. (2017). Alignment of occupancy grid and floor maps using graph matching. In *2017 IEEE 11th International Conference on Semantic Computing (ICSC)*, pages 57–60.

[22] Kelly, A. and Seegmiller, N. (2015). Recursive kinematic propagation for wheeled mobile robots. *I. J. Robotic Res.*, 34(3):288–313.

[23] Kuhnert, K.-D. and Seemann, W. (2007). Design and realisation of the highly modular and robust autonomous mobile outdoor robot amor. In *Proceedings of the 13th IASTED International Conference on Robotics and Applications*, RA '07, pages 464–469, Anaheim, CA, USA. ACTA Press.

[24] Leonard, J., Durrant-Whyte, H., and Cox, I. J. (1990). Dynamic map building for autonomous mobile robot. In *EEE International Workshop on Intelligent Robots and Systems, Towards a New Frontier of Applications*, pages 89–96 vol.1.

[25] Mataric, M. J. (2007). *The Robotics Primer*. MIT Press.

[26] McManus, C., Upcroft, B., and Newman, P. (2014). Scene signatures: Localised and point-less features for localisation. In *Proceedings of Robotics Science and Systems (RSS)*, Berkley, CA, USA.

[27] Pedrosa, D. P. F. (2012). Mapeamiento de ambientes estructurados con extracción de informaciones geométricas a través de datos sensoriales. Master's thesis, Universidade Federal do Rio Grande do Norte, Brazil.

[28] Ren, C. Y., Prisacariu, V. A., Kähler, O., Reid, I. D., and Murray, D. W. (2017). Real-time tracking of single and multiple objects from depth-colour imagery using 3d signed distance functions. *International Journal of Computer Vision*, 124(1):80–95.

[29] Richard, E., Obozinski, G. R., and Vert, J.-P. (2014). Tight convex relaxations for sparse matrix factorization. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N., and Weinberger, K., editors, *Advances in Neural Information Processing Systems 27*, pages 3284–3292. Curran Associates, Inc.

[30] Saad, Y. (2003). *Iterative methods for sparse linear systems*. Siam.

[31] Saeedi, S., Trentini, M., Seto, M., and Li, H. (2016). Multiple-robot simultaneous localization and mapping: A review. *J. Field Robot.*, 33(1):3–46.

[32] Santos, G. L. (2012). Mapeamiento de ambientes estructurados con extracción de informaciones geométricas a través de datos sensoriales. Master's thesis, Universidade Federal do Rio Grande do Norte, Brazil.

[33] Smith, R. C. and Cheeseman, P. (1986). On the representation and estimation of spatial uncertainly. *Int. J. Rob. Res.*, 5(4):56–68.

[34] Sousa, A. J., Costa, P. J., Moreira, A. P., and Carvalho, A. S. (2005). Self localization of an autonomous robot: using an EKF to merge odometry and vision based landmarks. In *ETFA*. IEEE.

[35] Sousa, P. P. and Muranho, J. (2011). Sparse matrix-vector multiplication on gpu: When is rows reordering worthwhile. In *International Conf. on Engineering - ICEUBI*, volume 1.

[36] Souza, A. and Gonçalves, L. (2015). Occupancy-elevation grid: an alternative approach for robotic mapping and navigation. 1:18.

[37] Suger, B., Tipaldi, G. D., Spinello, L., and Burgard, W. (2014). An approach to solving large-scale slam problems with a small memory footprint. In *Proc. of The International Conference in Robotics and Automation (ICRA)*.

[38] Thrun, S. (2002). Probabilistic robotics. *Commun. ACM*, 45(3):52–57.

[39] Thrun, S., Burgard, W., and Fox, D. (2005a). *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press.

[40] Thrun, S., Burgard, W., and Fox, D. (2005b). *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press.

[41] Yang, S. and Scherer, S. (2017). Direct monocular odometry using points and lines. In *Robotics and automation (ICRA), 2017 IEEE international conference on*. IEEE.

[42] Yang, S. W. and Wang, C. C. (2011). Feasibility grids for localization and mapping in crowded urban scenes. In *2011 IEEE International Conference on Robotics and Automation*, pages 2322–2328.