Generación de trayectorias para tareas de navegación autónoma y mapeo en ambientes interiores

Viridiana Yatzen Hernández Márquez

Universidad Politécnica de Tulancingo
Maestría en Automatización y Control
Tulancingo de Bravo, Hgo., México
Diciembre 2018

Generación de trayectorias para tareas de navegación autónoma y mapeo en ambientes interiores

Ing. Viridiana Yatzen Hernández Márquez

Becaria de CONACyT, expediente No.:
787113

Tesis presentada como requisito para optar al título de:
Maestra en Automatización y Control

Asesores:
Dr. Rafael Stanley Núñez Cruz
Dr. Carlos Enríquez Ramírez

Línea de Investigación:
Automatización y Control

Universidad Politécnica de Tulancingo
Maestría en Automatización y Control
Tulancingo de Bravo, Hgo., México
2016 - 2018

ACTA DE COLOQUIO DE TESIS

En la **UNIVERSIDAD POLITÉCNICA DE TULANCINGO**, el día 07 de Diciembre de 2018, se reunieron los Doctores: **DR. LUCIO FIDEL REBOLLEDO HERRERA, DR. HIPÓLITO AGUILAR SIERRA, DR. RAFAEL STANLEY NUÑEZ CRUZ, y DR. CARLOS ENRÍQUEZ RAMÍREZ,** integrantes del jurado de **PRE-EXAMEN DE GRADO DE LA MAESTRÍA EN AUTOMATIZACIÓN Y CONTROL** de la ingeniero:

Viridiana Yatzen Hernández Márquez

Que presentó el trabajo de Tesis titulado: **"Generación de trayectorias para tareas de Navegación Autónomas y mapeo en ambientes interiores".**

Después de escuchar la presentación y defensa del trabajo de tesis, el jurado examinó y resuelve: Aprobado Por Unanimidad

DR. LUCIO FIDEL REBOLLEDO HERRERA
Presidente

DR. HIPÓLITO AGUILAR SIERRA
Secretario

DR. RAFAEL STANLEY NUÑEZ CRUZ
Vocal

DR. CARLOS ENRÍQUEZ RAMÍREZ
Vocal

Calle Ingenierías No. 100, Huapalcalco Hidalgo. C.P. 43629
Tel. (775) 75 5 82 02 Ext. 1650 - 1651
www.upt.edu.mx

A mi madre... por el gran amor que nos tienes, por el apoyo incondicional que siempre me has dado, por tener la fortaleza de salir adelante sin importar los obstáculos, por la motivación constante que me ha permitido ser una persona de bien, y por ser la mujer que me dió la vida y me enseñó a vivirla... no hay palabras en este mundo para agradecerte mamá.

A ti mi amore... por tenerme paciencia, porque siempre me apoyaste, pero sobre todo por tanto amor... gracias por creer en mi.

Agradecimientos

En primer lugar quiero agradecer a mis asesores, el Dr. Rafael Stanley Núñez Cruz y el Dr. Carlos Enríquez Ramírez, por su gran ayuda para el desarrollo del presente trabajo, por su gran esfuerzo y dedicación, por guiarme y por enseñarme con amabilidad y paciencia, pero sobre todo por sus sabios consejos que me han ayudado a ser mejor y a crecer profesionalmente.

A mis profesores y compañeros... gracias por su tiempo, por compartir el conocimiento que llevan, por sus consejos, su paciencia y por la buena convivencia y los momentos divertidos llenos de risas, sé que cuento con ustedes.

Al Consejo de Ciencia y Tecnología (CONACyT-México) le agradezco la beca recibida durante la maestría.

A la escuela que me ha visto crecer, la Universidad Politécnica de Tulancingo, por el apoyo brindado durante este tiempo que me tomó desarrollar este proyecto de investigación.

Un agradecimiento muy especial al Dr. Juan Manuel Ibarra Zannatha del CINVESTAV, por su apoyo y confianza, así como también por el préstamo de la plataforma para el desarrollo de este trabajo.

A mis familiares y amigos quienes me han apoyado en los momentos buenos y en los difíciles, gracias por ser parte de mi vida.

Resumen

En este trabajo, se propone implementar un algoritmo de generación de rutas durante el proceso de creación de mapas autónomos por medio de un robot móvil. El problema a resolver es obtener automáticamente la ruta óptima en un ambiente desconocido para explorar y mapear todo el entorno.

Las principales contribuciones de este trabajo son: la formulación del problema de optimización, el análisis de la imagen propuesta para implementar la evaluación de la función objetivo y la comparación de diferentes técnicas de optimización aplicadas a este problema en pruebas experimentales.

La plataforma utilizada para la implementación es el robot de código abierto llamado TurtleBot 2, el algoritmo propuesto se implementó en un nodo ROS (*Robot Operating System*) que funciona en paralelo a los programas de mapeo y odometría ya incluidos en este sistema operativo.

Palabras clave: Generación de trayectoria, Optimización heurística, Algoritmos genéticos, Robots móviles..

Abstract

In this work we propose to implement a route generation algorithm during the process of creating autonomous maps by means of a mobile robot. The problem to solve is to automatically obtain the optimal route in an unknown environment to explore and map the entire environment.

The main contributions of this work are: the formulation of the optimization problem, the analysis of the proposed image to implement the evaluation of the objective function and the comparison of different optimization techniques applied to this problem in experimental tests.

The platform used for the implementation is the open source robot called TurtleBot 2, the proposed algorithm was implemented in a ROS node (*Robot Operating System*) that works in parallel to the mapping and odometry programs already included in this operating system.

Keywords: Trajectory Generation, Heuristic Optimization, Genetic Algorithms, Mobile Robots.

Índice de figuras

Índice general

1. Introducción

Se da inicio a este trabajo con la discusión sobre ¿qué es un robot? Esta palabra fue utilizada en 1921 en la obra de teatro R.U.R. (Robots Universales Rossum) del escritor checo Karel Čapek, para referirse a trabajadores artificiales, el término se derivaba de la palabra checa: *robota*, que significa trabajo forzado [Čapek, 1968].

El Instituto de Robótica de América (RIA) define a un robot como: un aparato o manipulador programable, multifunción designado para mover partes, herramientas u otros objetos especiales por medio de una serie de movimientos programados para la realización de sus diferentes tareas [Nehmzow, 2012].

Aunque existen varias definiciones del termino, queda claro que un robot es un aparato o entidad que puede ser programado para realizar alguna tarea específica en el mundo real y que para lograrlo requiere la habilidad de desplazarse en el entorno que lo rodea y ser capaz de interactuar con otros objetos existentes relevantes para su tarea, por lo que cierta interpretación del estado de su entorno es necesaria [Zannatha Juárez, 2010].

Figura **1-1**.: Campos de aplicación de los robots.

El objetivo de los robots es ayudar o reemplazar personas en tareas peligrosas [Trevelyan, Hamel y Kang, 2016] o tareas repetitivas o aquellas que requieren un alto grado de precisión [Chui, Manyika y Miremadi, 2015]. Inicialmente, el campo de aplicación de los robots estaba

restringido a los procesos de fabricación dentro de las industrias [Graetz y Michaels, 2015], hoy el trabajo de los robots se ha expandido a un gran número de áreas, incluyendo medicina [Nakadate y J. Arata, 2015], seguridad [Lapeña et al., 2014], inspección [Carreras et al., 2012], entre otros. Estos nuevos campos de aplicación implican desafíos más complicados porque el área de trabajo generalmente cambia, a veces se desconoce, además de una variedad de entornos como el agua, la tierra y el aire, tal como se ilustra en la figura **1-1**.

Si bien todo robot actúa en el mundo real y tiene partes móviles, se debe hacer la distinción entre robots móviles y fijos: Los robots móviles tienen actuadores especializados que les permiten desplazarse dentro de cierto medio y su movimiento es regido por su locomoción y limitado por la física del medio en el cual se desplaza; los robots fijos pueden contar con varios grados de libertad para realizar tareas muy precisas, pero tienen la restricción de que están sujetos a una base inmóvil, por lo que únicamente operan en un rango limitado de espacio. Actualmente la mayoría de los robots son de este tipo: fijos, y operan como manipuladores con fines industriales en líneas de ensamble. También existen diversos ejemplos de robots móviles para exploración y servicio, aunque en mucho menor número. En estas dos áreas las condiciones de trabajo rara vez pueden ser controladas. Algunos ejemplos de robots especializados para tareas en diversos entornos son presentados en [Siegwart et al., 2011], donde se abordan detalles de la locomoción de varios tipos de robots: articulados, industriales, aéreos, y acuáticos entre otros, como se ilustra en la figura **1-2**. Se hace la mención de que en este trabajo se hablará de robots de control diferencial debido a la plataforma que se utiliza para las pruebas.

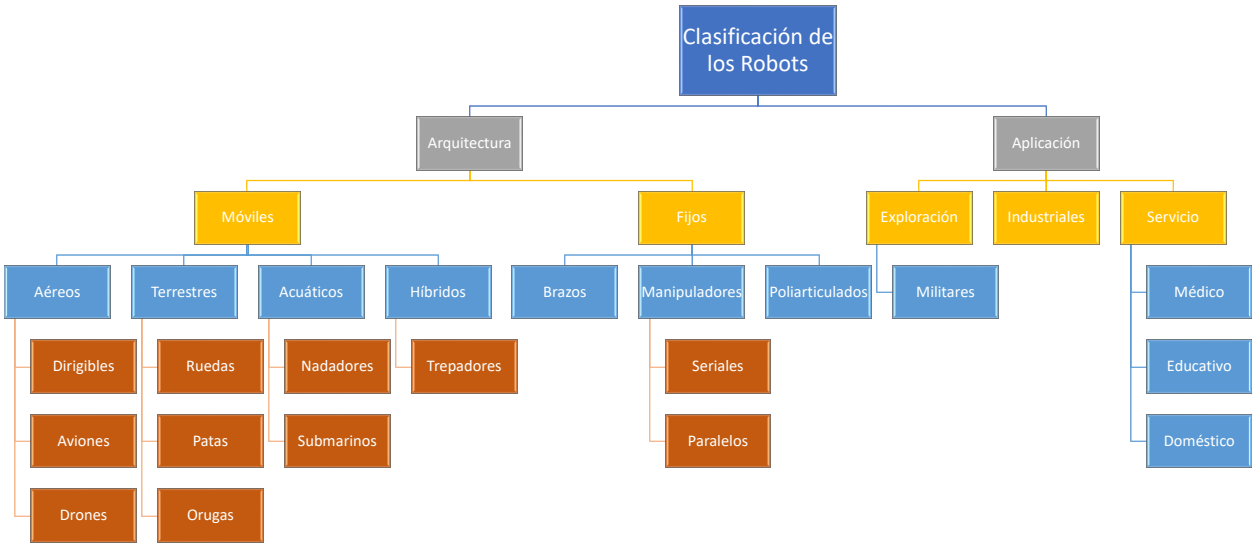


Figura **1-2**.: Clasificación de los robots.

Si se desea que un robot reemplace a una persona dentro de su propio entorno, será necesario que el robot tenga las mismas capacidades que permiten que el ser humano se mueva

en estos entornos. Esta es una tarea complicada que requiere resolver diferentes problemas como la representación del entorno en el que se mueve el robot [Hoy, Matveev y Savkin, 2015; Yu, Zamora y Soria, 2016], la generación automática de trayectorias [Capito et al., 2016], la detección de obstáculos [Schwager et al., 2017], la comunicación entre los agentes [Elizondo et al., 2016], entre otras situaciones.

Existen algunas ocasiones en las que el entorno en el que se desarrolla el robot es tan complicado que es necesaria la intervención de una persona para tomar decisiones en momentos adecuados para evitar accidentes o errores de funcionamiento, este tipo de operación se conoce como teleoperación [Pierce, 2015], como se observa en la figura **1-3**.

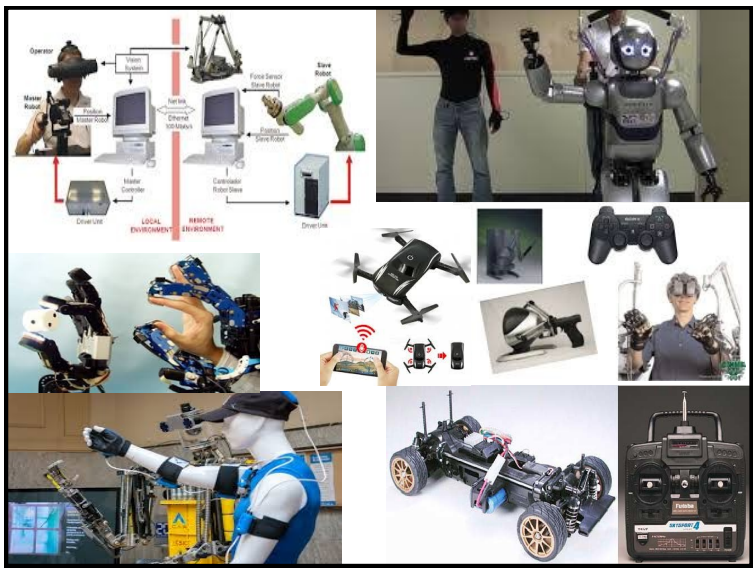


Figura **1-3**.: Teleoperación.

Un robot móvil que carece de información sobre su entorno, incluso de su propia posición inicial al arranque, pero que posee la habilidad de crear un mapa de su entorno mientras lo explora, podría luego definir y ejecutar trayectorias a cierta zona visitada previamente para poder realizar algún objetivo en particular sin intervención de un operador [Zannatha Juárez, 2010]. Hay situaciones en las que no es posible teleoperar a un robot, ya sea debido a limitaciones ambientales (entornos subacuáticos o cerrados) o limitaciones de comunicaciones debido a desastres naturales.

La comunidad científica de robots móviles tiene como uno de sus objetivos el de crear robots autónomos, si bien este objetivo implica resolver varios problemas como son: planeación de rutas, leyes de control, evasión de obstáculos, entre muchos otros. La robótica autónoma, intenta proporcionar a los robots las capacidades necesarias para resolver los problemas que enfrentan por sí solos. Esto puede implicar el uso de técnicas avanzadas de aprendizaje [Duan y Qiao, 2015].

En este sentido, en el presente trabajo de tesis se realiza la aportación a la creación de

mapas de entorno, debido a que se trata de un tema de gran importancia como se muestra en los siguiente trabajos: por ejemplo en [Fernández, 2018], del mismo modo en [Gil et al., 2007], además se muestra otro enfoque en [Restrepo et al., 2009]. Este trabajo se centra en el problema de la navegación para la creación de mapas en ambientes interiores cerrados a través de la generación autónoma de trayectorias.

1.1. Estado del arte.

En este apartado se presenta el estudio del estado del arte referente a los distintos trabajos sobre algoritmos de navegación y la forma en que los autores han resuelto éste problema. Del mismo modo, se estudian los distintos métodos de navegación y locomoción autónomas, así como la manera de recorrer el espacio.

En el artículo propuesto por [Velásquez Hernández, Chávez Chávez y Córdoba Nieto, 2015], los autores resuelven el problema de navegación y evasión de obstáculos usando la técnica Héctor*SLAM* como algoritmo de localización y mapeo implementando en un robot diferencial y *ROS* en entornos internos, dinámicos y semi estructurados, resaltan que se genera un algoritmo que genera y refina una trayectoria de navegación válida y segura para el robot.

En [González Arjona, 2011] implementan el algoritmo BOBMapping, basado en una simplificación de la técnica de "Rejilla de Ocupación" en un robot móvil basado en una FPGA, que genera un mapa en 2D de un entorno cerrado y no estructurado. Utilizaron el sistema de balizas ultrasónico y radiofrecuencia para obtener mediante trilateración las coordenadas relativas de éste, y encoders de rueda.

Por otra parte en [Zannatha Juárez, 2010], se implementa una técnica para la creación automática de mapas de entorno, en un robot móvil diferencial con FPGA y una cámara digital ya que se busca crear un mapa de su entorno utilizando visión. Trabajan con marcas artificiales en el techo de una zona particular aislada con suelo plano y liso. Al final comprueban la exactitud del mapa obtenido.

El proyecto de vehículo autónomo desarrollado en el AMTC [Parra Tsunekawa, 2014] está implementado en un Volkswagen Tiguan. Se utilizan IMU y sensores LIDAR, así como un sistema de estimación del estado del vehículo basado en el uso de un Filtro de Kalman Extendido en conjunto con una estructura de mapa de ocupación para generar un mapa de entorno local y está basado en la plataforma *ROS*.

En [Correa y Vásquez, 2012] se implementa una planificación de trayectorias para robots agrícolas mediante el algoritmo A* sobre un mapa 2D obtenido a partir de un sensor *Kinect*, el seguimiento de trayectorias se implementó con una aproximación numérica mediante el método de Euler y los parámetros correspondientes fueron obtenidos mediante algoritmos genéticos.

En el artículo [Guzmán Luna, Arango Sánchez y Jiménez Pinzón, 2012], se estudia la generación de trayectorias entre dos puntos mediante el mapa de visibilidad, se compara el

algoritmo genético y el algoritmo Dijkstra para encontrar la ruta óptima entre las trayectorias generadas. Implementan el algoritmo en Java y realizan pruebas en el robot *LEGO NXT 2.0* mediante el uso de sensores como compás y tacómetro, y el concepto de la odometría.

Se tiene que en [Moya Carrasco, 2016], el autor utiliza diagramas de Voronoi para modelar el entorno permitiendo determinar las zonas más alejadas de los obstáculos, se usa el algoritmo Dijsktra para la búsqueda del camino más corto. En la obtención de trayectorias para entornos conocidos utilizan MATLAB. Además se hace uso de ROS y el robot *TurtleBot* con una cámara *Kinect* a bordo en el entorno de simulación Gazebo.

Por otra parte, en [López et al., 2006] se describen los fundamentos del algoritmo RRT y se detallan las versiones más significativas aportadas por bibliografía reciente. Se ilustra su aplicación en sistemas robóticos no holónomos. Presentan una técnica basada en el concepto de maniobra restringida y su aplicación tanto a vehículos con guiado diferencial como con configuración Ackerman.

En [Rascón Crespo, 2014] se desarrolla un algoritmo de seguimiento de trayectorias basado en el conocido método de navegación *pure-pursuit* y se integra en la plataforma Manfred (manipulador móvil) usando *ROS*. Además usan el algoritmo de planificación de trayectorias denominado "fast marching".

Se observa que en [Ahmed Omara y Mohamed Sahari, 2015] se optimizan los parámetros de *gmapping* para mejorar la precisión de la generación del mapa y del escaneo láser. Se utiliza *ROS* y *TurtleBot* 2 y se hace una comparativa entre el sensor *Kinect* y un buscador de rango láser no filtrado. Se probó el rendimiento de la exploración de fronteras en un entorno oscuro mientras se realizaba *SLAM* junto con la técnica propuesta.

En [Hamzeh y Elnagar, 2015], el autor propone teleoperar un robot móvil de bajo costo para recopilar información y construir un mapa global de un entorno desconocido en una estación de trabajo remota. Utilizan la plataforma *ROS* y *TurtleBot* con un sensor *Kinect* para construir mapas globales de interiores para descubrir áreas inseguras donde no es posible enviar personal humano.

Actualmente es posible encontrar muchos otros trabajos relacionados con la generación de trayectorias como en [Quinonez, I.Tostado y Burgueno, 2015], también en [Liang et al., 2017], existe otro enfoque encontrado en [Ganeshmurthy y Suresh, 2015] y en [Song, Wang y Sheng, 2016].

En esta casa de estudios, la Universidad Politécnica de Tulancingo, se han encontrado trabajos relacionados con la robótica móvil, como el trabajo de tesis de [Sosa Cortés, 2017], en donde se observa el control de seguimiento de trayectorias mediante dos tipos de esquemas de control aplicados a un robot móvil *Ackerman*, utiliza algoritmos de optimización basados en el comportamiento de enjambres inteligentes: de murciélagos y de ballenas; y el método de polinización de las flores. En el trabajo se formulan trayectorias mediante curvas de Bézier y curvas paramétricas para las pruebas de físicas y de simulación.

Con base en los trabajos analizados previamente, se observa que existen grandes variantes a lo que se refiere al mapeo y generación de rutas, tanto en las técnicas y formas de resolver

o de aportar a la solución del problema, como en la forma en que se realizan las pruebas experimentales de forma física mediante el uso de diferentes plataformas.

Aún así, en la bibliografía reciente no se ha encontrado un tema completamente similar al que se realiza en este trabajo de tesis, es por ello que se llega a la conclusión de que realmente tiene aportaciones importantes al campo de la implementación física y comparación de algoritmos de optimización en robots móviles, sobre todo para este tipo de tareas de generación de rutas para mapeo autónomo.

1.2. Planteamiento del problema.

La robótica es la ciencia de la percepción y manipulación del mundo real a través de dispositivos mecánicos controlados por computadoras, lamentablemente la mayor parte de la robótica aún se encuentra en su infancia. Pero su investigación es motivada en gran parte por el potencial que tendrían agentes autónomos para cambiar a la sociedad, por mencionar algunas posibilidades: podríamos delegar la responsabilidad del manejo de vehículos a robots para reducir la posibilidad de accidentes, utilizarlos en ambientes hostiles para el ser humano como la construcción, minería, el manejo de desechos tóxicos o simplemente como robots de servicio que se hagan cargo de las tarea tediosas del hogar como barrer, sacar la basura, pasear al perro, y algunas otras. [Thrun et al., 2005].

Los robots móviles son una gran aproximación a agentes inteligentes, y un gran sueño del ser humano es el de crear aparatos que puedan imitar a seres vivos. Los conceptos de percepción y acción están fuertemente unidos en los seres vivos, al interactuar con el entorno los animales anticipan el resultado de sus acciones y predicen el comportamiento de otros objetos.

Los mapas o representaciones del entorno son importantes en la robótica móvil debido a su obligatorio uso para tareas de navegación, actualización y exploración del entorno mismo.

Surgen varios problemas al tratar de obtener el mapa de un entorno desconocido con un robot móvil terrestre como lo es el TurtleBot debido a que necesariamente se requiere de la participación del ser humano a través de la teleoperación (manejo a distancia por parte del usuario). Esto a su vez requiere una buena señal de *Wi-Fi* para realizar la tarea de manera eficiente ya que si la comunicación no es buena el robot puede experimentar ciertos errores de funcionamiento que tal vez harían que el mapa resultante quede totalmente diferente al entorno original, o peor aún, el robot podría dañarse si colisiona con algún objeto no detectado.

Se sabe que la tendencia de la robótica en la actualidad es la autonomía, es decir, no depender de un usuario que tome el control del robot para hacerlo moverse a cualquier dirección, sino que el robot sea capaz de tomar sus propias decisiones basadas en criterios previamente establecidos. Con lo anterior se buscan métodos que ayuden a hacer un mapa a pesar de estas dificultades.

Recientemente en la Ciudad de México se vivió un desastre natural, un sismo de gran

escala que provocó que varios edificios cayeran. Esto hizo que la idea de esta problemática se pudiera implementar no solo para la investigación, sino que en un ambiente como lo es un edificio derrumbado, que tal vez no tiene acceso para las personas y que además no tiene señal de *Wi-Fi* debido a que en un derrumbe se pierden todo tipo de señales, se pudiera tener acceso mediante robots mas pequeños para hacer un mapa de la zona y apoyar así a que los rescatistas puedan tener un panorama mas amplio de como entrar al lugar para realizar su labor sin exponer su vida. Esto claro es sólo una idea de la implementación que se le podría dar al trabajo aquí presentado.

Es por ello que en este trabajo de investigación se ha decidido generar un aporte a esta problemática, se pretende implementar algoritmos de generación de rutas de forma aleatoria en un inicio para hacer que el robot se vuelva autónomo en cierta medida para crear mapas de su entorno, como se mencionó, sin que se requiera de algún tipo de señal remota o control teleoperado.

1.3. Objetivo general.

Seleccionar e implementar un algoritmo de navegación autónoma que le permita a un robot móvil identificar en una imagen las zonas de interés de búsqueda y zonas a evadir, así como generar una trayectoria óptima que le permita al robot navegar en ambientes interiores cerrados para generar un mapa completo del entorno circundante.

1.4. Objetivos particulares.

- Uso de visión artificial para analizar digitalmente las imágenes (mapas) obtenidas a través del sensor.
- Identificar los puntos en el mapa que permitan clasificar tres zonas: de interés, mapeadas y desconocidas.
- Plantear un problema de optimización para generación de rutas.
- Implementar y comparar diferentes técnicas de optimización aplicadas al problema de optimización como: Blind Search, Hill Climbing y Algoritmos Genéticos.

1.5. Justificación.

La robótica móvil tiene la necesidad de saber en dónde se encuentra el robot para la toma de decisiones, por lo que se busca una representación del entorno o mapa que nos dé información fiable de este entorno.

Un robot móvil que puede y debe desplazarse con cierta libertad en un entorno dado para realizar alguna tarea, tiene ahora que enfrentarse con el problema de navegación, que puede

resumirse a tres preguntas de acuerdo con [Fu, Gonzalez y Lee, 1987]: ¿en dónde estoy?, ¿a dónde debo ir?, ¿y cómo puedo llegar ahí?, es de aquí que surge la necesidad de técnicas para planeación de rutas, evasión de obstáculos, localización y mapeo.

El problema de mapeo tiene probablemente más de tres décadas de ser un campo de investigación, al inicio se alimentaba al robot con un mapa prefabricado (el problema de localización), pero en realidad tal modelo no podría crear a un robot autónomo, el robot debería ser capaz de crear el mapa por sí mismo (el problema de mapeo).

Concretamente, a la necesidad de crear un mapa y, simultáneamente, resolver el problema de localización, a esta dependencia entre los problemas dio origen al término *SLAM*, por sus siglas en inglés: *Simultaneous Localization and Mapping* [Durrant-Whyte y Bailey, 2006]. La solución a este problema dota al robot de la autonomía necesaria para realizar otras tareas de alto nivel [Galdeano, 2011].

La construcción de mapas implica determinar la posición de objetos de interés relativa a cierta referencia global como el plano cartesiano. Para llevar a cabo esta labor el robot debe hacer mediciones relativas a tales objetos o entidades de interés por lo que requiere conocer su propia posición, pero debido a la inexactitud de su movimiento antes debe deducirla a partir de la posición de los objetos de interés, requiriendo que ambos problemas sean atendidos concurrentemente [Thrun, Burgard y Fox, 1998].

En los últimos años la importancia de la robótica móvil ha crecido de manera considerable, sobre todo en aplicaciones para la vida diaria del ser humano. Los robots de servicio se vuelven esenciales para labores cotidianas en las cuales deben interactuar en un entorno acotado, es ahí en donde se vuelve indispensable que el robot conozca su posición actual y el objetivo hacia el cual debe dirigirse, por lo que el ser capaz de obtener un mapa es primordial para el buen funcionamiento de los movimientos del robot, ya que al obtener una representación del entorno se debe tener la certeza de desplazarse en áreas seguras, además de que el ser humano puede observar el entorno sin necesidad de exponerse.

1.6. Alcances y limitaciones.

El presente trabajo pretende desarrollar un algoritmo de generación de trayectorias para un robot móvil diferencial, e implementarlo en este caso el robot de código abierto llamado TurtleBot 2.

Se requiere hacer uso de las propiedades del middleware robótico *ROS* (*Robot Operating System*) que funciona en paralelo a los programas de mapeo y odometría ya incluidos en las librerías de este sistema operativo para la implementación del algoritmo.

En este trabajo enfocaremos nuestros esfuerzos en el problema de la creación de mapas a través de la generación de trayectorias óptimas para navegación autónoma. Se debe mencionar que la generación del mapa se realizará por ahora solo en ambientes interiores cerrados debido a la naturaleza de la plataforma robótica, las ruedas y el sistema de locomoción que el TurtleBot 2 tiene no es suficientemente apto para que se desplace en superficies que no

sean planas o que tengan demasiada textura (como lodo, pasto, piedras), además de que el alcance de visión de la cámara *Kinect* que tiene integrada igualmente funciona mejor en ambientes cerrados en los que no existan cambios de luz muy drásticos ya que puede afectar la visión del programa de mapeo. Se espera un resultado parecido al de la figura **1-4**.

Figura **1-4**.: Mapa esperado con el proceso.

En caso de que se requiera que el sistema sea capaz de exponerse al medio ambiente en espacios abiertos se deberá cambiar la plataforma robótica por una con ruedas mas grandes o con un sistema de locomoción apto para exteriores y experimentar en ella antes de asegurar su correcto funcionamiento, además de que es muy probable que una cámara común no sea suficiente para reconocer obstáculos y se requiera de hacer cambios también en esa parte del sistema de visión.

1.7. Organización del trabajo.

El trabajo se presenta de la siguiente manera: en la sección II se describe el problema de la navegación autónoma, y se define el comportamiento general del proceso de mapeado así como la plataforma de implementación que se utilizó en este trabajo; la sección III describe el proceso de creación del mapa así como su interpretación; en la sección IV se mencionan algunos generadores de trayectorias que existen en la actualidad y se propone el generador que se utilizó en este trabajo; en la sección V se hace un breve resumen de los métodos de optimización utilizados para resolver este problema; la sección VI describe los resultados de las comparaciones entre los algoritmos utilizados en las pruebas experimentales, y finalmente la sección VII muestra las conclusiones y el trabajo futuro.

2. Problema de navegación autónoma.

Existen varias formas de resolver el problema de la navegación autónoma, como se menciona en la sección 1, es por ello que en este trabajo se propone abordarlo como un problema de optimización para el cual es necesario definir claramente las variables de decisión, el espacio de búsqueda, la función de costo y las restricciones.

El propósito del algoritmo es generar las trayectorias (rutas) que le permiten al robot navegar en toda el área en la que se encuentra para generar un mapa completo del entorno circundante.

2.1. Comportamiento general

A continuación se muestra el proceso de mapeado que se ha programado en el robot, como se puede observar en la figura **2-1** se describe un comportamiento cíclico ya que los pasos se repiten continuamente hasta obtener el mapa deseado, además en esta sección se describirán cada una de las partes que hacen que el robot se mueva y sea capaz de generar de forma automática el mapa de su entorno.

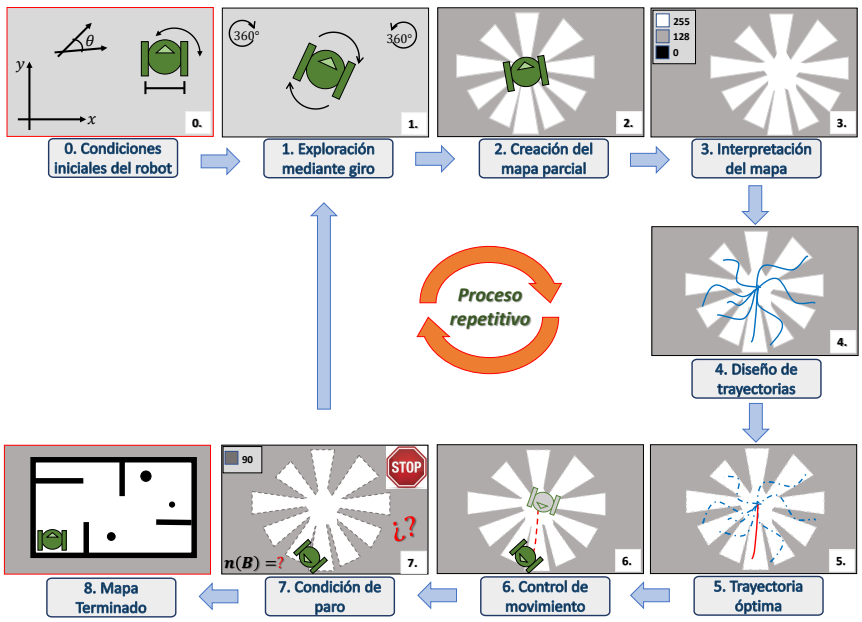


Figura **2-1**.: Diagrama de flujo del comportamiento general del robot.

En la figura se observa una secuencia que se ejecuta cada vez que el robot quiere realizar un movimiento. Para explicar esta secuencia hay que partir de la situación en la que el robot quiere ir desde su posición actual hasta la posición objetivo. En primer lugar, el subsistema encargado de la percepción del robot extrae información del entorno mediante los sensores que tiene. Posteriormente se construye un mapa con los datos obtenidos del medio que lo rodea y se localiza al robot en ese entorno. En seguida se determina el camino a seguir desde la posición actual hasta el objetivo, y finalmente se ejecutan los movimientos necesarios para que el robot siga el camino.

El proceso consta de las siguientes etapas:

0. **Condiciones iniciales del robot:** Cuando el robot inicia el proceso de mapeado se tienen ciertas condiciones en cuanto a la forma en que se debe obtener su posición y orientación en el mapa y a la forma de colocarlo en el área de pruebas físicas la primera vez que se ejecutará el algoritmo, ya que de este modo se puede continuar de manera eficiente con el proceso de mapeado.

   En este punto el robot reconoce los elementos que están a su alrededor y empieza a crear las bases de la odometría, es decir, en este punto es donde el robot comienza a ubicarse en el mapa gracias a los sensores que tiene, de ese modo se logra fijar como punto inicial o coordenadas "0"para dar inicio al proceso de mapeado.

   Es importante mencionar que el robot móvil tiene su propio eje de coordenadas, éste se mueve con él y es llamado marco o *frame* del robot. Este tipo de marco es comúnmente usado en la robótica, en el caso del *TurtleBot2*, se localiza de la siguiente manera: el eje X se encuentra alineado con el eje longitudinal del robot; el eje Y se encuentra perpendicular a este eje, apuntando al lado izquierdo del robot alineado con las ruedas motoras del robot; finalmente el eje Z apunta perpendicularmente a los ejes anteriores, pero hacia la vista superior del robot en este sistema, tal como se ilustra en la figura **2-2**.

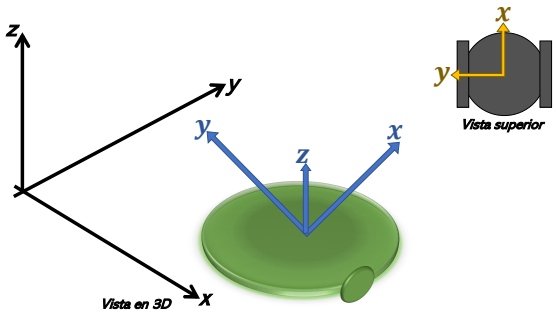


Figura **2-2**.: Marco de referencia del robot.

La posición puede ser estimada con las condiciones iniciales, el robot puede asumir cualquier posición y orientación deseada, tomando las condiciones iniciales en 0, estos

valores colocan al robot en el centro del plano de coordenadas de referencia, con una orientación inicial paralela al eje X del plano de coordenadas.

1. **Exploración mediante giro:** Es importante mencionar que el sensor de visión que utiliza el robot, tiene un campo de visión horizontal de 57 grados desde la base del sensor y tiene un alcance de profundidad de hasta 3 metros, eso hace que la primera vez que toma datos sea solamente una vista triangular y frontal de una parte del ambiente a su alrededor.

   Tomar solamente esa vista del mapa hace que se complique la generación y selección de rutas posteriormente en la parte de programación, ya que no se trata de un espacio pequeño para ese tipo en específico de sensor, pero en la práctica y para la tarea de mapeado que se requiere en este trabajo no es suficiente. Como se ilustra de manera representativa en la figura **2-3**.

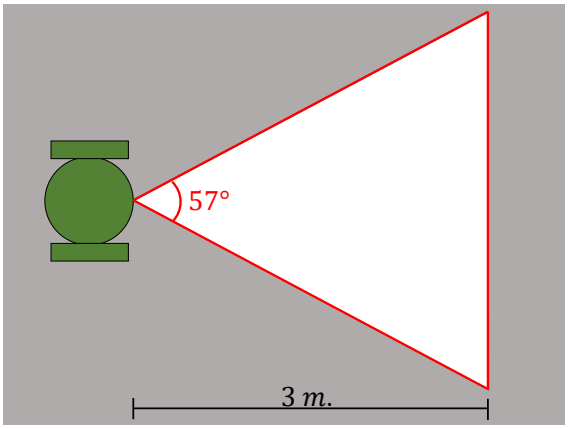


Figura **2-3**.: Primera captura del mapa del sensor de visión.

Debido a las limitantes de visión que se tienen por parte de la plataforma robótica, para obtener mejores resultados al momento de generar las trayectorias se decidió generar un mejor mapeado de la zona circundante al robot y en todas las direcciones posibles haciendo que este se pueda mover hacia diferentes direcciones, a diferencia de si se tomara solamente la captura inicial del sensor de visión ya que solo podría moverse en un ángulo limitado.

Por lo tanto, como siguiente paso en el proceso, el robot debe girar sobre si mismo para hacer el mapeado de la zona en la que se encuentra actualmente, esto se realiza mediante cinco movimientos en diferentes posiciones elegidas de manera aleatoria y a diferentes velocidades angulares para realizar un giro completo, entre cada movimiento el robot espera un tiempo de seis segundos para ir capturando de forma efectiva los datos del medio que lo rodea.

En cuanto se completa el giro se procede a continuar con el proceso de mapeado, y en

caso de que no se haya encontrado una buena ruta para seguir, el robot gira nuevamente pero en sentido contrario al que lo hizo la primera vez.

Se programaron los movimientos del giro haciendo que los ángulos variaran entre 60 y 180 grados, del mismo modo se variaba el tiempo de ejecución del giro entre tres y cinco segundos, estas variaciones se realizan en forma aleatoria en cada ocasión. Por ejemplo: el ángulo de giro podría resultar de 60 grados y la velocidad de cuatro segundos, entonces el robot ejecutaría una sexta parte de un giro completo en ese tiempo específico, después el robot espera para capturar los datos y nuevamente gira aleatoriamente, así hasta completar los cinco movimientos requeridos.

Como medida de compensación a esta problemática, se decidió que los ángulos y tiempos de giro sean aleatorios para colocar al robot en posiciones diferentes cada vez que se inicie el proceso de generación de rutas, esto con la finalidad de aumentar las posibilidades de crear y seleccionar buenas rutas, así como de que se pueda colocar en diferentes posiciones cada vez, para darle mayor oportunidad al algoritmo de encontrar mejores trayectorias.

2. **Creación del mapa parcial:** El problema de localización y mapeo simultáneo ($SLAM$) pregunta si es posible colocar un robot móvil en una ubicación y en un entorno desconocido, para que construya un mapa coherente de ese entorno al mismo tiempo que determina su ubicación dentro de este mismo mapa para hacer que el robot sea realmente autónomo [Durrant-Whyte y Bailey, 2006].

   Una de las soluciones al problema de la autonomía de los vehículos móviles puede proporcionarse mediante la implementación de un algoritmo probabilístico recursivo que construye un mapa del entorno mientras localiza el vehículo móvil al mismo tiempo, minimizando los errores.

   Aunque este algoritmo está procesando demanda de tiempo, se convierte en una solución poderosa cuando el vehículo tiene que navegar a través de entornos desconocidos o no declarados, obteniendo un mapa confiable del mismo [Cheein et al., 2010].

   $SLAM$ ha sido formulado y resuelto como un problema teórico en varias formas diferentes [Dissanayake et al., 2001]. Del mismo modo se ha implementado en varios dominios diferentes, desde robots para interiores, sistemas al aire libre o bajo el agua y en el espacio aéreo.

   La gran mayoría de trabajos se han enfocado en mejorar la eficiencia computacional [Dissanayake, Durrant-Whyte y Bailey, 2000] y al mismo tiempo garantizar estimaciones consistentes y precisas para el mapa y la posición del vehículo.

   Por otra parte, se han realizado muchas investigaciones sobre otras cuestiones [Ballesta et al., 2010] vitales para lograr una implementación de $SLAM$ práctica y sólida [Bailey y Durrant-Whyte, 2006].

El proceso del *SLAM* consiste en un cierto número de pasos: extracción de características, asociación de datos, estimación del estado y actualización de las características. La finalidad del proceso es usar el entorno para actualizar la posición del robot. Dado que la odometría del robot no es enteramente fiable, no se puede depender directamente de ella [Andrade y Martin, 2007].

Se deben usar sensores de distancia para corregir esta posición. Esto se consigue extrayendo características del entorno y observándolas mientras el robot se mueve. Un filtro extendido de Kalman (EKF, por sus siglas en inglés) es responsable de actualizar la estimación de la posición del robot basándose en estas características, llamadas puntos de referencia o *landmarks* [Viñals, 2012].

El filtro mantiene un estimado de la incertidumbre en la posición del robot y también de los *landmarks* que el robot ha visto en su entorno. Los puntos de referencia son características del entorno que pueden ser fácilmente distinguidas y observadas [López Torres, 2016].

Estas características son utilizadas por el robot para poder saber dónde se encuentra. Hay una serie de requisitos para considerar una característica como un buen *landmark* como se mencionan en [Riisgaard y Blas, 2004].

El problema de la asociación de datos consiste en hacer corresponder las distintas observaciones de un mismo *landmark*. Es decir, ser capaces de identificar cuándo se está observando una referencia ya extraída en una iteración anterior.

Tan pronto el proceso de extracción de características y la asociación de datos finaliza, el proceso de *SLAM* puede considerarse dividido en tres fases:

- Actualizar el estado estimado del robot usando la información dada por la odometría. Conociendo las acciones de control aplicadas al robot y la posición (estimada) anterior, es fácil estimar la posición actual.
- Corregir la estimación tras la observación de los *landmarks*. Usando la estimación de la posición es posible estimar dónde deberían estar ubicados. Usualmente, se encuentran diferencias. Básicamente, es la diferencia entre la posición estimada del robot y la posición real del robot, basada en lo que el robot es capaz de "ver". En este paso, la incertidumbre de cada *landmark* observado es actualizado también para reflejar los cambios recientes.
- Finalmente, comienza nuevamente el ciclo, añadiendo nuevos *landmarks* al mapa del robot. Se utiliza la información actualizada sobre posición e incertidumbre obtenida en este paso.

Entonces el proceso de *SLAM* se puede observar mas concretamente en la figura **2-4**, en esta imagen se muestra un esquema que incorpora la construcción del mapa y el mantenimiento del mismo, en el circuito de localización del robot.

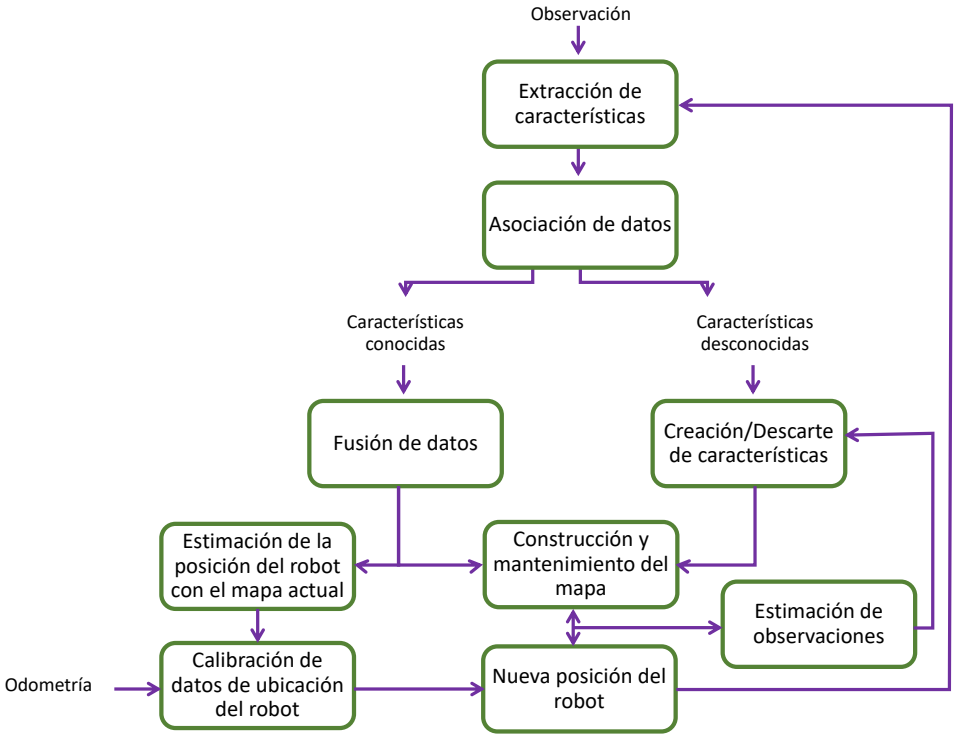


Figura **2-4**.: Esquema de un algoritmo de *SLAM*.

Existen dos grandes enfoques para la solución del problema *SLAM*, Bioinspirados y Probabilísticos. Estos enfoques difieren principalmente en la forma en que procesan la información de entrada de forma de encontrar una buena estimación de la ubicación del robot y mapa del entorno [Andrade y Martin, 2007].

Actualmente, el enfoque probabilístico domina el campo y ha logrado implementaciones que escalan a ambientes grandes y complejos [Milford y Wyeth, 2007]. Sin embargo, la gran capacidad de navegar que poseen mamíferos como las ratas, simios y seres humanos han motivado la investigación y desarrollo de sistemas que imitan los mecanismos biológicos de navegación de estos animales.

Por otro lado, los métodos probabilísticos se concentran en encontrar la distribución de probabilidad de la posición del robot y del mapa del entorno a través del tiempo [Andrade y Martin, 2007].

3. **Interpretación del mapa:** En este paso el algoritmo analiza los datos recolectados por el sensor de visión y los interpreta para construir un mapa que el robot pueda utilizar para localizarse en ese entorno y moverse de forma segura a través de él. La forma en que el robot realiza el proceso de analizar y clasificar la información que se recibe del entorno exterior se menciona de forma detallada en el capítulo 3.

4. **Diseño de trayectorias:** Es el encargado de desarrollar un camino para que el robot sea capaz de ir desde su posición actual hasta una localización deseada evitando colisionar con obstáculos y teniendo en cuenta las limitaciones cinemáticas del robot, es por ello que esta etapa se presenta de manera mas detallada en el capítulo 4.

5. **Cálculo de la trayectoria óptima:** Para poder obtener el mejor camino que el robot puede seguir, es necesario hacer el cálculo mediante algoritmos de búsqueda sobre el mapa del entorno que permitan comparar las opciones generadas y obtener la mejor ruta. Este proceso se menciona de manera mas detallada en el capitulo 5 de este trabajo de tesis.

6. **Control de movimiento:** El control de movimiento es el componente responsable de traducir el camino dado por el componente que calcula el camino que seguirá el robot para alcanzar su objetivo, en señales de entrada para que los motores ejecuten los movimientos necesarios para que el robot siga el camino para que el robot llegue a la meta [Armstrong y Crane, 1998].

   La implementación se realiza en un robot móvil diferencial, el cual funciona haciendo mover cada una de sus ruedas: es decir, en caso de que el robot quiera desplazarse hacia el frente, las ruedas deben girar al mismo tiempo, en el mismo sentido y a la misma velocidad; si se requiere hacer un giro, una de las rudas debe girar mas rápido que la otra o dependiendo el tipo de giro, una rueda puede moverse en dirección opuesta a la otra o incluso, una rueda puede moverse mientras la otra permanece quieta.

   En este paso, el robot se mueve mediante el uso del tópico */move_base* de la librería de ROS, al cual se suscribe el algoritmo que se hizo para esta tarea llamado *autonomousNavigation2*.

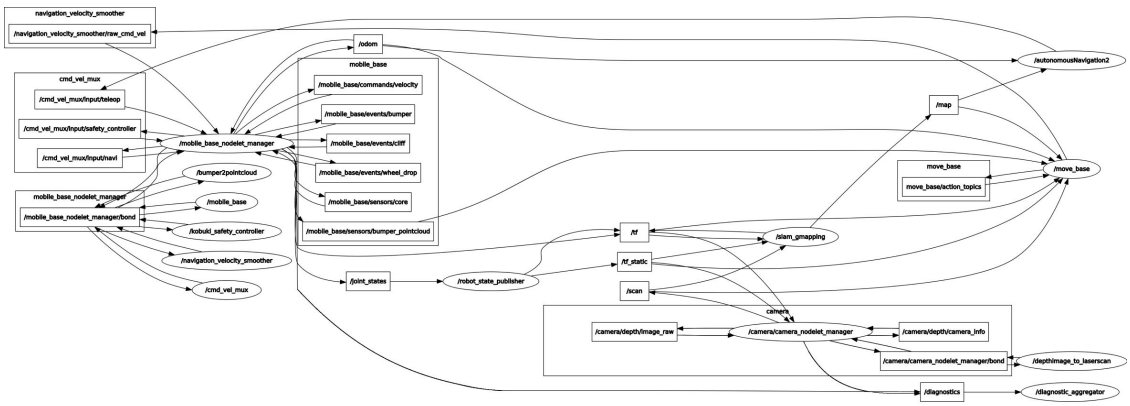


Figura **2-5**.: Esquema de conexión de nodos en *ROS*. (AnexoB)

En la figura **2-5** se muestran los nodos y conexiones en *ROS* con los que se comunica el programa que se ha creado para la tarea de Navegación autónoma.

Para el uso de este tópico se requiere especificar la velocidad lineal y la velocidad angular en cada uno de los ejes: x, y y z; como se sabe, se trata de un robot diferencial, en este caso solo se requiere la velocidad lineal en el eje x y la velocidad angular del eje z, las demás permanecerán con valor 0.

El paso anterior calcula los datos de cada tramo de la trayectoria, puntos de posición y orientación inicial y final que el robot móvil deberá seguir. Con estos datos se pueden obtener las distancias y velocidades que cada rueda debe recorrer por cada tramo para poder seguir la trayectoria generada de manera que el robot llegue a su destino.

Finalmente, se hace el control del movimiento haciendo que el programa use los datos calculados en cada ocasión y los envía a través de mensajes y funciones específicas, creando de esa manera el movimiento que se espera al seguir la ruta elegida.

7. **Condición de paro:** El algoritmo detiene la ejecución cíclica del proceso de mapeado en cuanto se detecta la condición final. En este caso esta condición es la encargada de revisar que el mapa esté completo, es decir, en cuanto ya no se detecten áreas desconocidas en el mapa dentro del espacio cerrado.

   En cuanto el mapa se haya completado el programa terminará ahí haciendo que el robot regrese a su posición inicial, si aún faltan secciones del mapa por completar, el proceso de mapeado seguirá ejecutándose hasta que la condición final lo indique.

   Para realizar la comparación con diferentes algoritmos de este proceso de mapeado, no se implementa la condición de paro al completar el mapa, sino que se hace evaluando qué tan completo queda el mapa generado con respecto al área de implementación física después de cierto numero de iteraciones. Éste proceso se explica de manera detallada en el capítulo 6.

2.2. Plataforma de implementación

La plataforma utilizada para implementar la tarea de navegación autónoma es un robot comercial llamado *TurtleBot2* impulsado por una computadora *Raspberry Pi 3B* que ejecuta *ROS Kinetic*. Una computadora de escritorio también se utiliza para controlar el progreso de la tarea de navegación y para visualizar el mapa obtenido actualmente.

2.2.1. TurtleBot2

El *TurtleBot2* es un robot móvil de cinemática diferencial que está programado con *ROS* y puede utilizarse para múltiples aplicaciones, es una plataforma robótica abierta que está diseñada específicamente para la educación e investigación. Fue creado por la empresa *Willow Garage*, en California, por los ingenieros Melonee Wise y Tully Foote, en noviembre de 2010.

Una de las mayores fortalezas de *TurtleBot* es su comunidad de soporte. *TurtleBot* tiene una comunidad mundial en constante crecimiento con miles de *TurtleBots* en todo el mundo. Además, ha sido adoptado por muchos laboratorios de investigación para realizar investigaciones con múltiples robots e investigaciones de interacción con robots humanos.

Actualmente existen varios trabajos y proyectos que se basan en este robot. Por ejemplo, en [Georgoulas et al., 2012] se utiliza un *TurtleBot* para asistencia y mejoramiento de la calidad de vida de personas mayores mediante un entorno de inteligencia ambiental, ya que se utiliza como asistente del usuario, controlado mediante comandos de voz y puede abordar una gran variedad de servicios para la vida diaria.

En [Quiñonez, Tostado y Burgueño, 2015] también utilizan *TurtleBot* y *ROS* para la aplicación de técnicas evolutivas para la navegación autónoma de robots, utilizan redes de neuronas artificiales y algoritmos evolutivos.

En [Boucher, 2012] utilizan el *TurtleBot* para detectar y evitar de forma autónoma los obstáculos en un ambiente interior libre, el algoritmo es una síntesis del método tradicional de elegir las direcciones de giro en función del centroide de los puntos detectados y una búsqueda más novedosa del plano de tierra para bordes y fronteras obteniendo buenos resultados.

En [Gallardo et al., 2016] utilizan tres robots *TurtleBot* para el control de formación de una colección de vehículos como enjambre de agentes mientras son supervisados por un *drone* manteniendo la formación según la posición del líder.

En [Claessens, Müller y Schnieders, 2013] utilizan un gráfico basado en el algoritmo *GraphSLAM* para explorar y mapear el entorno con un *TurtleBot* y *ROS*.

En este trabajo [Fernández-Vega, 2017] implementan un entorno de programación en MATLAB para el *TurtleBot2*, dotado de un brazo articulado WidowX. El brazo puede mover un objeto en un escenario y además utilizan un algoritmo de navegación para mover todo el conjunto robótico.

Existen muchos otros trabajos que también utilizan esta plataforma robótica, los que se mencionan son solo algunos de los mas recientes encontrados en la literatura, además se observa que esta plataforma abarca diferentes áreas de investigación y de implementación en temas variados, tanto de la vida cotidiana como de colaboración entre agentes.

Como se observa en la figura **2-6**, se trata de un robot móvil diferencial, como se ha mencionado, es usado principalmente en la educación y es de bajo costo. Su velocidad puede alcanzar hasta 50 cm/s y la batería le proporciona una autonomía de hasta siete horas.

Los sensores que lleva la base Kobuki del *TurtleBot* son:

- Parachoques (*Bumper*): Este sensor le indica al robot cuando esta en contacto con algo físico o cuando no. Se convierten en la ultima línea de defensa en los robots para detectar obstáculos. Cuando todos los demás sensores han fallado y no se ha detectado algún obstáculo, el interruptor brinda una indicación muy segura de que se ha producido una colisión y es hora cambiar el rumbo, además se pueden programar para forzar la detención inmediata del robot en caso de que choque.

Figura **2-6**.: Robot móvil *TurtleBot2*

- Sensores de escalera (*Cliff sensor*): Este sensor es capaz de detectar las escaleras y los desniveles que encuentre a su paso para evitar caer, no se despegará de un acantilado con una profundidad superior a 5 cm, esta característica permite que el robot no se dañe en caso de detectar algún desnivel pronunciado al moverse. Existen tres sensores colocados en la parte baja de la base del robot: uno al centro, uno de lado izquierdo y el del lado derecho.
- Sensor de caída de rueda (*Wheel Drop*): Un sensor de caída de rueda, está asociado con cada rueda motriz, y está configurado para detectar cuando una de las ruedas se mueve en una dirección alejada de la posición actual del robot en el plano z, además envía una señal que indica una condición de caída de la rueda motriz, comportándose como un sensor de "baches" para evitar atascamientos de las ruedas. Se tienen dos sensores de caída de rueda debido a la configuración diferencial del robot, uno izquierdo y uno derecho.
- IMU (Unidad de Medición Inercial): Una unidad de medición inercial es un dispositivo electrónico que mide e informa acerca de la velocidad, orientación y fuerzas gravitacionales de un aparato, usando una combinación de acelerómetros y giróscopos. Funciona detectando la actual tasa de aceleración y detecta los cambios en atributos rotacionales. Los datos recolectados por la IMU permiten al computador seguir la posición del robot, usando un método conocido como navegación por estima.
- *Kinect* de Microsoft Corp. Xbox 360: El XBox *Kinect* tiene un proyector de infrarrojos y una cámara de infrarrojos separados por unos 7.5 cm, y una cámara a color a unos 2 cm de distancia de este último [Nguyen, 2012]. El par infrarrojo es capaz de ensamblar una cuadrícula de mediciones de distancia trianguladas del desplazamiento lateral de los puntos proyectados desde el patrón de emisor conocido. El dispositivo no puede

realizar ninguna medida de distancia más cercana a aproximadamente 0.5 m. El robot usa el Microsoft *Kinect* como sensor principal para medir la distancia de los puntos circundantes al robot.

TurtleBot2 es una etapa de tecnología mecánica abierta destinada a la instrucción e investigación en robótica de vanguardia. Además, es un aparato eficaz para mostrar y aprender *ROS* y capitalizar esta innovación de vanguardia. Al estar equipado con un sensor 3D, puede mapear y moverse dentro de un ambiente interior.

Los paquetes disponibles en *ROS* proveen de todo lo necesario para hacer funcionar el robot dándole algunas capacidades como navegación o mapeo, entre muchas otras, y de forma inmediata. Existe un número muy elevado de paquetes *ROS* que pueden utilizarse con el *TurtleBot*, desde librerías como *OpenCV* o PCL hasta drivers de bajo nivel y algoritmos avanzados.

2.2.2. Raspberry Pi 3B

La placa *Raspberry Pi 3B* es una poderosa herramienta que brinda la posibilidad de crear todo tipo de proyectos de electrónica, robótica, mecatrónica y más.

Se trata de un ordenador o una computadora de pequeñas dimensiones, en la que se carga un sistema operativo por medio de una memoria micro SD. Se utiliza también como centro multimedia, centro de información o para fines recreativos.

La *Raspberry Pi 3B* es una popular computadora integrada, que vende más de cuatro millones de tarjetas [Blaza y Tech, 2015] a una comunidad que ha estado creciendo desde 2009.

Cuando se trata de potencia de procesamiento, la *Raspberry Pi 3B* utiliza una ARM Cortex-A53 de 1,2 GHz. Esto permite un procesamiento de código mucho más rápido, que a su vez ayuda con el tiempo de respuesta y la confiabilidad de un robot.

La *Raspberry Pi* fue una de las primeras computadoras integradas en salir al mercado en 2011 y tiene bastantes recursos basados en la comunidad con: videos, libros, blogs y foros, ya que su comunidad de desarrollo es grande y está en continuo crecimiento [Raspberry, 3].

La imagen **2-7** muestra la tarjeta que se utiliza en el proyecto, se observa que se ha montado en un soporte especialmente diseñado para evitar sobrecalentamientos ya que se agrega un ventilador sobre un diseño de tortuga creando el efecto de que se trata de una colaboración de *Raspberry* y *TurtleBot* aunque en realidad se trata de un diseño único.

La placa incorpora los siguientes elementos:

- 4 puertos USB
- 1 puerto Ethernet (rj45)
- Jack HDMI

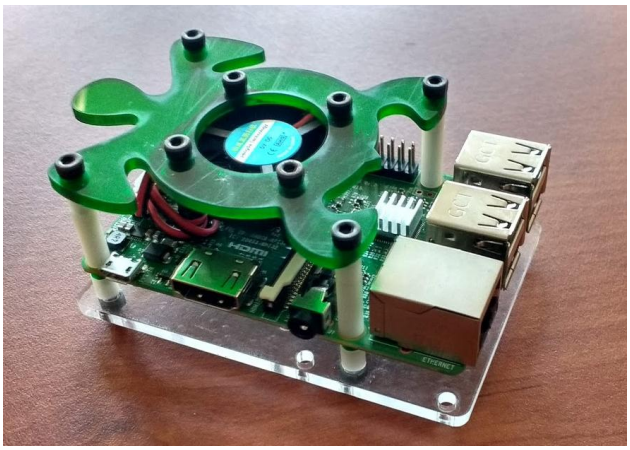


Figura **2-7**.: Placa *Raspberry Pi 3B* usada en este trabajo.

- Jack micro USB (para alimentación)
- Jack 3.5 mm
- Ranura para micro SD
- Conector de 40 pines GPIO
- Conexión Bluetooth
- Conexión *Wi-Fi*
- Interfaz de cámara CSI
- Interfaz de pantalla DSI

A esta tarjeta se le implementó el sistema operativo Ubuntu Mate junto con el middleware robótico *ROS* en su versión *Kinetic* para la implementación de las características que se requieren para este proyecto de tesis.

Inicialmente en este proyecto se utilizaba una computadora portátil o Laptop con características comunes, sobre la cual se cargó el sistema operativo y *ROS*, pero en la práctica resultaba una computadora muy pesada y aunque finalmente se obtenía un buen funcionamiento se decidió cambiar por esta tarjeta, mas pequeña y de fácil acceso, además de ser mas económica y con buenas características para realizar el proyecto.

Se han encontrado varios trabajos en la literatura, que también utilizan esta mini computadora, los cuales no solo motivaron la decisión del cambio de la computadora portátil, sino que también resultaron de gran utilidad para lograr la implementación y correcta instalación para el equipo completo.

Algunos de ellos, por ejemplo, en [Prabha et al., 2014], diseñan un sistema con un robot autónomo para detectar datos ambientales como la temperatura, la humedad y la calidad del aire, para almacenarlas en la nube con una Raspberry y crear conciencia del cambio climático.

En esta investigación [Aagela, Al-Nesf y Holmes, 2017] crean un mapa interior utilizando *SLAM* y además crean modelos 3D para los objetos circundantes con un robot *TurtleBot* además del uso de la *Raspberry* debido a que permite al robot ahorrar más energía en contraste con una computadora portátil normal.

En este trabajo [Vincentelli S. y Hughes, 2016], crean una plataforma de peso ligero para implementar la localización y navegación diseñada para implementarse en vehículos terrestres y aéreos usando la *Raspberry*, los autores implementan pruebas de un pasillo en un *TurtleBot*.

Existen muchos otros trabajos que utilizan esta tarjeta tanto en temas de robótica como en otros de investigación y desarrollo en áreas diferentes, en esta ocasión solo se mencionan algunos para ilustrar el enfoque de este tema de tesis.

2.3. Robot Operating System (ROS)

ROS (Robot Operating System), es un *middleware* robótico de código abierto que proporciona bibliotecas y herramientas para crear aplicaciones robóticas. Proporciona toda una serie de servicios y librerías que simplifican considerablemente la creación de aplicaciones complejas para robots ya que permite el uso de distintos lenguajes de programación como Python y C++.

ROS ofrece los servicios comunes a todos los sistemas operativos, como una capa de abstracción de hardware, control de dispositivos a nivel bajo, intercambio de mensajes entre procesos y manejo de paquetes. Está basado en una estructura con forma de grafo, donde el procesamiento tiene lugar en cada uno de los nodos, que pueden recibir, enviar y manipular mensajes de otros procesos, sensores, actuadores, ordenes de control, de estado y de planificación, entre otros.

ROS está compuesto básicamente de dos partes: el núcleo del sistema *ROS* y los paquetes, bibliotecas y programas contribuidos por los usuarios, agrupados generalmente en pilas o *stacks*.

La información y los procesos de este *middleware* se dividen en las siguientes categorías:

- Nodos: Los nodos son procesos que realizan algún tipo de computación o cálculo en el sistema. Gracias a su estructura granular, el sistema de control de un robot generalmente estará compuesto de diversos nodos individuales comunicándose entre sí, como hilos de ejecución.
- Tópicos (temas): Son canales de información entre los nodos. Es un mensaje enviado. La distribución de mensajes se realiza cuando un nodo publica un mensaje en un tópico. Todos aquellos nodos que deseen recibir dichos mensajes se suscribirán a dicho tópico. De esta manera, la comunicación emula la comunicación en bus, separando la generación de contenido de su consumición.

- Mensajes: Es una estructura de datos y permite la comunicación entre nodos. Los nodos se comunican entre sí pasándose mensajes. Un mensaje es simplemente una estructura de datos, como las de C, que se rellenan apropiadamente.
- Servicios: Todo nodo que puede recibir una conexión de otro nodo. Definen las estructuras de datos para las peticiones y respuestas de aquellos procesos que actúen como servicios en *ROS*. Un servicio es todo nodo que puede recibir una conexión de otro nodo, y que solo actúa como respuesta a una petición de un nodo externo, de manera muy similar al paradigma cliente-servidor en telecomunicaciones. La comunicación con estos se realiza mediante conexiones y el intercambio directo de mensajes.

En **2-8** se muestra la forma en que se comunican los nodos en *ROS*.

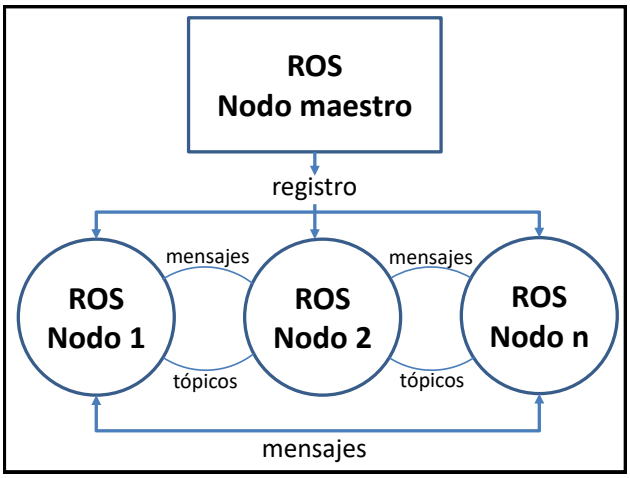


Figura **2-8**.: Comunicaciones de nodos *ROS*

De los múltiples *middlewares* que existen y que están disponibles en la actualidad se eligió *ROS*, por estar al alza, avalado por una gran comunidad de investigadores y desarrolladores, y que dispone de muchas implementaciones de algoritmos, listas para ser usadas en un robot.

En la literatura actual se utiliza mucho este sistema operativo en proyectos de robótica, como en el trabajo de [Ray, Chettri y Thapa, 2018] se presenta una configuración experimental para validar el concepto de control robótico de automóviles basado en *Internet of Things (loRCar)*, útil en diferentes áreas de la ciencia además de mejorar la calidad de vida, que aprovecha la plataforma *ROS* para su implementación.

En [Hulbert, 2016], los autores proponen un método para la navegación autónoma centrándose en la navegación limitada al uso de información visual y la información semántica asociada. Para ello usan *ROS*, además del software *Gazebo* y *TurtleBot*, el reconocimiento de objetos *YOLO (You Only Look Once)* y la extracción de características SIFT para comparar los resultados de navegación entre dos métodos.

En el trabajo de [Ioakeimidis, 2018] se presenta un algoritmo implementado en Python junto con *ROS* y *TurtleBot2* para la navegación autónoma en interiores utilizando LIDAR

para mapear el entorno que rodea al robot y RFID para identificar diferentes objetos alrededor del robot.

En este articulo [Kaczmarek et al., 2018] los autores desarrollan una aplicación que permite la cooperación de robots móviles con robots industriales mediante el protocolo de comunicación TCP/IP. Las aplicaciones se probaron en modo virtual utilizando RobotStudio y *ROS*, además se implementaron usando robots móviles *TurtleBot* e industriales IRB 120.

Cada vez más investigadores de la robótica deciden utilizar esta plataforma debido a su fácil acceso, se puede decir que *ROS* es un gran sistema operativo de código abierto mantenido por la *Open Source Robotics Foundation (OSRF)* y tiene una comunidad enorme y en crecimiento.

Este proyecto comenzó en 2007 por *Willow Garage*, un laboratorio de robótica, para proporcionar a las personas que trabajan con robots un conjunto estándar de configuración y herramientas que les permitieran programar y trabajar mejor y más rápido. *ROS* está ahora detrás de muchos tipos de robots, como *drones*, coches autónomos, robots similares a los humanos, brazos robóticos, entre otros.

La relevancia de *ROS* radica en que cada vez más investigadores, empresas y amantes de la robótica, utilizan este software para programar sus robots, además, *ROS* busca crear una comunidad que comparta sus conocimientos adquiridos.

3. Proceso de creación de mapas.

3.1. Construcción del mapa

Al usar la información del sensor de visión que utiliza el robot, en este caso un *Kinect*, se puede obtener un mapa parcial del área circundante utilizando un programa en *ROS* llamado *gmapping* que a su vez usa la distancia a los objetos y la medición de la odometría para calcular la posición de los objetos en el mapa.

El paquete de navegación en 2D de *ROS* obtiene información de la odometría, de los sensores y de la posición del destino para enviar los comandos de velocidad necesarios a la base del robot. La ventaja de usar este paquete es que bajo las condiciones adecuadas puede conducir al robot hasta su destino sin ningún problema y además evita obstáculos.

Lógicamente para realizar una tarea tan compleja, como es la navegación, es necesario la intervención de varios procedimientos que, actuando en conjunto son capaces de llevar el robot hasta su destino. A continuación se muestra un diagrama del proceso necesario para crear un mapa mediante teleoperación.

En la figura **3-1** se observa este conjunto de procesos que se requieren en *ROS* para conseguir el mapa del entorno en el que se encuentra el robot.

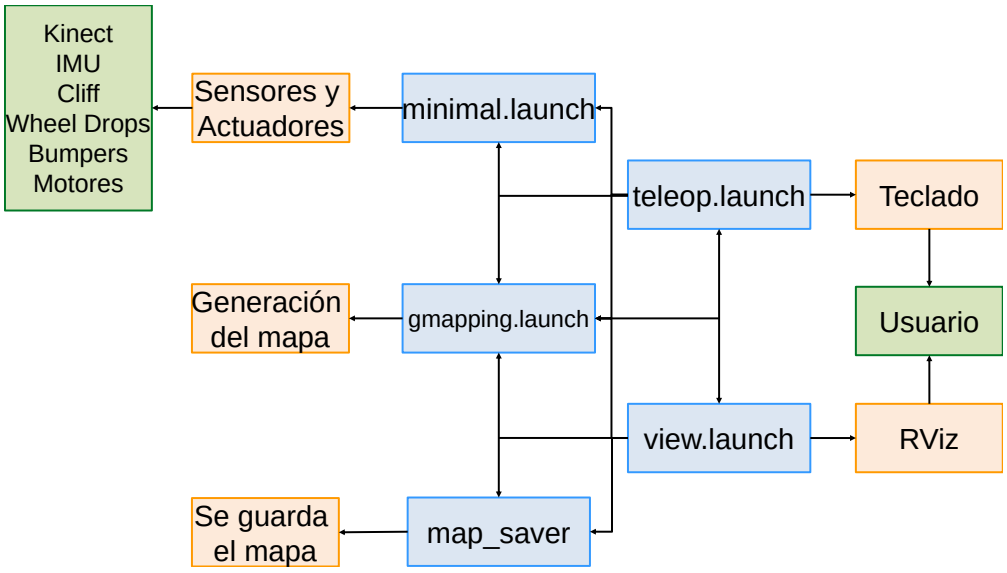


Figura **3-1**.: Esquema del proceso de mapeado.

Como se mencionó en el capítulo 2, una técnica muy utilizada en la robótica para la construcción de mapas en entornos desconocidos es el *SLAM*. La ventaja de esta técnica es que la construcción del mapa se realiza en tiempo real mientras el robot se mueve. Aunque también existen ciertos inconvenientes que podrían hacer que la construcción de este mapa resulte muy complejo, como por ejemplo: la precisión del sensor, ya que es necesario tener una gran precisión de la posición del robot mientras se construye el mapa.

El paquete *gmapping* contiene una implementación del algoritmo de *SLAM* que utiliza un láser o algún sensor similar a través del nodo *slam_gmapping*. Este nodo toma los datos del láser y de la odometría del robot y construye un mapa de cuadrícula de ocupación 2D. Además este mapa es actualizado mientras el robot se mueve. El mapa es publicado en el topic */map* y utiliza el mensaje *nav_msgs/OccupancyGrid*.

La descripción general del algoritmo utilizado para la localización del robot y la generación del mapa al usar el proceso de *gmapping* es la siguiente: En primer lugar se estima la posición y la orientación del robot para saber en dónde se encuentra, para ello se utilizan los sensores IMU; posteriormente se obtiene una nube de puntos que genera el sensor *Kinect* al crear la imagen de los objetos que se encuentran en su rango de alcance; una vez que se realizó este paso se mueve el robot mediante la teleoperación a una posición cercana, se calcula la odometría de la nueva posición para saber cuánto se ha desplazado y en que dirección, así como de igual modo se realiza la nube de puntos de los objetos detectados; finalmente se realiza una concatenación de la nube de puntos que se tenía en un principio con la que se obtiene de la posición actual para generar una imagen nueva. Este proceso se repite hasta obtener el mapa deseado.

Las herramientas del ambiente *ROS* permiten hacer este procedimiento, en la figura **3-2** se muestra la interfaz de *RViz*, que permite observar el proceso de mapeado en tiempo real.

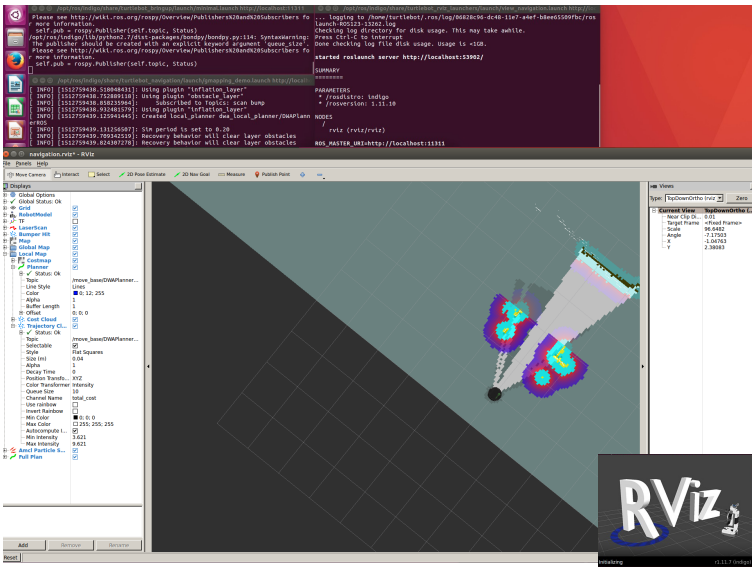


Figura **3-2**.: Visualización gráfica del proceso de mapeado en *RViz*.

A continuación, en la siguiente subsección de este capitulo se describe lo que es una cuadricula de ocupación, en la cual cada uno de los píxeles representan 5 cm. de la dimensión real del espacio, es decir, que un metro medido de manera física en el entorno real, se encuentra representado en una imagen compuesta por 20 píxeles.

3.2. Método de cuadrícula de ocupación

Una cuadrícula de ocupación es una discretización del entorno del robot en celdas. Dicha discretización vendrá dada por el tamaño del entorno en el que se encuentra el robot. Se maneja una matriz cuyas celdas contendrán un valor de probabilidad, el cual indica la certidumbre de que en esa posición se encuentre algún obstáculo o exista espacio libre.

El mapa obtenido por el proceso de *ROS gmapping*, tal como se ha mecionado, se denomina cuadrícula de ocupación (Occupancy Grid Map) [Elfes, 1989], que es un vector que describe la probabilidad de ocupación de cada punto en el mapa. Básicamente este mapa representa la ocupación del entorno que rodea al robot. Divide el entorno en celdas, donde cada una de ellas esta representanda por un rango de $(0, 100)$. Siendo 0 completamente libre y 100 completamente ocupada. Existe un valor especial -1 para indicar que no se conoce nada de esa celda, como un punto desconocido.

Esta información se obtuvo del tópico de *gmapping.launch* como vector de datos al que es posible acceder para observar los valores de la cuadrícula de ocupación, en la figura **3-3** se muestra la forma en que se pueden visualizar los datos del vector que el tópico */map* contiene, se observa la interfaz de *RViz* y la terminal con los rangos de las celdas.

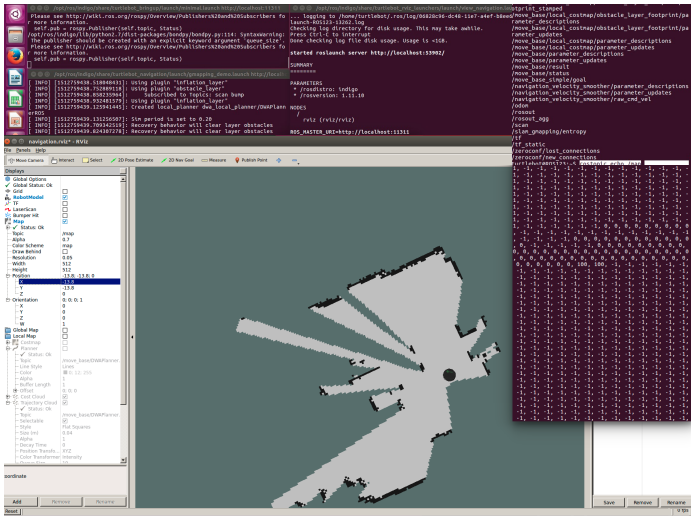


Figura **3-3**.: Obtención de valores de las celdas de ocupación en *RViz*.

La imagen final del mapa representa los espacios libres en blanco y los espacios ocupados en tonos grises detectados por la aplicación. El algoritmo propuesto crea una imagen en

escala de grises basada en la información de la cuadrícula de ocupación. Los obstáculos O se dibujan en negro ($g = 0$), el área libre F es dibujada totalmente en blanco con ($g = 255$) y el área desconocida U se dibuja en un tono de gris intermedio ($g = 128$) tal como se ilustra en la figura **3-4**.

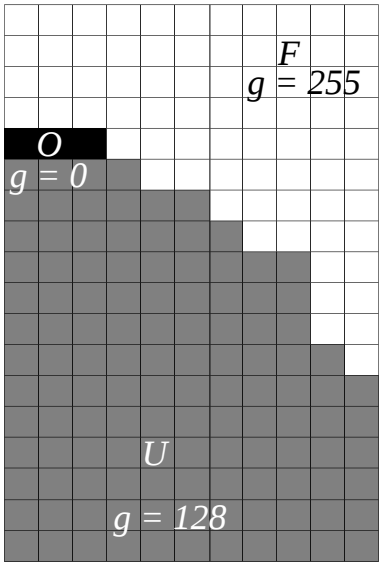


Figura **3-4**.: Imagen creada a partir de la cuadrícula de ocupación.

3.3. Interpretación del mapa

3.3.1. Notación de conjuntos

A partir de esta sección, se hará uso de la notación de conjuntos para describir las regiones del mapa actual y para formular la función objetivo.

El conjunto de todos los puntos que forman parte del mapa se llama M. En robótica, un mapa suele ser una descripción de un área que clasifica cada posición $m \in M$ en tres conjuntos: el espacio libre F, los obstáculos O y el área desconocida U. Estas definiciones implican que $M = F \cup O \cup U$.

La tarea asignada al robot es explorar un área interior cerrada para construir un mapa completo. Un mapa se completa cuando todos los puntos que puede alcanzar el robot pertenecen al espacio libre F y todos los puntos con los que el robot puede colisionar pertenecen a O, esto quiere decir que el mapa esta completo cuando ya no existen puntos que pertenezcan al área desconocida U.

El proceso de mapeo es iterativo, el robot debe moverse a diferentes áreas para clasificar los puntos usando un sensor de distancia o equivalente. Es importante mencionar que el robot no puede clasificar puntos detrás de un obstáculo a menos que se mueva a su alrededor.

Teniendo en cuenta el proceso de mapeo, la trayectoria óptima será la que impulse al robot sobre un conjunto de posiciones P, lo que le permite al robot mapear áreas desconocidas.

Con base en la declaración anterior, uno podría tener la tentación de definir el problema de optimización como la minimización de la distancia entre la posición actual del robot y las áreas desconocidas, sin embargo, no todos los puntos del mapa se pueden mapear, por ejemplo, todos los puntos detrás de una pared no pueden ser alcanzados, así que no pueden ser mapeados.

Se debe definir un nuevo conjunto para describir los puntos en el área desconocida U que si se pueden mapear. Estos puntos son los que están en el borde entre F y U, denotados por B, considerando la definición en la ecuación 3-1.

$$B = \{u \in U \mid (\exists f \in F)[\, \|f - u\| \leq 1]\} \tag{3-1}$$

Como cada punto en M es un vector con coordenadas (x, y), el operador $\|\cdot\|$ es usado para representar la distancia entre dos puntos mediante la norma euclidiana.

Se propone que la aptitud A_ξ de una trayectoria definida por ξ sea medida por el número de puntos en la frontera B que se encuentran en las proximidades de los puntos en P.

El entorno establecido Λ_k se define como todos los puntos en M cuya distancia hasta cierto punto en P es menor o igual a λ_k, que define la distancia máxima permitida. Este conjunto se define considerando la ecuación 3-2 de la siguiente manera:

$$\Lambda_k = \{m \in M \mid (\exists p \in P)[\, \|p - m\| \leq \lambda_k]\} \tag{3-2}$$

Una descripción gráfica de todos los conjuntos definidos en esta sección se muestra en la figura **3-5**.

3.3.2. Formulación del problema de optimización

Teniendo en cuenta todas las definiciones anteriores, el problema se formula como una maximización de la aptitud A de una trayectoria, calculada como el número de elementos $\mathrm{n}(\cdot)$, en B que también pertenecen a la diferencia $\Lambda_2 \setminus \Lambda_1$. La función de aptitud o costo se describe mediante la siguiente ecuación:

$$A = \mathrm{n}(\{b \mid b \in B \cap (\Lambda_2 \setminus \Lambda_1)\}) \tag{3-3}$$

Como se muestra en la figura **3-5** consideramos $\lambda_2 > \lambda_1$, los puntos en el conjunto Λ_1 están excluidos del cálculo de la función de costo porque las áreas desconocidas pueden contener obstáculos que aún no se han mapeado y acercarse demasiado a la zona desconocida puede llevar al robot a una colisión.

Existen algunas restricciones que una trayectoria factible P debe mantener por razones de seguridad. Se definen tres funciones para garantizar que el robot no colisionará con los obstáculos.

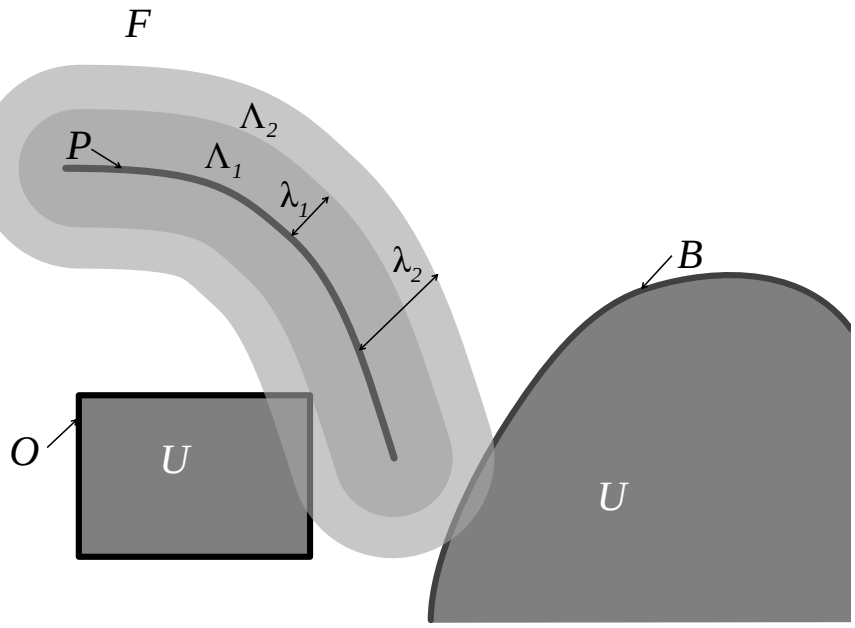


Figura **3-5**.: Establecer definiciones dentro del mapa.

Es importante que todos los puntos en una trayectoria factible P se encuentren ubicados dentro del espacio libre F, esto significa que $P \subset F$. Por lo tanto, el nivel de transgresión a esta restricción R_1 se calcula como el número de elementos en P que no están en F:

$$R_1 = \mathrm{n}(\{p \mid p \in P \setminus F\}) \tag{3-4}$$

También es importante que los elementos en la trayectoria P estén lo suficientemente lejos de los obstáculos para evitar colisiones, la distancia mínima es λ_1. Por lo tanto, el nivel de transgresión a esta restricción R_2 se calcula como el número de elementos en O que también están en Λ_1:

$$R_2 = \mathrm{n}(\{o \mid o \in O \cap \Lambda_1\}) \tag{3-5}$$

Es inconveniente que el robot navegue demasiado cerca de las fronteras en B ya que podría encontrar obstáculos que todavía no se han mapeado. Por lo tanto, el nivel de transgresión a esta restricción R_3 se calcula como el número de elementos en B que también están en Λ_1:

$$R_3 = \mathrm{n}(\{b \mid b \in B \cap \Lambda_1\}) \tag{3-6}$$

Es posible ver que la función de costo en la ecuación 3-3, así como las funciones de restricción en las ecuaciones 3-4, 3-5 y 3-6 dependen de la ruta P que a su vez depende del vector de parámetro ξ por lo que finalmente el problema de optimización se puede formular

de la siguiente manera:

$$\begin{aligned} \underset{\xi}{\text{Maximize}} \quad & A(\xi) \\ \text{sujeto a:} & \\ & R_1(\xi) \leq 0 \\ & R_2(\xi) \leq 0 \\ & R_3(\xi) \leq b_m \end{aligned} \tag{3-7}$$

Tal que el vector de parámetros ξ pertenece al espacio de búsqueda definido por la ecuación 4-3. El valor b_m define el valor límite superior permitido para la restricción R_3.

3.4. Algoritmos de procesamiento de imágenes usados para analizar el mapa

3.4.1. Calculo de bordes.

Un borde es el límite entre dos regiones con niveles de gris distintos, de manera que la transición entre dichas regiones se pueda observar como un cambio abrupto. La idea básica para la detección de bordes es el cálculo de una derivada, teniendo en cuenta que para una constante el valor es cero y para algún cambio será diferente de cero [Duque y Ospina, 2004].

El Laplaciano es un operador escalar de segunda derivada y para funciones en dos dimensiones se define mediante la ecuación 3-8.

$$\nabla^2 f(x,y) = \frac{\partial^2}{\partial x^2} f(x,y) + \frac{\partial^2}{\partial y^2} f(x,y) \tag{3-8}$$

La ecuación se implementa en forma digital haciendo la convolución de una imagen con una región o máscara de 3x3 como se observa en la imagen **3-6**.

z_1	z_2	z_3
z_4	z_5	z_6
z_7	z_8	z_9

Figura **3-6**.: Región de imagen de 3x3.

El principio para la definición de este operador es que el coeficiente asociado al píxel central sea positivo y los coeficientes asociados a los píxeles exteriores sean negativos, tal que la suma de todos los coeficientes sea cero.

Al hacer la convolución de alguna de las máscaras de la figura **3-7** con la imagen del mapa, el resultado será cero cuando que el punto central tenga el mismo valor que sus vecinos [González y Woods, 1996; Kenneth, 1979].

0	-1	0
-1	4	-1
0	-1	0

-1	-1	-1
-1	8	-1
-1	-1	-1

Figura **3-7**.: Operadores Laplacianos.

Para el caso particular de una imagen como la que se genera a partir de la información del mapa, se eligió el operador Laplaciano con elemento central $z_5 = 8$, definido en la ecuación 3-9. Es claro que si el valor del píxel central es diferente de alguno de sus 8 vecinos se detecta un borde.

$$\nabla^2 f = 8z_5 - (z_1 + z_2 + z_3 + z_4 + z_6 + z_7 + z_8 + z_9) \tag{3-9}$$

El conjunto de bordes B se calcula al encontrar todos los píxeles en U que están al lado de un píxel en F usando un criterio de conectividad de ocho direcciones. La figura **3-8** muestra el proceso para encontrar los píxeles en B, estos puntos se dibujan en gris oscuro ($g = 90$).

El proceso para realizar el cálculo de los bordes es el siguiente: En primer lugar se recorre la imagen del mapa buscando un píxel gris, es decir, si es blanco o negro se omite el cálculo. Cuando se encuentra ese píxel, se busca en los 8 vecinos por lo menos un píxel blanco, el cual indica que hay un borde, entonces ese píxel se redibuja con un tono de gris mas oscuro. En caso de que los 8 vecinos sean grises no existe un borde y se continúa el proceso.

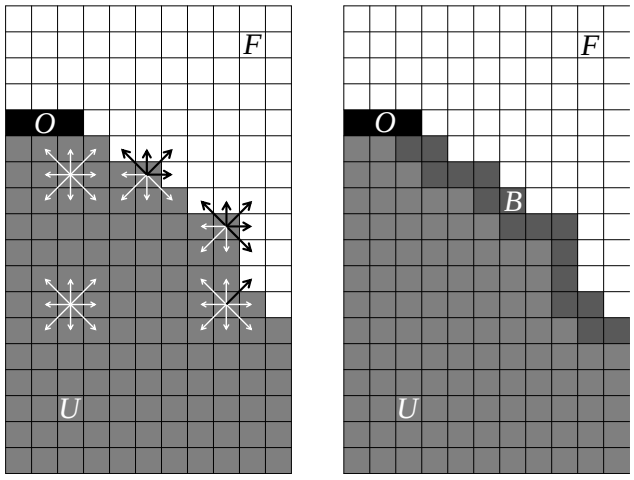


Figura **3-8**.: Criterio de conectividad para definir el conjunto de bordes B.

En la figura **3-8** se muestra la conectividad de píxeles en U a píxeles en F con flechas negras en la imagen de la izquierda, los píxeles encontrados en B se muestran en un tono de gris más oscuro en la imagen de la derecha.

3.4.2. Algoritmo de Dilatación

La morfología matemática fundamenta todas sus operaciones en dos operadores: la erosión y la dilatación asociadas con un elemento estructurante o máscara. El elemento estructurante es un objeto de referencia de dimensiones pequeñas, por lo regular se consideran algunos píxeles, además de que presenta una estructura geométrica sencilla, como un punto, una línea, un cuadrado o alguna otra, y está asociado a la estructura o topología predominante en los elementos presentes en la imagen [Torres y Bello, 2002].

La dilatación en las imágenes en niveles de gris se obtiene al desplazar el elemento estructurante, sobre la imagen y reemplazar el píxel central por el máximo de los niveles digitales cubiertos por el mismo elemento estructurante [Soille, 2013]. La dilatación permite destacar las zonas claras presentes en la imagen. La expresión que define la dilatación de la imagen f por un elemento estructurante B se presenta en la ecuación 3-10.

$$\delta_B(f)(x) = \max_{b \in B} \quad f(x+b) \tag{3-10}$$

La vecindad establece Λ_1 y Λ_2, se calculan utilizando el algoritmo de *dilatación*, este proceso está destinado a ser utilizado en una imagen binaria, por lo que se crea una nueva imagen del mapa, donde los elementos de P se dibujan con ($g = 255$) y todos los demás píxeles se dibujan en negro ($g = 0$), la nueva imagen tiene las mismas dimensiones que el mapa original.

El algoritmo se implementa reemplazando el valor de cada píxel con el valor máximo en una máscara centrada en cada píxel, el tamaño de la máscara λ define el grado de dilatación.

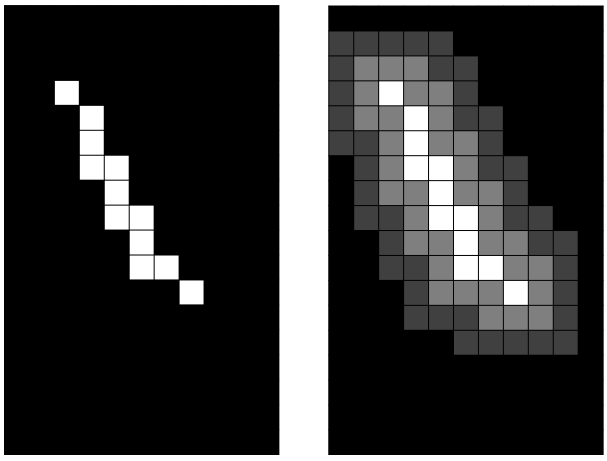


Figura **3-9**.: Proceso de dilatación utilizado para encontrar conjuntos de proximidad Λ

La figura **3-9** muestra el proceso de dilatación, la imagen de la izquierda es la original; los píxeles agregados mediante el proceso de dilatación se muestran en gris en la imagen de la derecha, usando $\lambda = 1$ y $\lambda = 2$, los valores de escala de grises son solo para fines de demostración, los puntos agregados para establecer Λ y los puntos originales en P tendrán el mismo valor.

El mapa discreto representado por una imagen obtenida mediante el uso de los procesos descritos en esta sección se muestra en la figura **3-10** como la versión discreta del mapa que se muestra en la figura **3-5**.

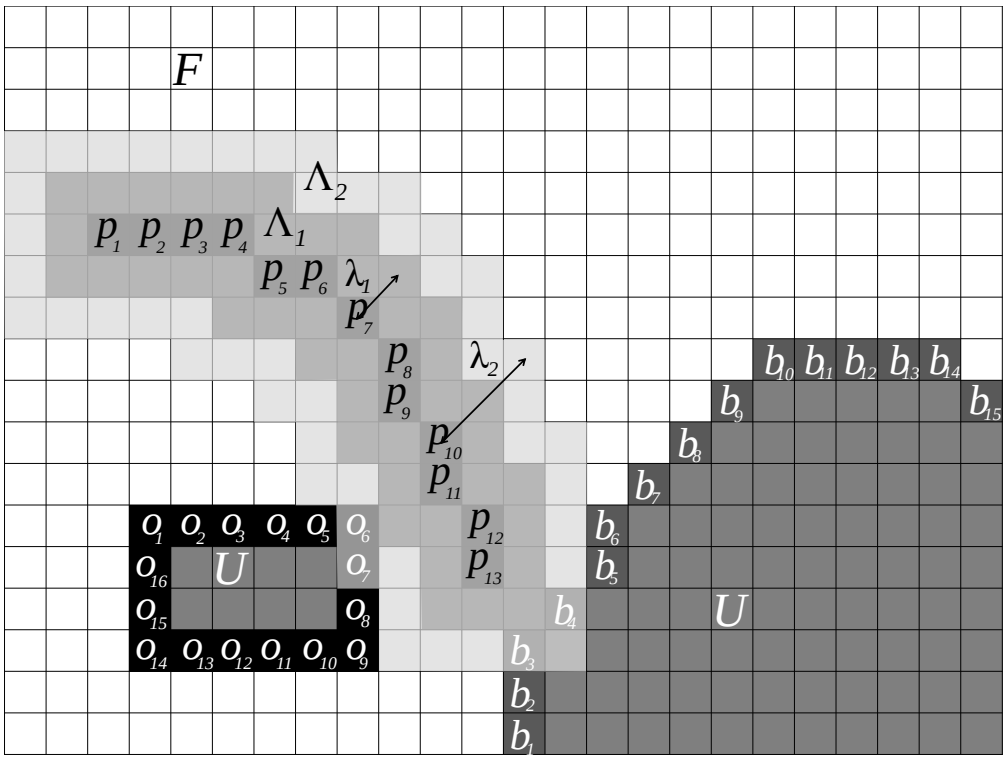


Figura **3-10**.: Mapa discreto.

En los conjuntos de implementación reales P y Λ_k se guardan en diferentes imágenes aparte del mapa original, la razón es que cada píxel en M pertenece a solo uno de los conjuntos F, O, U y B, esto significa que estos conjuntos son disjuntos, por lo que es posible describir la pertenencia a cualquiera de estos conjuntos por una escala de grises diferente en la misma imagen sin ambigüedad, sin embargo, cualquier posición en M también puede ser en P o Λ_k, por lo que no es posible dibujar estos conjuntos en la misma imagen. La figura **4-5** y **3-10** muestran todos los conjuntos en la misma imagen solo con fines de demostración.

Teniendo en cuenta las imágenes definidas en esta sección, es posible evaluar la pertenencia a un elemento en M comparando los valores de escala de grises de los píxeles en las imágenes correspondientes. Al usar este método es posible evaluar las expresiones en la formulación del problema de la ecuación 3-7.

4. Diseño de trayectorias.

4.1. Generadores de Trayectorias

En la actualidad se han desarrollado diversos métodos para lograr que un robot se mueva en un entorno para lograr variados propósitos como evadir obstáculos, navegar a un sitio específico o hacer un mapa interno de un entorno.

De forma genérica, el problema de planificación de movimientos consiste en llevar un cuerpo, desde una configuración inicial hasta otra final dentro del espacio de configuraciones libres de colisión [López et al., 2010].

Este problema ha sido ampliamente abordado en la literatura [Barraquand y Latombe, 1991; Laumond et al., 1994; Muñoz, 1995], existiendo un gran numero de métodos efectivos para resolver el problema de la planificación en tiempo real, tales como campos potenciales, grafos de visibilidad, diagramas de Voronoi, entre otros.

Los métodos clásicos de planificación se fundamentan en una representación del espacio libre [Acevedo y Castañeda, 2007], según información obtenida del entorno previamente, para posteriormente emplear un algoritmo de búsqueda que encuentre el camino óptimo según cierta función de costo.

Una primera y amplia clasificación de las técnicas de planificación se basa en la información utilizada a la hora de realizar la planificación. De esta forma, se pueden clasificar en técnicas locales o globales [García, 2004].

Las técnicas locales tan sólo tienen en cuenta una porción del espacio total en el que se realiza el movimiento. Así, se realiza el movimiento de una configuración a la siguiente de la ruta teniendo en cuenta las características de un reducido entorno a su alrededor. Estas técnicas tienen la ventaja de que tienen una reducida carga computacional, pero tienen grandes desventajas como:

- En numerosos casos no se alcanza una solución óptima.
- Pueden dar lugar a oscilaciones en pasillos estrechos.
- En algunos casos no se encuentra la solución al llegar a un lugar del que estas técnicas no permiten salir, como puede ser el caso de la aparición de mínimos locales.

Por otra parte, las técnicas globales tratan de encontrar la ruta completa teniendo en cuenta la conectividad de todo el espacio libre. Al considerar todo el espacio libre la complejidad computacional es muy elevada con la consiguiente lentitud. Pero sin embargo, tiene la ventaja de que se puede conseguir una ruta óptima.

Las heurísticas son procedimientos simples que realizan una exploración limitada del espacio de búsqueda y dan soluciones de calidad aceptable en tiempos de cálculo generalmente moderados. Las soluciones obtenidas con esta clase de procedimientos pueden, en general, ser mejoradas utilizando métodos de búsqueda más sofisticados, pero incurriendo en elevados tiempos de ejecución.

Para obtener mejores soluciones que las heurísticas, es necesario recurrir a técnicas que realicen una mejor exploración del espacio de soluciones. Las metaheurísticas son procedimientos genéticos de exploración del espacio de soluciones para problemas de optimización y búsqueda. Proporcionan una línea de diseño que, adaptada en cada contexto, permite generar algoritmos de solución.

En general, las metaheurísticas obtienen mejores resultados que las heurísticas clásicas, pero incurriendo en mayores tiempos de ejecución, aunque de todos modos son inferiores a los de los algoritmos exactos.

Los Algoritmos de Hormigas son procedimientos basados en agentes que utilizan métodos constructivos aleatorizados y cooperan entre sí compartiendo información [Olivera, 2004]. Los algoritmos de Búsqueda Tabú son métodos de búsqueda local que aceptan empeorar las soluciones para escapar de los óptimos locales. Los Algoritmos Genéticos se basan en mantener un conjunto de soluciones lo suficientemente diverso como para cubrir gran parte del espacio de soluciones.

Otro muy conocido método de planificación es el de campo de potenciales, el cual también es utilizado en aplicaciones de control reactivo. Este método, pese a ser rápido, presenta el problema de la existencia de mínimos locales. Para solucionar esta limitación se recurre a algoritmos de generación aleatoria que permiten realizar planificaciones locales hacia un nuevo mínimo [Barraquand y Latombe, 1991].

A continuación se presentan algunos de estos métodos explicados de manera mas detallada, la selección de estos métodos es debido a que se acercan lo mas posible al método aquí propuesto. Uno de ellos es el de Campos potenciales, el de Rapidly-Exploring Random Trees (RRT) y el de Curvas de Bézier.

4.1.1. Campos potenciales

El método de campo de potencial es conocido como un método local, ya que sólo calcula el movimiento para una configuración inicial-final dada. Se comienza en la inicial y se intenta mover el robot hacia el objetivo en pequeños pasos. La introducción del concepto del campo de potencial en la planificación de movimientos se debe a Khatib [Khatib, 1986].

La interpretación física de los campos de potenciales puede entenderse como si existieran una serie de fuerzas, atractivas hacia la configuración final y repulsivas desde los obstáculos que generan estos potenciales y que guían al robot por la región libre hacia la configuración final. Estas fuerzas serían el gradiente negado de un potencial, por lo que se puede pensar en una superposición de los potenciales asociados a estas fuerzas.

Como un ejemplo se puede imaginar que se tiene un espacio de configuración donde los objetos están definidos por polígonos irregulares, al dibujar esos polígonos en una manta y al hacer que se levanten, poniendo objetos por debajo de estos polígonos, formando una superficie con altas (objetos) y bajas (trayectorias libres de colisión o la meta).

Continuando con el ejemplo, ahora se define un punto inicial y un punto final sobre la manta. Posteriormente se levanta la manta lo necesario para que el punto inicial sea el más alto y el punto final sea el más bajo. Entonces, se coloca un balín en el punto inicial y en este caso es posible observar como el balín es atraído al punto más bajo (meta), evitando los obstáculos. De esta manera la manta funciona como la fuerza repulsiva que mantiene al balín lejos del los obstáculos y al mismo tiempo guía al robot al punto final [Gonzáles y Parkin, 2005].

En la imagen de la figura **4-1** se observa la representación de un campo potencial artificial.

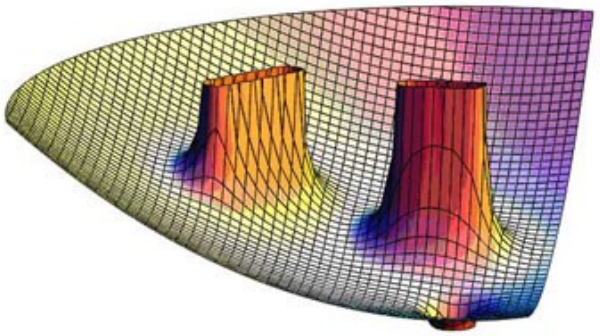


Figura **4-1**.: Ejemplo de un campo potencial artificial representado por dos obstáculos y la meta [Gonzáles y Parkin, 2005].

Una posibilidad sería que el potencial repulsivo fuera igual al inverso de la mínima distancia a los obstáculos y el atractivo fuera igual al cuadrado de la distancia a la configuración final. Sin embargo, se pueden definir numerosos tipos de potenciales para hacer la planificación, pero se presenta el problema de que según se sigue el sentido opuesto del gradiente se llegue a un punto en el que se alcance un mínimo local del potencial del que fuera difícil escapar [García, 2004].

Uno de los primeros trabajos presentados sobre el uso de campos de potenciales para la planificación de caminos fue [Hwang y Ahuja, 1989], donde se realiza un recorrido sobre las posibilidades que ofrece este método.

Este problema se puede resolver si se utilizan unos potenciales que no presenten ningún mínimo local salvo el final [Koditschek, 1987] o si se utilizan algunos mecanismos muy potentes para escapar de los mínimos locales una vez que se han alcanzado [Barraquand y Latombe, 1991].

La figura **4-2** muestra el campo potencial con un obstáculo tipo barrera, que presenta un mínimo local.

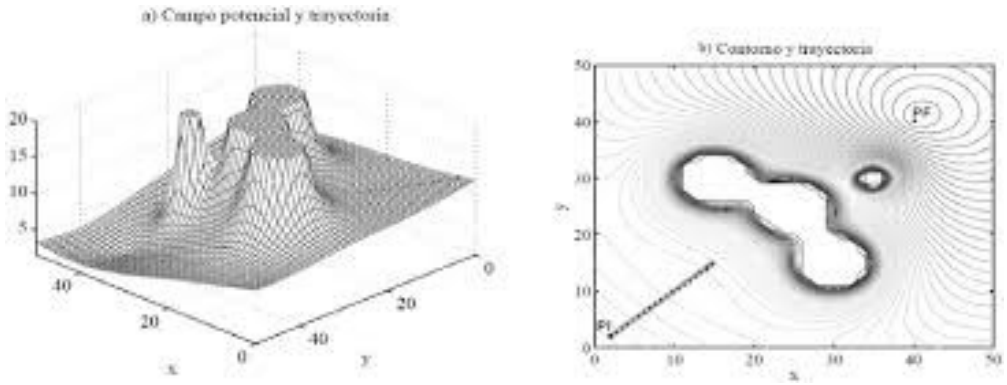


Figura **4-2**.: a) Campo potencial con la presencia de una barrera, b) Contorno del campo potencial con la presencia de una barrera. [Cuchango y Esmeral, 2012].

Gracias al desarrollo del método de campos potenciales surgió una primera aproximación a los métodos de planificación aleatoria [Latombe, 1991]. En este tipo de métodos, se parte de un escenario donde el espacio de configuraciones se encuentra dividido en porciones discretas estableciendo una rejilla regular.

4.1.2. Rapidly-Exploring Random Trees (RRT)

El árbol aleatorio de exploración rápida (RRT) no precisa el establecimiento de un campo de potencial, con el consecuente ahorro del procesamiento. Al mismo tiempo, el RRT asegura una exploración equiprobable de todo el espacio de configuraciones. Por último, es sencillo, rápido y de fácil extensión a escenarios complejos [López et al., 2010].

El algoritmo se basa en la construcción de un árbol de configuraciones que crece buscando a partir de un punto origen. El objetivo consistía en construir un árbol de exploración que cubriera uniformemente todo el espacio libre de colisión. Para ello [LaValle, 1998], desarrolló un algoritmo que tiene como misión seleccionar un punto de forma aleatoria y extender hacia el árbol de configuraciones.

La figura **4-3** muestra una visualización de un gráfico RRT después de varias iteraciones.

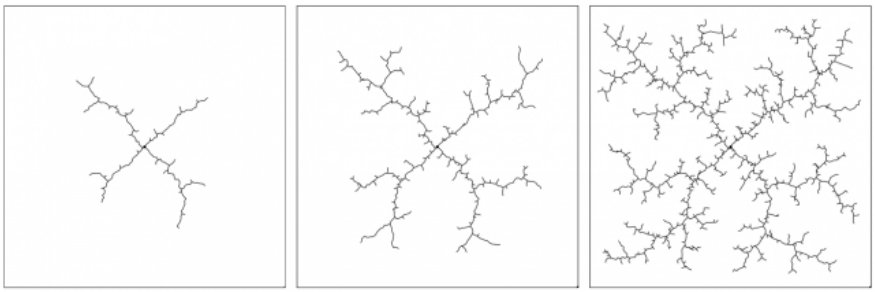


Figura **4-3**.: Construcción incremental de un árbol aleatorio de exploración rápida (RRT) [LaValle, 1998].

El comportamiento de este algoritmo con respecto a otros es mejor en cuanto a la homogeneidad del espacio explorado. La naturaleza del RRT le hace avanzar con más avidez en aquellas zonas donde haya un mayor espacio libre, pues es allí donde hay más posibilidad de encontrar puntos factibles.

Los árboles obtenidos en la aplicación de los distintos algoritmos RRT pueden resultar complejos para ser recorridos. Normalmente, los arboles son susceptibles de simplificación (por ejemplo, en algunos casos los extremos de ambos arboles podrían unirse con una simple línea recta). Por consiguiente, en las aplicaciones prácticas del método RRT resulta conveniente aplicar a los arboles obtenidos un postproceso que permita reducir su irregular topología [López et al., 2010].

En este sentido, al igual que ocurre en otras técnicas de planificación, se suelen aplicar algoritmos de simplificación, muy rápidos y sencillos, que parten de la información suministrada por el planificador y generan, de forma iterativa, un camino más simple.

4.1.3. Curvas de Bézier

Existen varias tareas de control ampliamente utilizadas para controlar a los robots de manera que se desplacen de un lugar a otro ejecutando movimientos predeterminados o deseados, por ejemplo: regulación, seguimiento de trayectorias y evasión de obstáculos.

El seguimiento de trayectorias es el tema más estudiado debido a su importancia práctica. La solución a dicha tarea conlleva a la generación de la trayectoria deseada y la aplicación del control para el seguimiento de la trayectoria [Sanchez et al., 2016]. Si la trayectoria generada no es la apropiada el robot puede deslizar o derrapar. Esto ocasionaría errores en la postura del móvil y conllevaría, a su vez, errores en el seguimiento de la trayectoria.

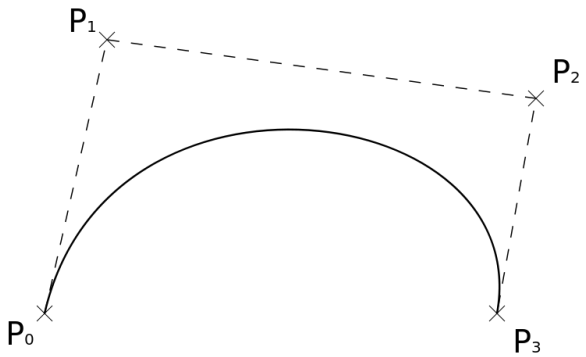


Figura **4-4**.: Ejemplo de una curva de Bézier cúbica [Park y Ravani, 1995].

Las curvas de Bézier son paramétricas y diseñadas para unir suavemente un grupo de puntos, una vez que se dispone de los puntos por los que el robot debe pasar. Estas curvas pueden ser de diversos grados, dependiendo del nivel de suavizado que se desee. El requerimiento mínimo en planificación de trayectorias es pasar por el punto de inicio y por el final

con una dirección especifica. Así, el grado mínimo de la curva de Bézier que satisface esta condición es de tres [Correa Farias y Vásquez Alfaro, 2012].

Aunque, si bien, los polinomios de Bézier son uno de los métodos más empleados para la unión de puntos o segmentos, también lo son para la generación de trayectorias.

La popularidad de las curvas de Bézier se debe a sus numerosas propiedades matemáticas que facilitan su manipulación y análisis. Además, no se requieren grandes conocimientos matemáticos para utilizarlas [Pérez, 2012].

4.2. Propuesta de generador

Una vez que se analizaron algunas de las técnicas de generación de trayectorias que existen en la actualidad se decidió realizar una propuesta basada en los métodos mencionados en la sección anterior.

Se propone una ruta para el robot generada de manera aleatoria, definida por ecuaciones polinómicas simples como alternativa al uso de curvas de Bézier, a continuación se menciona de manera mas detallada como es que se emplearon estas funciones para este caso.

4.2.1. Parámetros de la ruta del robot

La ruta del robot P se describe mediante una función paramétrica polinómica de quinto grado definida por las siguientes ecuaciones:

$$\begin{aligned} x(t) &= a_0^x + a_1^x t + a_2^x t^2 + a_3^x t^3 + a_4^x t^4 + a_5^x t^5 \\ y(t) &= a_0^y + a_1^y t + a_2^y t^2 + a_3^y t^3 + a_4^y t^4 + a_5^y t^5 \end{aligned} \tag{4-1}$$

Los seis parámetros de las ecuaciones 4-1 se calculan mediante las siguientes seis condiciones: las trayectorias comienzan y terminan en reposo para que el robot se mueva continuamente, la posición inicial q_0 y la orientación se calculan mediante la posición actual del robot, el punto intermedio q_1 y el punto final q_2 están definidos por un ángulo θ_k y una distancia r_k con respecto al punto inicial como se muestra en la figura **4-5**.

Entonces, una vez que se conocen los parámetros necesarios para la solución de estas ecuaciones, la descripción completa de la estructura del polinomio 4-2 para la generación de trayectorias quedaría de la siguiente forma:

$$\begin{aligned} x &= [x_1 \quad x_2 \quad x_3 \quad x_4], \quad y = [y_1 \quad y_2 \quad y_3 \quad y_4] \\ t &= [0 \quad t_{inc}/20 \quad t_{inc} \quad 2t_{inc}] \end{aligned}$$

$$\begin{aligned} x(t) &= a_0^x + a_1^x t + a_2^x t^2 + a_3^x t^3 + a_4^x t^4 + a_5^x t^5 \\ y(t) &= a_0^y + a_1^y t + a_2^y t^2 + a_3^y t^3 + a_4^y t^4 + a_5^y t^5 \\ \dot{x}(t) &= a_1^x + 2a_2^x t + 3a_3^x t^2 + 4a_4^x t^3 + 5a_5^x t^4 \\ \dot{y}(t) &= a_1^y + 2a_2^y t + 3a_3^y t^2 + 4a_4^y t^3 + 5a_5^y t^4 \end{aligned}$$

$$\dot{x}_1 = 0, \quad \dot{y}_1 = 0$$
$$\dot{x}_4 = 0, \quad \dot{y}_4 = 0$$

$$x_2 = \cos\theta[(0{,}1 \cdot |q_1 - q_0|) + x_1]$$
$$y_2 = \sin\theta[(0{,}1 \cdot |q_1 - q_0|) + y_1] \tag{4-2}$$

Se debe mencionar que la componente de x_2 se toma como la orientación inicial del robot y se calcula mediante el uso de la norma euclidiana ya que se trata de puntos con coordenadas, para lo cual se tiene lo siguiente: $|q_1 - q_0| = \sqrt{(x_3 - x_1)^2 + (y_3 - y_1)^2}$, debido a las características de este polinomio este punto solo se toma en cuenta a la hora de generar la trayectoria como un apoyo para darle la orientación correcta al robot.

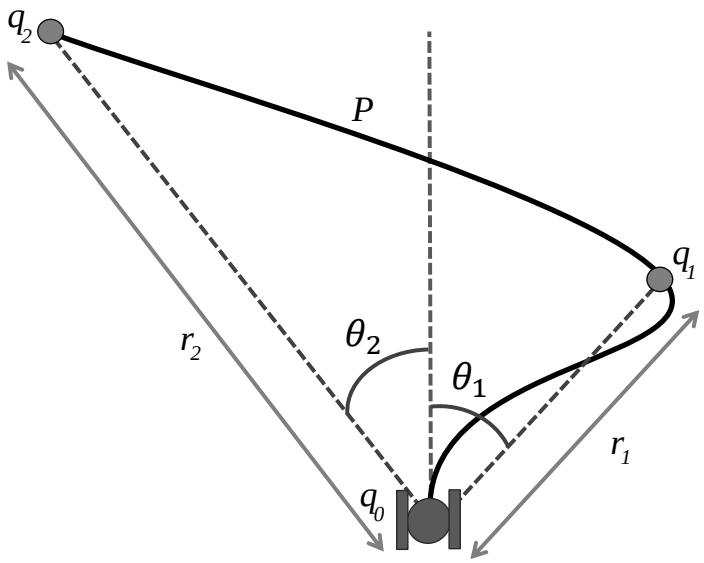


Figura **4-5**.: Parámetros de la trayectoria.

Las primeras cuatro condiciones para calcular la ruta del robot son fijas, de modo que el vector de variables de decisión se calcula mediante los parámetros utilizados para definir la posición de los puntos q_1 y q_2:

$$\xi = \begin{bmatrix} \theta_1 \\ \theta_2 \\ r_1 \\ r_2 \end{bmatrix} \tag{4-3}$$

El espacio de búsqueda de un problema de optimización es el dominio de la función a optimizar, en este caso el dominio es el conjunto de valores utilizados para encontrar la solución óptima, cada conjunto se define por un límite superior e inferior de la siguiente

manera:

$$\begin{aligned} \theta_1 &\in (\ -\phi_1\ , \phi_1) \\ \theta_2 &\in (\ -\phi_2\ , \phi_2) \\ r_1 &\in (\ 0\ , l_1) \\ r_2 &\in (\ 0\ , l_2) \end{aligned} \tag{4-4}$$

Donde ϕ_k es el límite para el ángulo θ_k y l_k es el límite para la distancia r_k.

Al resolver el sistema de ecuaciones de 4-2 en el software matemático de MATLAB, se realizaron simulaciones para verificar la generación de trayectorias que podría resultar de este sistema, para ello se realizaron algunas pruebas dando valores diferentes a los puntos por los que debía pasar la trayectoria, de igual modo se cambió la orientación inicial para determinar la forma de la ruta.

Las figuras **4-6** y **4-7** muestran los resultados de pruebas realizadas al hacer que la trayectoria pase por los puntos en las coordenadas (x, y) y dándoles diferentes ángulos de orientación al iniciar la ruta del robot, en este caso a 0, 90, 180 y 270 grados.

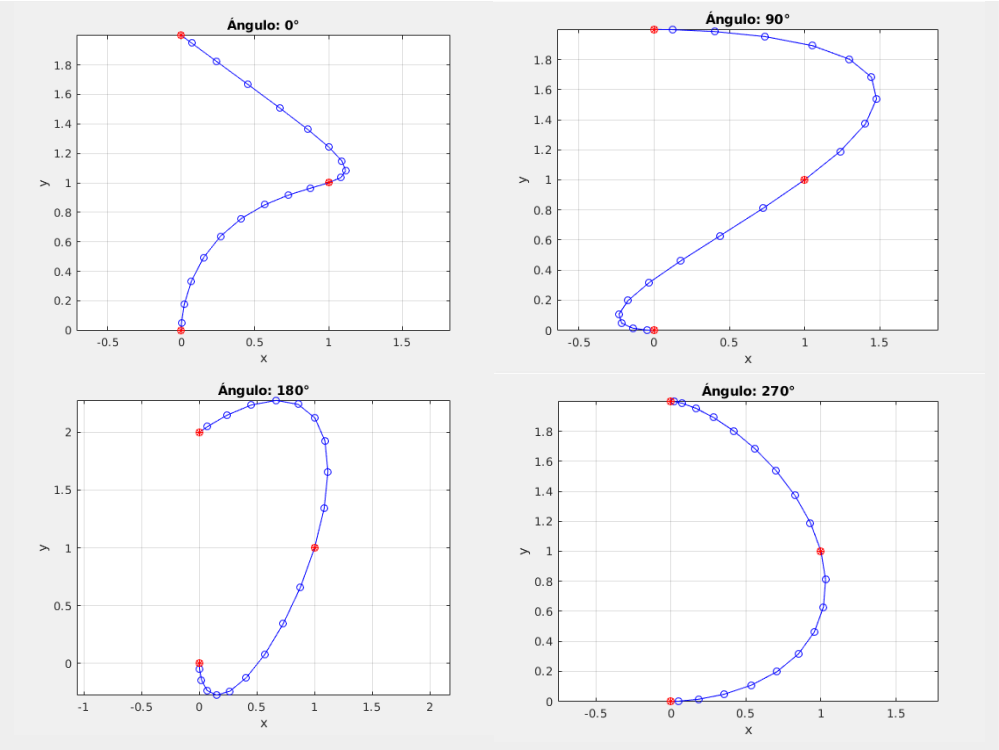


Figura **4-6**.: Prueba uno de generación de trayectorias en MATLAB.

Para la prueba uno los puntos son: $(0, 0)$ para el punto inicial en q_0; $(1, 1)$ para el punto intermedio en q_1 y para el punto final q_2 se asigna el valor de $(0, 2)$. Los puntos se ubican de esa manera debido a que se simula que el robot se encuentra frente a un obstáculo que le impide seguir una linea recta en su camino.

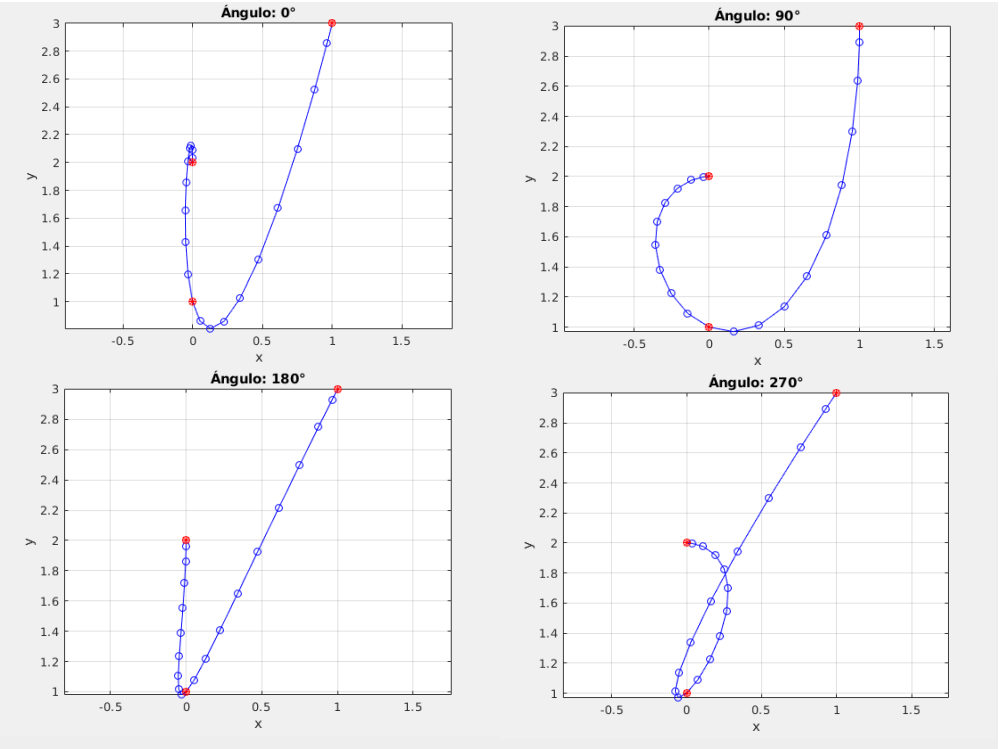


Figura **4-7**.: Prueba dos de generación de trayectorias en MATLAB.

Para la segunda prueba los puntos son: $(0, 2)$ para el punto inicial en q_0; $(0, 1)$ para el punto intermedio en q_1 y para el punto final q_2 se asigna el valor de $(1, 3)$; esta vez se distribuyen los puntos de esa manera sólo para generar rutas mas complejas.

Se observa que la ruta es diferente en cada caso debido a la orientación del ángulo inicial, pero aún así se logra que la trayectoria pase por los puntos establecidos para cada prueba.

5. Métodos de optimización

5.1. Técnicas de optimización

En toda disciplina, la optimización juega un papel importante en el diseño de las actividades a seguir; desde simples actividades rutinarias, en la vida diaria se presentan diversas situaciones que implican la optimización, incluso de manera intuitiva.

La optimización puede verse como el proceso de obtener la mejor solución a un problema [Chong y Zak, 2013]. En términos generales, las características y requerimientos del problema determinan si la mejor solución de todas (óptimo global) puede ser encontrado. Hay que ajustar las entradas del sistema o modelo matemático, para encontrar el valor máximo o mínimo de la salida como resultado. La entrada consiste en variables, el proceso o función se denomina función de costo o función objetivo. La salida es el costo o aptitud [Randy, 2004].

Los algoritmos de optimización son métodos de búsqueda, donde el objetivo es encontrar la solución a un problema de optimización, tal que una cantidad dada sea optimizada, probablemente sujeta a un conjunto de restricciones [Engelbrecht, 2006]. Esta simple definición esconde varios problemas complejos involucrados en el proceso de optimización; por ejemplo, la solución podría consistir en una combinación de diferentes tipos de datos, restricciones no lineales del espacio de búsqueda, que las características del problema están en función del tiempo, o las cantidades a optimizar tengan objetivos encontrados, refiriéndose a optimización multi-objetivo.

Cada problema de optimización tiene al menos estos tres elementos a considerar:

- La función de costo, que representa la cantidad a optimizar, sea maximizar o minimizar. Sea f la función de costo. Entonces el máximo de f es un mínimo de –f. Algunos problemas no definen explícitamente la función objetivo, sino que se pretende encontrar una solución que satisfaga a todo el conjunto de restricciones.
- Un conjunto de variables, el cual afecta el valor de la función objetivo. Si x representa una variable (variable independiente), entonces f(x) representa la calidad de la solución candidato x.
- Un conjunto de restricciones, que limitan o restringen los valores a los que pueden asignarse las variables. Muchos problemas definen al menos un conjunto de límites, los cuales definen el dominio de cada variable. Las restricciones pueden llegar a ser más complejas, de tal manera que pueden excluir soluciones candidato de ser consideradas las óptimas. Se dice que un punto es viable si éste satisface todas las restricciones.

Los esquemas de optimización pueden dividirse en nueve categorías, los cuales no son mutuamente excluyentes: En primer lugar la metodología a seguir, el número de variables, de igual modo en función del tiempo, el tipo de variable, el número de restricciones, la forma de búsqueda, el grado de linealidad de la función objetivo, el número de óptimos y finalmente el número de criterios de optimización [Randy, 2004; Engelbrecht, 2006].

Las soluciones que arroja un algoritmo de optimización pueden clasificarse de acuerdo a la calidad de la solución. Entonces el óptimo global es la mejor solución de un conjunto de soluciones candidato, por lo tanto, las condiciones especificadas garantizan la existencia de un mínimo global si éstas son satisfechas; sin embargo, no niegan la presencia de una solución global si tales condiciones no se cumplen. Hasta ahora, no hay un método que identifique si existe un óptimo para toda clase de problemas [Porta García, 2007].

Los métodos de optimización clásicos pueden clasificarse en dos grupos: métodos directos y basados en gradiente [Randy, 2004]. En los métodos directos, solo la función objetivo y las restricciones son utilizadas para guiar la estrategia de búsqueda, mientras que en los métodos basados en gradiente se utilizan la primera y/o segunda derivada de la función objetivo así como las restricciones para guiar el proceso de búsqueda. Los métodos directos son más lentos, ya que no utilizan la información arrojada por las derivadas; pueden ser aplicados a diversos problemas sin mayores modificaciones. Por otro lado, los métodos basados en gradiente convergen más rápido a la cercanía de la solución óptima; pero no son eficientes en problemas discontinuos o no diferenciables.

La mayoría de las técnicas numéricas están diseñadas para converger a soluciones que se encuentran cerca del punto de inicio de las iteraciones del algoritmo. Por ejemplo, algunos de los métodos numéricos de optimización más conocidos aplicados a problemas unidimensionales pueden ser el de Newton-Raphson, caminata aleatoria y paso descendiente [Porta García, 2007].

Los métodos convencionales de optimización presentan algunas dificultades tales como [Engelbrecht, 2006]:

- La convergencia a una solución óptima depende de la solución inicial seleccionada.
- Muchos algoritmos tienden a estancarse en mínimos locales.
- Un algoritmo eficiente para resolver determinado problema de optimización puede que no sea eficiente para otro diferente.
- Los algoritmos no son eficientes para manejar problemas donde el espacio de búsqueda es discreto.
- Los algoritmos no pueden ser utilizados eficientemente en una máquina de procesamiento paralelo. Los algoritmos naturales de optimización, que superan varias de las desventajas mencionadas, poco a poco han ido reemplazando a los métodos clásicos para resolver problemas prácticos.

En inteligencia artificial, un algoritmo evolutivo es un algoritmo genérico de optimización metaheurística basado en poblaciones. Un algoritmo evolutivo utiliza mecanismos inspirados en la evolución biológica: reproducción, mutación, recombinación, selección natural y la supervivencia del más apto. Las soluciones candidato del problema de optimización juegan el papel de individuos en una población, y la función de costo determina el ambiente en el cual dichas soluciones "viven". Entonces tal población evoluciona después de determinada cantidad de iteraciones donde participan los operadores mencionados [Porta García, 2007].

Algunas técnicas similares englobadas dentro de los algoritmos evolutivos que difieren en detalles de implementación y la naturaleza particular del problema de aplicación que existen en la actualidad son los Algoritmos Genéticos, la programación genética, programación evolutiva y estrategias evolutivas [Engelbrecht, 2006].

Otros algoritmos naturales de optimización que modelan procesos naturales son: temple simulado (Simulated Annealing), algoritmos culturales, optimización con enjambres de partículas (particle swarm optimization) y el algoritmo de optimización de la hierba mala invasora (invasive weed optimization algorithm), entre otros.

Los métodos de optimización clásica requieren utilizar la información de la derivada de la función de costo. En este caso, la función de costo definida por la ecuación 3-3 representa el número de elementos en un conjunto, que se calcula utilizando el método numérico descrito en la sección 3, esto significa que no hay una función explícita de la derivada.

Este tipo de problemas de optimización pueden resolverse utilizando el enfoque de los métodos heurísticos, por lo general, una técnica heurística es la intuitiva utilizada para obtener una solución aproximada a un problema cuando los métodos clásicos fallan o son muy lentos.

Al utilizar métodos heurísticos no es necesario garantizar que la función objetivo sea convexa, diferenciable o incluso continua porque estos métodos se basan solo en evaluaciones de funciones. La compensación es que la heurística no puede garantizar la convergencia asintótica.

Esta sección proporciona una breve descripción de las tres técnicas heurísticas utilizadas para resolver el problema de optimización previamente formulado.

5.2. Método de búsqueda ciega (Blind search)

Cuando presenta un problema, a menudo se analizan las acciones que se pueden realizar para resolverlo, por lo regular existen al menos dos opciones para llegar a un estado deseado, al igual que para llegar a una ciudad existen distintos caminos, así mismo existen varios caminos que pueden llevar a una solución.

Al igual que elegir un camino, los agentes resolventes de problemas también tienen varios caminos con los cuales pueden llegar a una solución, estos caminos son acciones que pueden realizar, y dichas acciones tienen consecuencias que provocan un estado, si este estado es el

deseado entonces se sabe que el camino o la acción que se realizó es correcta, si no, se puede concluir que ese camino no es adecuado y se debe buscar otra opción.

Lo anterior se asemeja a las estrategias de búsqueda, la búsqueda constituye las acciones a realizar para encontrar la solución y las estrategias son la elección de los distintos caminos, ya que de la elección de los mismos depende el éxito de la solución del problema [Bratko, 2001].

La búsqueda es una técnica para resolver problemas cuya solución consiste en una serie de pasos que frecuentemente deben determinarse mediante la prueba sistemática de las alternativas. Desde los inicios de la Inteligencia Artificial, la búsqueda se ha aplicado en diversas clases de problemas como juegos de dos jugadores, problemas de satisfacción de restricciones y problemas de encontrar solución a través de agentes.

El problema se plantea entonces en términos de encontrar una configuración objetivo a partir de una configuración inicial dada, aplicando transformaciones válidas según el modelo del universo. La respuesta es la secuencia de transformaciones cuya aplicación sucesiva lleva a la configuración deseada.

Debido a la complejidad exponencial del grafo implícito, se irá generando, paso a paso, una porción del grafo conforme avance el proceso de búsqueda. El grafo explícito es el subgrafo del grafo implícito que se va generando durante el proceso de búsqueda de una secuencia de operadores que resuelva nuestro problema (camino solución), usualmente, en forma de árbol, de ahí su nombre: Árbol de Búsqueda [Romero, 2016]. En la figura **5-1** se observa la representación de este árbol:

Figura **5-1**.: Árbol de búsqueda [Romero, 2016].

El método de búsqueda ciega se puede describir como una secuencia de iteración basada en dos operaciones: generación de la solución y procedimiento de actualización.

El generador de soluciones es la parte del algoritmo que produce una solución candidata en el espacio de búsqueda, generalmente se usa una distribución uniforme para producir la solución aleatoria.

El procedimiento de actualización es la regla que se usa para reemplazar la solución aceptada actual por la solución candidata, la regla más básica es reemplazar la solución

aceptada cuando la solución candidata produce un mejor valor cuando se evalúa en la función de costo.

El método finaliza cuando se cumple algún criterio, generalmente un número máximo de evaluaciones o cuando la solución ya no mejora en las últimas iteraciones.

Este método básico tiene la ventaja de que la probabilidad de encontrar la solución óptima aumenta con cada iteración, la desventaja es que la convergencia es lenta y es posible que el error sea grande cuando no se utilizan suficientes evaluaciones de la función de costo.

Se desarrollan algoritmos más complejos basados en el método de búsqueda ciega, es decir, todavía se usa como una medida de comparación de la eficiencia de otros algoritmos.

Algunas características de este método según [Bravo y Raúl, 2010] son:

- El orden en que la búsqueda se realiza no depende de la naturaleza de la solución buscada.
- Exploración del árbol de búsqueda sistemáticamente, pero sin información.
- La localización de la meta no altera el orden de expansión de los nodos.
- No tienen en cuenta el coste de la solución en la búsqueda.
- Su funcionamiento es sistemático, siguen un orden de visitas y generación de nodo establecido por la estructura del espacio de búsqueda.
- Los dos métodos básicos de la búsqueda a ciegas son la búsqueda primero a lo ancho (breadth-first search) y la búsqueda primero en profundidad (depth-first search).

Las búsquedas no informadas también son denominadas búsqueda a ciegas, y esto es porque no tienen información suficiente acerca de los estados. Existen varios tipos de búsqueda no informada [Torres Soler y Garzón Torres, 2015], para continuar se dará una breve explicación de ellos.

5.2.1. Búsqueda primero en anchura.

La búsqueda en anchura consiste en buscar horizontalmente los nodos, es decir expandir de izquierda a derecha, todos los nodos de un nivel, para así poder pasar al siguiente nivel, es una búsqueda óptima cuando el espacio de estados no es infinito, sin embargo, esta búsqueda tiene complejidad de espacio ya que se deben guardar en memoria, los nodos extendidos [Russell y Norvig, 2004].

La búsqueda en anchura tiene una estructura de cola FIFO (del inglés First In First Out), debido a que el primer elemento en entrar será también el primero en salir, es decir, el primero en extender es el primero en extender a nuevos nodos y así sucesivamente tal como se ilustra en la figura **5-2**.

Se trata de una búsqueda completa si el número de estados posibles es finito. Por otro lado, no es óptima si el factor de ramificación es infinito, existe complejidad de tiempo y se debe guardar en la memoria los nodos expandidos.

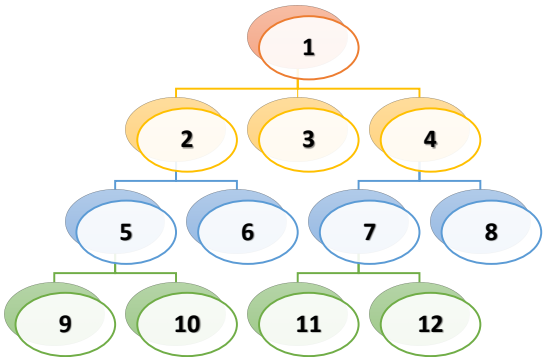


Figura **5-2**.: Ejemplo de búsqueda primero en anchura.

5.2.2. Búsqueda primero en profundidad

La búsqueda primero en profundidad consiste en buscar verticalmente, es decir, el nodo raíz se expande, luego el hijo es expandido, a los siguientes de manera vertical y por la izquierda, en caso de que el nodo este repetido se pasa al siguiente nodo de la derecha, esto se realiza hasta terminar en el último nodo de esa rama. Se puede decir que tiene una estructura de cola LIFO (First In, Last Out; primero en entrar, último en salir), tal como se ilustra en la figura **5-3**.

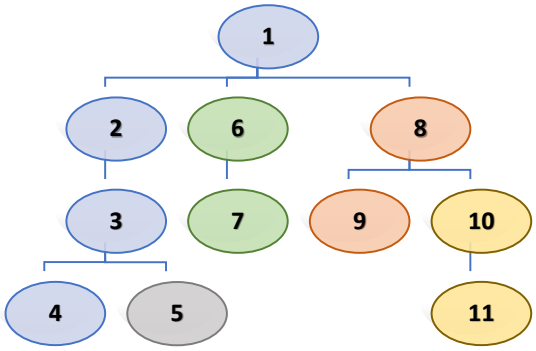


Figura **5-3**.: Ejemplo de búsqueda primero por profundidad.

La búsqueda es incompleta si se hace una mala elección en la rama, ya que podría suceder que nunca se llegue a la solución. Tampoco es óptima cuando hay un sin número de estados posibles, debido a que no existe solución óptima. El tiempo puede ser infinito si se toma un mal camino. Por otra parte, no se necesita tanto espacio de memoria, ya que se almacena un solo camino.

La búsqueda primero en profundidad tiene 3 variantes: Búsqueda en profundidad hacia atrás, Búsqueda con profundidad limitada y Búsqueda de profundidad iterativa [Russell y Norvig, 2004].

5.2.3. Búsqueda bidireccional.

Es aquella en la que se pueden tomar dos direcciones, hacia adelante desde el estado inicial o hacia atrás desde el estado objetivo; estas direcciones se toman al mismo tiempo.

La búsqueda bidireccional es considerada una búsqueda que posee un algoritmo llamado de fuerza bruta, debido a que necesita tener un estado objetivo planteado, es decir que necesita conocer cuál será el objetivo, por lo tanto, no es simplemente una prueba para una condición deseada. En la imagen de la figura **5-4** se observa este procedimiento.

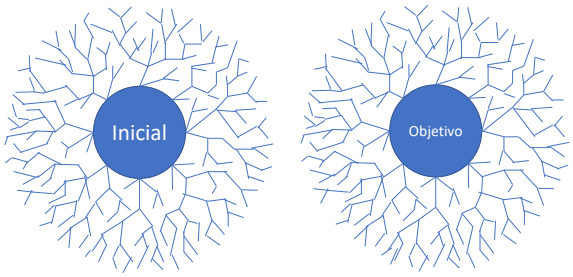


Figura **5-4**.: Ejemplo de búsqueda bidireccional.

Se trata de una búsqueda completa ya que si ambas búsquedas, sea hacia adelante o hacia atrás, están en la misma frontera se habrá encontrado la solución.

Este tipo de búsqueda es óptima, debido a que se encuentra la solución en menos pasos que las otras búsquedas, además se optimiza el tiempo ya que la solución siempre está en medio y al menos una de las búsquedas debe ser guardada en el espacio de la memoria [Bratko, 2001].

Las estrategias de búsqueda no informada son utilizadas cuando no se sabe del problema más que su estado inicial y su estado objetivo, estas estrategias sirven para encontrar una solución a un problema [Cruz y Herrera, 2011].

Existen distintas estrategias, cada una tiene una variación en cuanto a los factores más importantes en la resolución de un problema, los cuales son la optimización, completitud, la complejidad de espacio y la complejidad de tiempo.

Una búsqueda sobre un espacio consiste en generar una sucesión de puntos del espacio pasando de uno a otro por medio de una serie de transformaciones o movimientos [Melián, Pérez y Vega, 2003]. Un procedimiento de búsqueda para resolver un problema de optimización realiza recorridos sobre el espacio de las soluciones alternativas y selecciona la mejor solución encontrada en el recorrido.

5.3. Método de alpinismo (Hill climbing)

Debido a que en muchos problemas el tamaño del espacio de búsqueda hace que sea inabordable de manera exhaustiva, se deben buscar estrategias diferentes a las mencionadas hasta ahora [Russell y Norvig, 2004]. En primer lugar, puede haber pocas posibilidades de guardar información para recuperar caminos alternativos dado el gran número de alternativas que se presentan, lo que obliga a imponer restricciones de espacio decidiendo qué nodos deben ser explorados y cuáles deben ser descartados sin volver a considerarlos [Skiena, 1998].

Este método general de eliminar ramas de exploración del árbol de búsqueda da lugar a una familia de algoritmos conocida como de ramificación y poda (branch and bound), en la que olvidamos aquellas opciones que no parecen prometedoras. Estos algoritmos permiten mantener una memoria limitada, ya que desprecian parte del espacio de búsqueda, pero se arriesgan a no hallar la mejor solución, ya que ésta puede estar en el espacio de búsqueda que ha sido eliminado.

De los algoritmos de ramificación y poda, uno de los más utilizados es el de ascenso de colinas o de alpinismo (Hill climbing) [Cohen, Greiner y Schuurmans, 1991], es un algoritmo heurístico e iterativo que inicia dando una solución arbitraria al problema, después intenta encontrar una solución mejor por incrementalmente el cambio de un solo elemento de la solución. Si el cambio produce una mejor solución, se hace un cambio incremental a la nueva solución, repitiendo hasta que ya no se puedan encontrar mejores soluciones. Es bueno para encontrar un óptimo local pero no es necesariamente garantía de encontrar la mejor solución posible de todas las posibles soluciones [Gerkey, Thrun y Gordon, 2005].

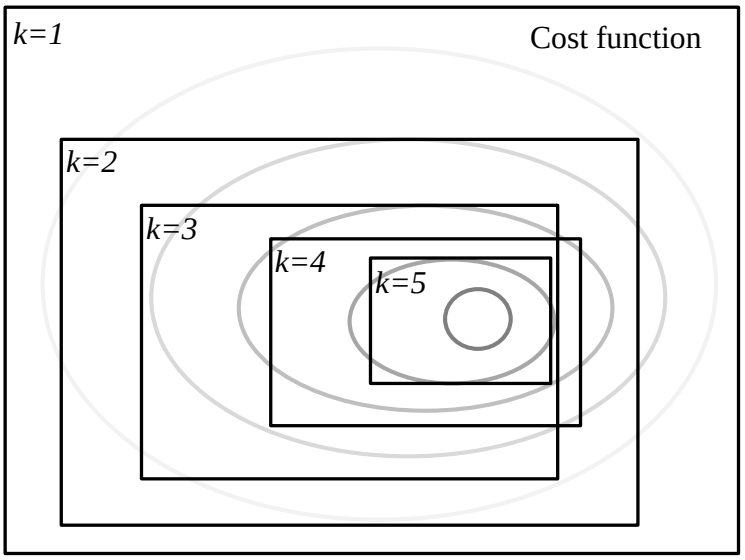


Figura **5-5**.: Contracción del espacio de búsqueda.

El método de alpinismo es una extensión de la búsqueda ciega, los mismos dos pasos definen una iteración, pero se presentan varias diferencias en la implementación.

La primera diferencia es que el generador de soluciones no producirá una sola solución candidata sino una colección de soluciones en torno a la solución aceptada actual.

Otra diferencia es que el espacio de búsqueda se contrae en cada iteración usando alguna regla determinista, por ejemplo, los límites de los intervalos se pueden acercar a la solución aceptada en función de algún factor.

En el procedimiento de actualización, la solución aceptada se reemplaza por la mejor solución entre todas las soluciones candidatas generadas durante cada iteración.

La idea detrás del método de alpinismo es examinar el vecindario de la solución aceptada actual para elegir la dirección más prometedora para moverse. Este enfoque puede verse como una herramienta para aproximar el gradiente de la función.

En la figura **5-6** se observa de manera gráfica la explicación de operación del método.

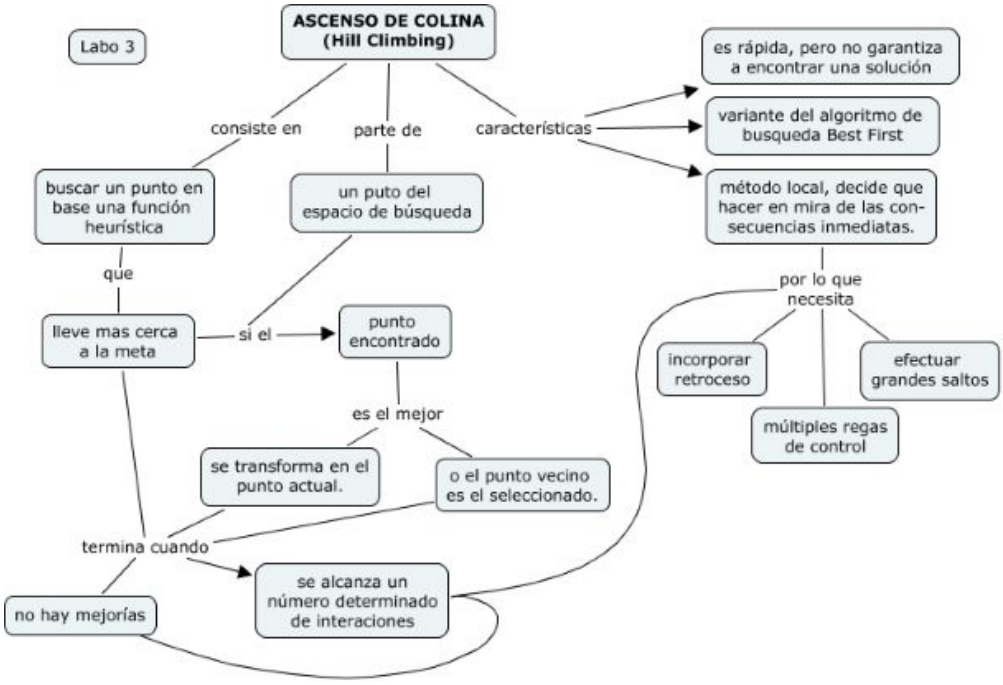


Figura **5-6**.: Algoritmo de Búsqueda Heurística: Ascenso de Colina [CmapTools ihmc].

El comportamiento del método de alpinismo se describe por dos etapas: la exploración y la etapa de aproximación.

La etapa de exploración ocurre durante las primeras iteraciones, cuando el espacio de búsqueda es lo suficientemente grande como para cubrir todas las soluciones factibles, el propósito de esta etapa es generar una solución lo suficientemente cercana a la solución global óptima.

La etapa de aproximación ocurre durante las últimas iteraciones, cuando el espacio de búsqueda se hace más pequeño, el propósito de esta etapa es aumentar la precisión de la solución aceptada.

La principal ventaja del método de alpinismo sobre la búsqueda ciega es que produce resultados más precisos cuando la función de costo es uniforme. Tiene la desventaja de que el algoritmo puede obtener valores en soluciones óptimas locales.

La figura **5-5** muestra una descripción gráfica de la contracción esperada del espacio de búsqueda mediante el método de alpinismo. De este algoritmo existen variantes [Melián, Pérez y Vega, 2003] que pueden ser útiles en distintos casos:

5.3.1. Alpinismo simple

Consiste en elegir siempre el primer operador que suponga una mejora respecto al nodo actual, de manera que no se exploran todas las posibilidades accesibles, esto permite ahorrar el explorar cierto número de descendientes. Presenta la ventaja de ser más rápido que explorar todas las opciones, pero tiene la desventaja de que hay más probabilidad de no alcanzar las mejores soluciones.

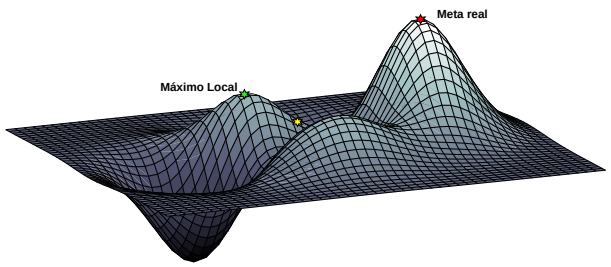


Figura **5-7**.: Ejemplo de alpinismo simple.

5.3.2. Alpinismo por máxima pendiente (Steepest ascent Hill climbing)

Expande todos los posibles descendientes de un nodo y elige el que suponga la máxima mejora respecto al nodo actual. Este algoritmo supone que la mejor solución se encontrará a través del sucesor que mayor diferencia tenga respecto a la solución actual, siguiendo una política avariciosa. La utilización de memoria para mantener la búsqueda es nula, ya que solo se tiene en cuenta el mejor nodo.

Se pueden hacer versiones que guarden caminos alternativos que permitan una vuelta atrás en el caso de que se considere que la solución a la que ha llegado no es suficientemente buena, pero en este caso se imponen ciertas restricciones de memoria para no tener un coste en espacio demasiado elevado.

La estrategia que se usa en este algoritmo hace que sus problemas vengan principalmente derivados por las características de las funciones heurísticas que se utilizan. Por un lado, se

considera que el algoritmo dejará de explorar cuando no encuentra ningún nodo accesible mejor que el actual, pero como no mantiene memoria del camino recorrido, le será imposible reconsiderar su decisión de modo que, si se ha equivocado, puede ocurrir que la solución a la que llegue no sea un óptimo global, sino solo un óptimo local.

Esta posibilidad, muy común al atacar problemas reales, se puede presentar cuando la función heurística que se utilice tenga óptimos locales, de forma que, dependiendo de por donde se comience la búsqueda, se acaba encontrando un óptimo distinto.

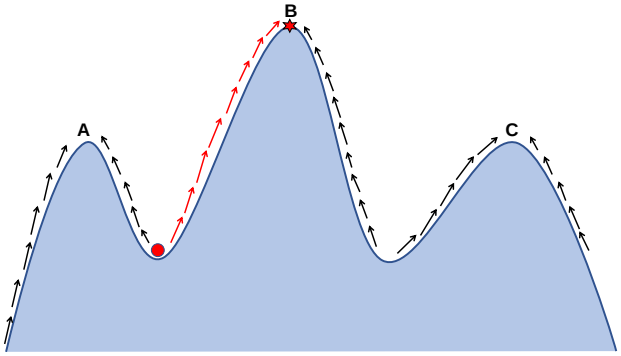


Figura **5-8**.: Ejemplo de alpinismo por máxima pendiente.

5.3.3. Búsqueda por ascenso con reinicio aleatorio (random restarting hill climbing)

Para subsanar este problema se puede intentar la ejecución del algoritmo cierto número de veces desde distintos puntos escogidos aleatoriamente y quedarse solo con la mejor exploración. Por simple que parezca, muchas veces esta estrategia es más efectiva que otros métodos más complejos, y resulta con un costo muy bajo de implementar.

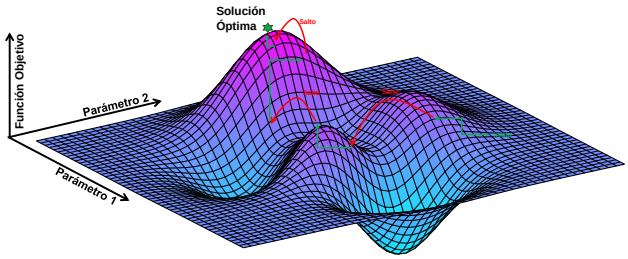


Figura **5-9**.: Ejemplo de alpinismo de búsqueda por ascenso con reinicio aleatorio.

La única dificultad que se puede encontrar radica en la capacidad para poder generar los

distintos puntos iniciales de búsqueda, ya que en ocasiones el problema no permite hallar soluciones iniciales con facilidad.

5.3.4. Búsqueda en haces (beam search)

Un problema con el que se encuentran este tipo de algoritmos son las zonas del espacio de búsqueda en las que la función heurística no es informativa, como por ejemplo las denominadas mesetas y las funciones en escalón, en las que los valores de los nodos vecinos al actual tienen valores iguales, y por lo tanto una elección local no permite decidir el camino a seguir. Para evitar estos problemas se debe extender la búsqueda más allá de los vecinos inmediatos para obtener información suficiente para encaminar la búsqueda, pero supone un coste adicional en cada iteración, prohibitivo cuando el factor de ramificación es excesivamente grande.

Una alternativa es permitir que el algoritmo guarde parte de los nodos visitados para proseguir la búsqueda por ellos en caso de que el algoritmo se quede atascado en un óptimo local.

El algoritmo más utilizado de este tipo es el denominado búsqueda en haces (beam search). En este algoritmo se van guardando un número N de las mejores soluciones, expandiendo siempre la mejor de ellas. De esta manera no se sigue un solo camino sino N caminos eventualmente distintos. Las variantes que se pueden plantear están en cómo se escogen esas N soluciones que se mantienen. Si siempre se sustituyen las peores soluciones de las N por los mejores sucesores del nodo actual, se cae en el peligro de que todas acaben en el mismo camino, reduciendo la capacidad de volver hacia atrás, así que el poder mantener la variedad de las soluciones guardadas es importante.

Está claro que cuantas más soluciones se guarden, más posibilidad habrá de encontrar una buena solución, pero se tiene un coste adicional tanto en memoria como en tiempo, ya que habrá que decidir con qué sucesores quedarse y qué soluciones guardadas se sacrifican.

El algoritmo acaba cuando ninguno de los sucesores mejora a las soluciones guardadas, lo que quiere decir que todas las soluciones son un óptimo local, esta técnica de búsqueda tiene bastantes puntos en común con los algoritmos genéticos.

5.4. Algoritmo genético

Los primeros ejemplos de lo que hoy se podrían llamar algoritmos genéticos aparecieron a finales de los 50 y principios de los 60, programados en computadoras por biólogos evolutivos que buscaban explícitamente realizar modelos de aspectos de la evolución natural [Marczyk, 2004].

En 1962, investigadores como [Box, 1957], [Friedman, 1959], [Bledsoe, 1961] y [Bremermann, 1962] habían desarrollado independientemente algoritmos inspirados en la evolución

para optimización de funciones y aprendizaje automático, pero sus trabajos generaron poca reacción [Sobhaninehad, Noorzad y Ansari, 2007].

En 1965 surgió un desarrollo más exitoso, cuando [Rechenberg, 1965] entonces de la Universidad Técnica de Berlín, introdujo una técnica que llamó estrategia evolutiva. En esta técnica no había población ni cruzamiento; un padre mutaba para producir un descendiente, y se conservaba el mejor de los dos, convirtiéndose en el padre de la siguiente ronda de mutación. Versiones posteriores introdujeron la idea de población.

El siguiente desarrollo importante en el campo vino en 1966, cuando [Fogel, Owens y Walsh, 1966] introdujeron una técnica que llamaron programación evolutiva. En este método, las soluciones candidatas para los problemas se representaban como máquinas de estado finito sencillas; al igual que en la estrategia evolutiva de Rechenberg, su algoritmo funcionaba mutando aleatoriamente una de estas máquinas simuladas y conservando la mejor de las dos. Sin embargo, lo que todavía faltaba en estas dos metodologías era el reconocimiento de la importancia del cruzamiento.

En 1962, el trabajo de [Holland, 1992] sobre sistemas adaptativos estableció las bases para desarrollos posteriores; y lo que es más importante, Holland fue también el primero en proponer explícitamente el cruzamiento y otros operadores de recombinación. Sin embargo, el trabajo fundamental en el campo de los algoritmos genéticos apareció en 1975, con la publicación del libro "Adaptación en Sistemas Naturales y Artificiales". Basado en investigaciones y documentos anteriores del propio Holland y de colegas de la Universidad de Michigan, este libro fue el primero en presentar sistemática y rigurosamente el concepto de sistemas digitales adaptativos utilizando la mutación, la selección y el cruzamiento, simulando el proceso de la evolución biológica como estrategia para resolver problemas.

El libro también intentó colocar los algoritmos genéticos sobre una base teórica firme introduciendo el concepto de esquema. Ese mismo año, la importante tesis de [De Jong, 1975] estableció el potencial de los algoritmos genéticos demostrando que podían desenvolverse bien en una gran variedad de funciones de prueba, incluyendo paisajes de búsqueda ruidosos, discontinuos y multimodales.

Estos trabajos fundacionales establecieron un interés más generalizado en la computación evolutiva. Entre principios y mediados de los 80, los algoritmos genéticos se estaban aplicando en una amplia variedad de áreas, desde problemas matemáticos abstractos como el problema de la mochila (bin-packing) [Korf, 2002] y la coloración de grafos hasta asuntos tangibles de ingeniería como el control de flujo en una línea de ensamble, reconocimiento y clasificación de patrones y optimización estructural.

Al principio, estas aplicaciones eran principalmente teóricas. Sin embargo, al seguir proliferando la investigación, los algoritmos genéticos migraron hacia el sector comercial, al cobrar importancia con el crecimiento exponencial de la potencia de computación y el desarrollo de Internet.

Hoy en día, los algoritmos genéticos están resolviendo problemas de interés cotidiano en áreas de estudio tan diversas como la predicción en la bolsa y la planificación de la cartera

de valores, ingeniería aeroespacial, diseño de microchips, bioquímica y biología molecular, y diseño de horarios en aeropuertos y líneas de montaje. La potencia de la evolución ha tocado virtualmente cualquier campo que uno pueda nombrar, modelando invisiblemente el mundo que nos rodea de incontables maneras, y siguen descubriéndose nuevos usos mientras la investigación sigue su curso.

Y en el corazón de todo esto se encuentra la simple y poderosa idea de Charles Darwin: que el azar en la variación, junto con la ley de la selección, es una técnica de resolución de problemas de inmenso poder y de aplicación casi ilimitada.

Aunque se ve a los algoritmos genéticos como máximos representantes de estas ideas, hay otros muchos ejemplos en los que se pueden utilizar las ideas de selección por supervivencia para conseguir optimizaciones de problemas.

Basándose en la idea de que los más aptos sobreviven mejor a las situaciones del entorno, es posible crear simulaciones en las que los individuos están obligados a sobrevivir en un ambiente hostil, y solo aquellos con las condiciones más adecuadas para ese entorno serán capaces de durar el tiempo suficiente para reproducirse y dejar descendencia en la que sus caracteres positivos den más opciones a los nuevos individuos, que junto con la mutación, puede producir una paulatina adaptación al medio. Tras este proceso durante generaciones, los individuos que queden reflejarán en sus características una solución (cuasi) óptima al problema de la supervivencia en el entorno.

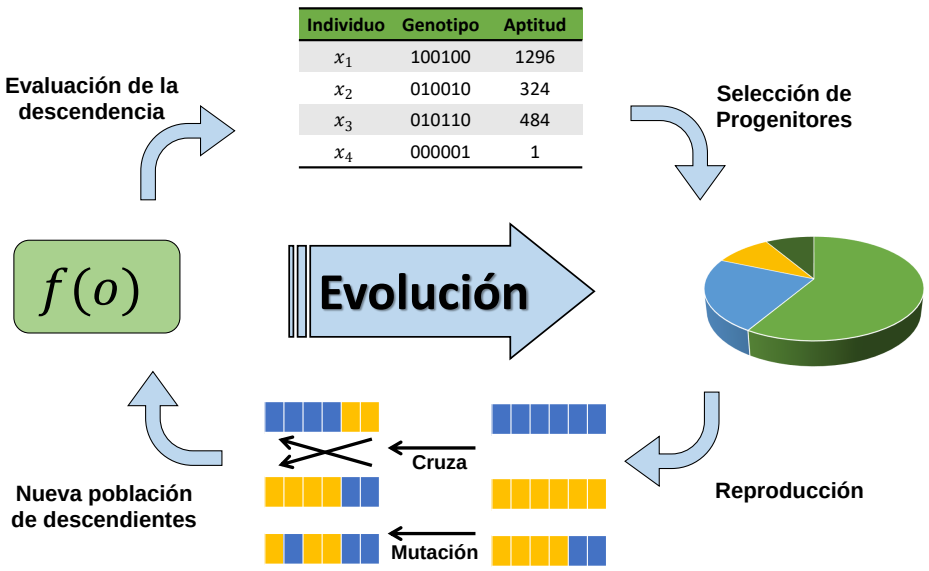


Figura **5-10**.: Proceso de evolución de un algoritmo genético.

Un algoritmo genético se basa en un paradigma diferente al de los métodos de optimización previos, cae en una categoría llamada algoritmos de inspiración biológica. Este tipo de heurística se basa en procesos naturales, por ejemplo, distribución térmica, proceso de poli-

nización, etc., otros se basan en el movimiento de algún grupo de animales como murciélagos, hormigas, ballenas, etc.

Los algoritmos genéticos, por otro lado, se basan en el proceso de selección natural, la función de costo está asociada a la aptitud de cada solución, teniendo en cuenta este concepto, cada problema debe formularse como un problema de optimización cuando se utiliza esta heurística.

Una de las características principales de los algoritmos genéticos es que las soluciones están codificadas, inspiradas en las cadenas de ADN, en forma de cadenas binarias, cada elemento de la cadena binaria se llama *gen*. Para calcular la longitud de las cadenas binarias, el espacio de búsqueda debe discretizarse en función de la resolución que se espera obtener y los intervalos que definen las soluciones factibles. Hay otros tipos de representación que incluyen el valor real y los árboles binarios, entre otros.

Otra gran diferencia es que los algoritmos genéticos son métodos de población, esto significa que se crean varias soluciones en todo el espacio de búsqueda, cada una de ellas se mueve a la solución óptima de forma independiente, esta característica permite que el algoritmo salga de los valores óptimos locales, siempre que ya que las soluciones son lo suficientemente diversas.

En estos algoritmos, una solución representa un individuo, las iteraciones representan generaciones que crean una nueva descendencia utilizando la información de la población anterior mediante el uso de operadores matemáticos inspirados en procesos naturales:

- Evaluación: este es el proceso donde las cadenas binarias se convierten en soluciones en los espacios de búsqueda y se evalúan en la función de costos. Esta herramienta está inspirada en la relación entre el fenotipo y el genotipo en un organismo biológico.
- Selección: Este paso representa la competencia de los diferentes individuos para poder reproducirse, en general el proceso de selección implica la comparación de la aptitud del individuo para elegir a los mejores individuos para crear una población de padres. Existen muchas herramientas para implementar este proceso dependiendo de la representación de las soluciones y el uso de variables aleatorias.
- Cruza: esta parte del algoritmo produce la descendencia usando segmentos de las cadenas de los individuos que se seleccionaron en el paso anterior. En esta parte, las cadenas de los individuos en la descendencia se construyen peinando las cadenas de los padres. Este proceso está inspirado en la recombinación genética que ocurre durante la fecundación.
- Mutación: esta herramienta está inspirada en los cambios aleatorios que se producen en los organismos biológicos, el objetivo es mantener la diversidad de los individuos y ser capaz de producir soluciones en todo el espacio de búsqueda. Usualmente este proceso se implementa cambiando aleatoriamente el valor de algunos elementos en la cadena binaria usando una función de distribución probabilística.

Al final de cada generación, la población de los descendientes reemplaza la población de los individuos anteriores; en algunos casos, el mejor individuo, que representa la solución óptima actual, se agrega a la siguiente generación para evitar perder el punto óptimo.

La principal ventaja de los algoritmos genéticos sobre los métodos anteriores es la exploración paralela del espacio de búsqueda que generalmente termina encontrando la solución óptima con alta precisión. La desventaja es que el rendimiento del método depende del problema y la selección adecuada de las herramientas y los parámetros para la implementación.

Aunque puede parecer asombroso y no intuitivo, los algoritmos genéticos han demostrado ser una estrategia enormemente poderosa y exitosa para resolver problemas, demostrando de manera espectacular el poder de los principios evolutivos. Se han utilizado algoritmos genéticos en una amplia variedad de campos para desarrollar soluciones a problemas tan difíciles o más difíciles que los abordados por los diseñadores humanos. Además, las soluciones que consiguen son a menudo más eficientes, más elegantes o más complejas que las que un humano podría producir.

6. Comparación de los métodos de optimización

La comparación de eficiencia de los algoritmos de los diferentes métodos de optimización implementados se llevó a cabo mediante la realización de pruebas reales, tal como se describió en la sección 3, mediante el uso de la plataforma robótica comercial llamada *TurtleBot2*, además de una computadora embebida *Raspberry Pi 3B* con Ubuntu 16,04 Xenial y la versión de *ROS Kinetic*. La siguiente figura muestra una imagen de la plataforma:

Figura **6-1**.: Plataforma experimental.

Una vez programado el algoritmo y puesto en marcha, el robot es completamente autónomo, pero también fue necesario el uso de una computadora de escritorio conectada de manera remota a la computadora embebida para supervisar el proceso de mapeo y recopilar los resultados de cada prueba.

La imagen de la figura **6-2** muestra el área de pruebas en la que se realizaron los experimentos. Se trata de un espacio cerrado con algunos obstáculos, además de que fue adaptado para realizar de forma conveniente estas pruebas dentro del Laboratorio de Automatización y Control de la Universidad Politécnica de Tulancingo.

A continuación se explica de manera mas detallada la forma en la que realizó la medición de eficiencia de los algoritmos comparados durante el proceso de implementación física en el laboratorio de pruebas.

Figura **6-2**.: Área de pruebas físicas en el Laboratorio AyC.

6.1. Medición de eficiencia

El objetivo de la tarea de mapeo es explorar todas las zonas en un área cerrada, ésta tarea se completará cuando el conjunto B, definido por la ecuación 3-1, esté vacío. Luego se propone que la medición de la eficiencia de los algoritmos sea $n(B)$ después de que se ejecute un número de iteraciones en cada algoritmo. La figura **6-3** muestra un ejemplo del comportamiento de esta medición de eficiencia.

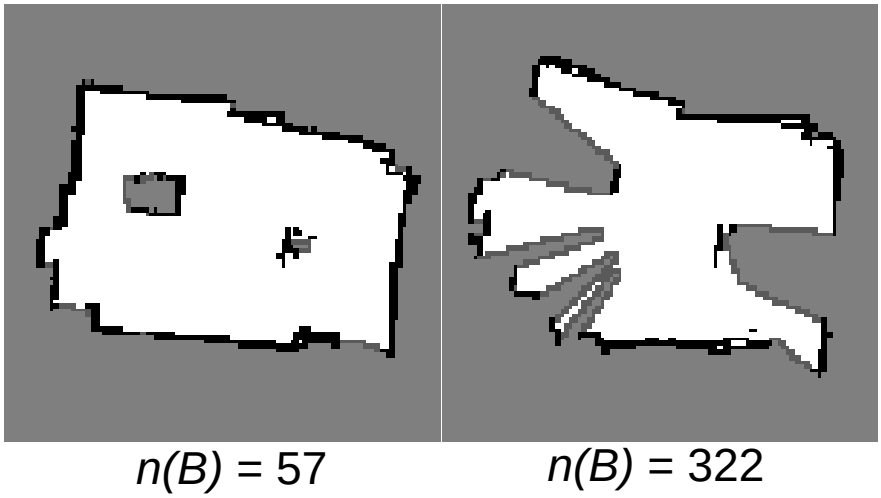


Figura **6-3**.: Comparación usando la medición del calculo de bordes.

Como se observa en la imagen, el mapa de la figura del lado izquierdo tiene un menor número de puntos en la frontera y el mapa está casi completo. Por otra parte si se analiza el mapa de la figura de la derecha se observa claramente que el mapa no está completo, y el

número de puntos que se encuentran en la frontera es mayor.

Por lo tanto, la medición de eficiencia de los algoritmos tomará como el mejor al que tenga un número menor de puntos en la frontera. A continuación se plantean dos comparaciones diferentes de acuerdo con este criterio. Los datos se presentan en forma de gráficas de distribución para observar el desempeño de cada método analizado.

Para realizar las pruebas fue necesario repetir cada algoritmo colocando el robot en la misma posición al iniciar cada uno de los experimentos, el robot se mueve el mismo número de ciclos para cada algoritmo y para cada prueba.

Los datos se obtuvieron utilizando el escritorio remoto conectado al robot mientras ejecuta su tarea, estos datos fueron capturados y almacenados para obtener su media y varianza, y de ese modo presentar la información obtenida en forma de gráficas para facilitar su interpretación.

6.2. Comparación usando ejecuciones independientes.

El uso de variables aleatorias en las técnicas heurísticas implica que cada vez que se ejecutan los algoritmos se obtendrá una solución diferente. Una forma de medir el rendimiento de un método heurístico es ejecutar el programa varias veces y calcular la solución promedio y la varianza de los resultados. Los resultados de esta comparación se muestran en la figura **6-4**.

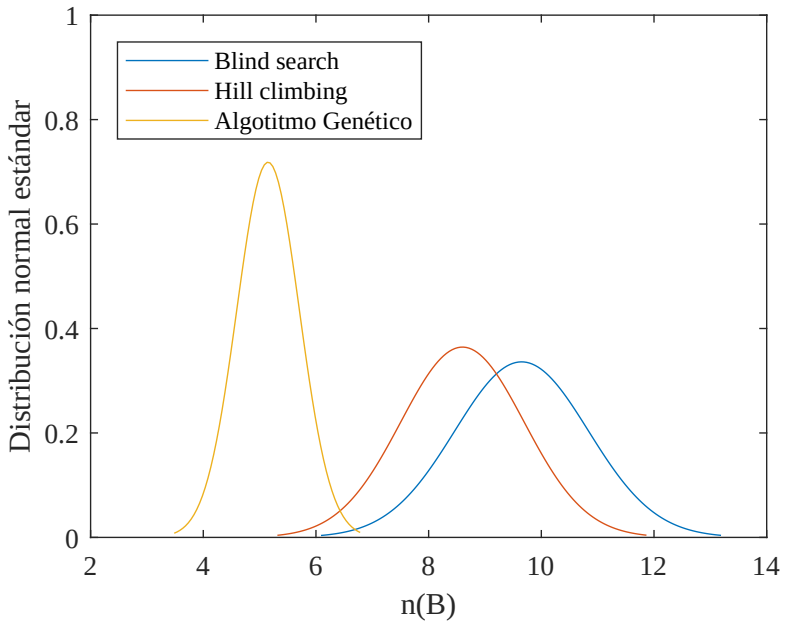


Figura **6-4**.: Comparación usando ejecuciones independientes.

Como se puede ver en la figura **6-4**, después de 30 experimentos realizados, el algoritmo

genético resulta con un promedio menor ya que la varianza del gráfico muestra una tendencia a cinco, que es menor en comparación con el algoritmo de escalada que tiene un promedio de ocho, o la búsqueda ciega con un promedio de diez, por esta razón se sabe que el algoritmo genético es el mejor en esta comparación.

6.3. Comparación usando la misma ejecución.

Es necesario recordar que el proceso de mapeo es una secuencia repetitiva de pasos alternando entre la generación de trayectoria y el movimiento, esto significa que una trayectoria subóptima elegida en algún punto afectaría a todo el siguiente proceso.

Hay muchos factores que pueden inducir error en el proceso de mapeo, por ejemplo, un retraso en la generación de mapeo, cambios en la iluminación, mala aproximación de la posición u odometría de los objetos, etc. Estas fuentes de error son independientes del algoritmo de optimización utilizado.

Por lo tanto, una vez analizado y comprendido el comportamiento anterior, se propone una segunda medición de eficiencia, esta vez todos los algoritmos se ejecutarán con la misma información de los sensores del robot, la mejor trayectoria será seguida por el robot y se compararán los algoritmos en función de cuántas veces cada algoritmo produce la mejor solución durante un único proceso de mapeo.

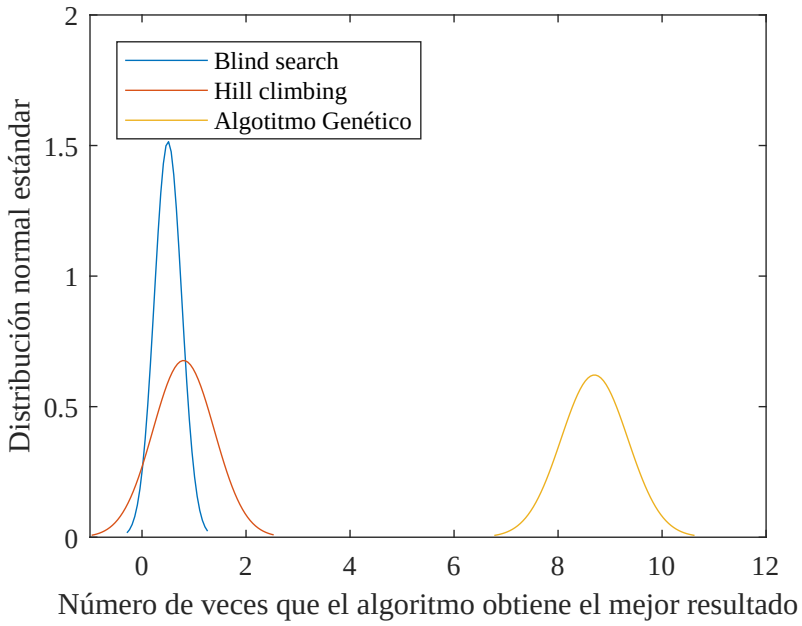


Figura **6-5**.: Comparación usando la misma ejecución.

En **6-5** se observan los resultados del segundo experimento, que muestra la cantidad promedio de veces que un algoritmo fue mejor que otro durante la misma iteración y las

mismas condiciones en la ejecución del mapeo. En este caso se observa que el algoritmo genético fue seleccionado un promedio de nueve veces durante la ejecución del proceso de mapeado, lo que indica que es mejor en comparación con el método de Hill Climbing que se seleccionó dos veces y el método de Blind Search que solamente fue seleccionado una vez.

6.4. Parámetros utilizados

Los resultados de ambas gráficas, mostrados en las figuras **6-4** y **6-5** se obtuvieron usando 30 experimentos de cada algoritmo. La comparación se realizó utilizando un número máximo de evaluación $M_{eva} = 100$ para todos los algoritmos

En el caso del método de Hill Climbing, las evaluaciones se dividieron en $N_{it} = 10$ número de iteraciones, utilizando $N_{ev} = 10$ número de evaluaciones por iteración.

El algoritmo genético fue implementado mediante la Selección de Torneo Binario, el Cruce de dos puntos con la tasa $C_r = 0{,}9$ y la Mutación uniforme con la tasa $M_r = 0{,}9$.

7. Conclusiones y trabajo a futuro

7.1. Conclusiones.

Elaborar este trabajo de tesis fue un reto muy grande, se requiere de esfuerzo y dedicación en diferentes aspectos, sobre todo adquirir conocimientos de programación en el sistema operativo de *ROS* y el aprender a usar la plataforma robótica *TurtleBot* ya que no se tenían conocimientos previos en estas áreas.

Fueron muchas horas de investigación y de trabajo duro que considero que valieron la pena, ver este trabajo terminado es para mí una gran recompensa y una forma de demostrar que si se puede.

Esta experiencia sirvió para mejorar mi preparación académica. Considero que quien quiera realizar un trabajo en esta área debería investigar no solo sobre robótica en general, sino también de temas específicos como programación en *ROS* y hacer pruebas de cada cosa que le resulte complicado para tener nociones mas claras y reforzar los conocimientos.

Los resultados de ambas pruebas muestran que el algoritmo con el menor número de puntos en el borde así como el que genera mejores resultados en mayor número de ocasiones es el algoritmo genético. Sin embargo, en las gráficas de las figuras **7-1** se pueden observar solapamientos en las funciones de probabilidad, lo que significa que en ocasiones alguno de los otros algoritmos puede ser mejor, aun así se observa que es poco probable.

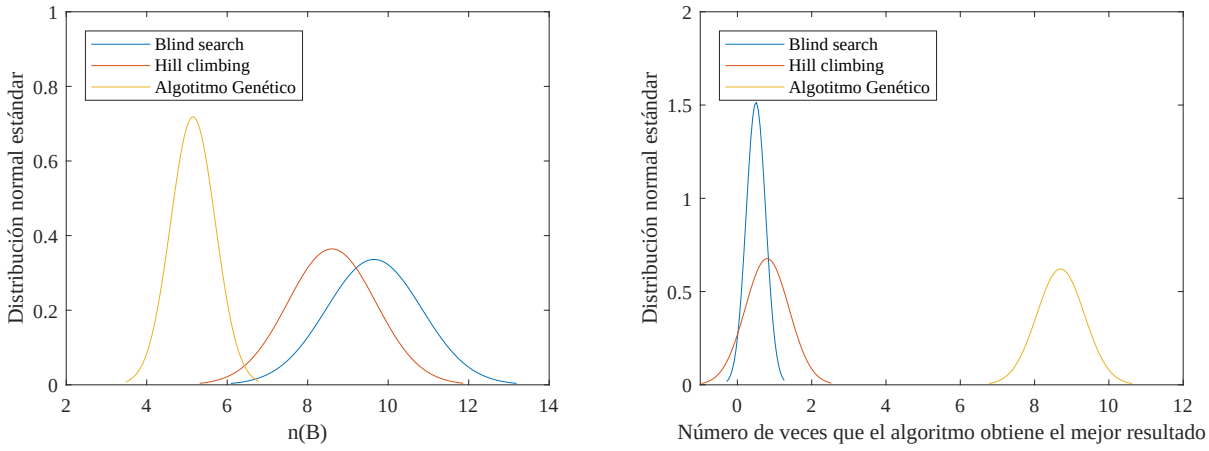


Figura **7-1**.: Comparación de métodos.

En algunas ocasiones, el robot perdía su ubicación debido a fallas de los sensores de visión y odometría, por lo que el mapa estaba sobrecargado y las rutas se generaron en lugares diferentes a la posición del robot, lo que provocó que colisionara. Por otra parte, este mismo problema también podría evitar que el mapa se actualice, por lo que el algoritmo no pudo encontrar las rutas adecuadas, lo que provocó que el robot siguiera girando.

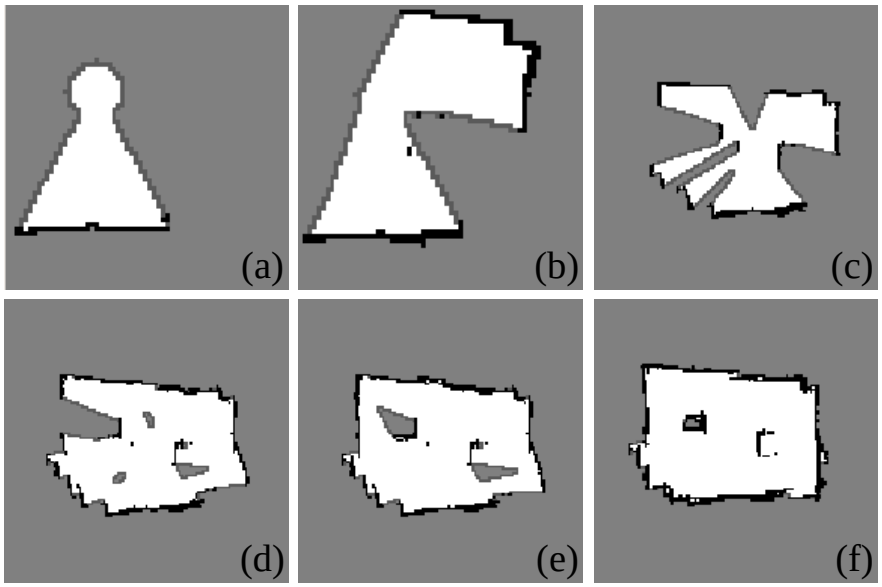


Figura **7-2**.: Creación del mapa.

La figura **7-2** ilustra la creación del mapa en seis imágenes, en la figura **7-2**.a se tiene la primera posición del robot que muestra solo una parte mapeada, esto se toma como punto de partida para generar las trayectorias, en las figuras **7-2**.b- **7-2**.e se muestran más puntos que completan el mapa, se agregan a medida que el robot toma decisiones de rotar o moverse a través de la zona libre una vez que se generan las trayectorias, finalmente en la figura **7-2**.f, el mapa completo se muestra después de varias iteraciones de este proceso.

7.2. Resultados

Este trabajo fue presentado en la Ciudad de México, en la *XV Conferencia internacional sobre ingeniería eléctrica, ciencias de la computación y control automático (CCE 2018)*, en las instalaciones del Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional (CINVESTAV).

Esta conferencia se realiza cada año, en esta ocasión del 5 al 7 de septiembre de 2018, y ofrece una oportunidad para que los investigadores y estudiantes se reúnan con colegas más experimentados de diferentes partes del mundo.

A continuación se mencionan los datos del artículo en caso de consultas posteriores, así como se muestra a continuación en la imagen de la figura **7-3**:

Figura **7-3**.: Página del artículo publicado.

El título del trabajo es:
"Optimal trajectories generation for autonomous navigation tasks in mobile robots"
y se encuentra disponible en la página web de IEEE Xplore:
https://ieeexplore.ieee.org/document/8533936/authors#authors
DOI: 10.1109/ICEEE.2018.8533936.

7.3. Trabajos futuros.

Como trabajo futuro se recomienda implementar otros algoritmos para seguir realizando pruebas de manera física y obtener así su comportamiento en tareas de generación óptima de trayectorias, algunos de los algoritmos que se proponen son:

- La exploración rápida de árboles aleatorios (RRT) [Lavalle, 1998].
- El método de Campos Potenciales [Koren y Borenstein, 1991].
- El método Símplex de Nelder-Mead [Wang y Shoup, 2011]

Debido a que se espera que la implementación de estos algoritmos permita realizar un nuevo análisis y observar su comportamiento a través de diferentes perspectivas para la solución de este tipo de problemas.

Se recomienda hacer otra comparativa de los algoritmos propuestos, ahora no solo para comparar la complejidad del mapa obtenido, sino también para evaluar la capacidad de cada uno de ellos tanto en eficiencia energética como en tiempo de respuesta y observar de esa manera las propiedades de cada uno.

Del mismo modo, se propone implementar dichos algoritmos en otras plataformas robóticas móviles, bajo el soporte de *ROS*, tal vez se podría probar en un robot aéreo o en un robot submarino dependiendo las capacidades propias de creación del mapa de cada uno, o en su defecto, se recomienda crear su propio algoritmo *gmapping* esto con la finalidad de tratar de mejorar sus características y propiedades para la tarea de mapeo ya que, como se mencionó anteriormente, se observaron varios errores al implementar el algoritmo de generación de rutas que no eran propios de los algoritmos evaluados, sino de la plataforma.

A. Anexo: Instalación de ROS en la Raspberry Pi

El primer paso para poder instalar este sistema ROS es tener una Raspberry Pi 3B, es importante que se tenga el modelo específico ya que no se asegura el funcionamiento en otro modelo o versión aunque se trate del mismo dispositivo.

Se debe tener una memoria micro SD con capacidad de almacenamiento de al menos 8 GB de memoria y clase 10 para garantizar que soportará el sistema operativo. De igual modo se requieren los periféricos de entradas y salidas necesarios para utilizar la Raspberry por primera vez, en este caso, el monitor y cable HDMI, teclado y mouse.

Una vez teniendo los elementos necesarios, debemos asegurarnos de que la señal de internet sea buena y además de que no este restringida o bloqueada, ya que en ocasiones no permite descargas de otros sistemas operativos y causa conflictos a la hora de instalar ROS.

A.1. Instalación de la imagen Ubuntu Mate 16.04

Ya que se ha revisado que exista lo necesario se procede a la instalación, en primer lugar del sistema operativo Ubuntu Mate Xenial, para la cual se requiere realizar la descarga desde el sitio oficial de Ubuntu.

La compilación de Ubuntu Mate 16.04 LTS puede ser descargada desde el sitio web oficial de Ubuntu (https://ubuntu-mate.org/download/), en este sitio se debe elegir la versión adecuada para la tarjeta Raspberry Pi.

A continuación se describen los pasos a seguir para cargar la imagen en la memoria SD:

1. En una computadora portátil o *Laptop* con Ubuntu insertar la memoria SD de al menos 8 GB clase 10.

2. Dar doble clic en el archivo **.img.xz* que se descargo de la página de Ubuntu, para abrir la aplicación de restauración de imágenes, seleccionar la ubicación de la memoria SD y aceptar.

3. Una vez terminado el proceso de carga, se debe expulsar la memoria SD de forma seguro e insertarla con la imagen ya cargada en la ranura de la Raspberry.

4. En esta parte es cuando se deben conectar por puerto USB el teclado y el muse, además del monitor con cable HDMI a la tarjeta para encenderla correctamente.

5. Una vez hecho esto se debe conectar la tarjeta a una fuente de alimentación de 5V y al menos 2A para proceder a encenderla y utilizarla de manera eficiente.

6. La primera vez que se enciende la Raspberry se configura el espacio libre de la memoria, este proceso puede tardar de 5 a 10 minutos.

A continuación se debe verificar que la conexión de internet no este restringida ni bloqueada para proceder con la instalación de las librerías y de ROS.

A.2. Instalación de ROS Kinetic

Una vez que la tarjeta está encendida y ya está lista con el sistema operativo se procede a seguir los siguientes pasos:

1. Ir a *"System"*, después a *"Administration"* y seleccionar *"Software&Updates"*.

2. Marcar las casillas *"restricted"*, *"universe"* y *"multiverse"* para permitir estos repositorios.

3. Abrimos una terminal presionando *Ctrt+t* y se procede a configurar *"sources.list"* escribiendo lo siguiente en la terminal:
   sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main"> /etc/apt/sources.list.d/ros-latest.list'

4. Ahora se configuran las llaves usando:
   wget http://packages.ros.org/ros.key -O - | sudo apt-key add -

5. Hay que asegurarse de que la compilación de Ubuntu Mate esta actualizada, para ello usamos:
   sudo apt-get update

6. Ahora se debe instalar "ros-kinetic-desktop-full":
   sudo apt-get install ros-kinetic-desktop-full

7. Ahora se debe inicializar "rosdep", entonces usamos las siguientes lineas:
   sudo rosdep init
   rosdep update

8. También se deben configurar las variables de ambiente de ROS usando:
   echo "source /opt/ros/kinetic/setup.bash">> ∼/.bashrc
   source ∼/.bashrc

9. Crear e inicializar el espacio de trabajo "Catkin":
   mkdir -p ∼/catkin_workspace/src
   cd catkin_workspace/src
   catkin_init_workspace
   cd ∼/catkin_workspace/
   catkin_make

10. Agregamos ahora el *"catkin_workspace"* al entorno ROS con:
    source ∼ / catkin_workspace / devel / setup.bash
    echo "source ∼ / catkin_workspace / devel / setup.bash" >> ∼ / .bashrc

11. Ahora se comprueban las variables de entorno ROS usando:
    exportacion grep ROS

12. Si no existe ningún error, la instalación ha finalizado correctamente.

Si se requiere, se puede comprobar que la instalación de ROS este correcta, para ello abrimos una terminal nueva y escribiremos *roscore* y dejamos que salgan todas las lineas de ejecución, después abrimos una nueva terminal y ahora se escribe *rosrun turtlesim turtlesim_node*. Debe salir una imagen con una tortuga, como se observa en la figura **A-1**:

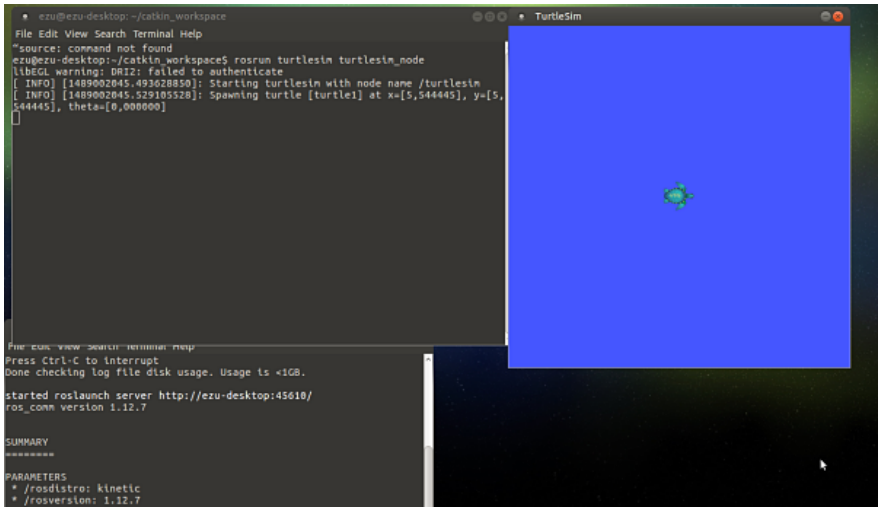


Figura **A-1**.: Se observa *"turtlesim"* operando en ROS.

El error que ve en la terminal (advertencia de libEGL:DRI2: no se pudo autenticar) generalmente se debe a que la asignación de memoria de gráficos en la Raspberry Pi es demasiado baja, pero ROS funciona.

A.3. Configuración de la red WiFi y conexión por puerto ssh

Lo primero que se debe hacer es crear una red WiFi como se haría normalmente, es decir, seleccionamos el icono de red WiFi en la parte superior derecha de la pantalla, seleccionamos *"Editar las conexiones"* como se muestra en la figura **A-2** y enseguida *"Añadir nueva red"*. Ahí se crea una red inalámbrica.

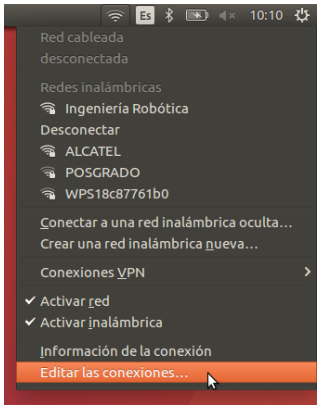


Figura **A-2**.: Creación de una red WiFi.

Ahora se configura la red como se muestra a continuación en la figura **A-3**. En la pestaña *"General"* hay que asegurarse de que ambas casillas estén seleccionadas.

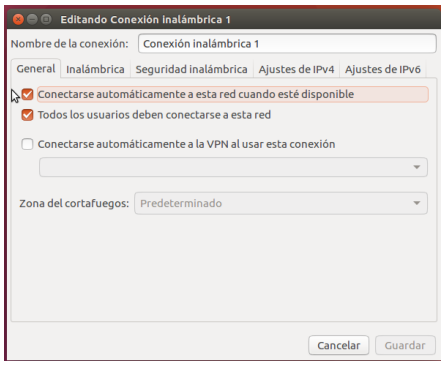


Figura **A-3**.: Editar conexión inalámbrica.

Ahora pasamos a la pestaña *"Inalámbrica"* como se observa en la figura **A-4**, ahí podemos elegir el nombre de la red a nuestro gusto, seleccionamos el modo como *"punto de acceso"* y dejamos las demás configuraciones como están predeterminadas.

Si se requiere que la red solicite una contraseña para dar acceso a ella, es necesario acceder a la pestaña *"Seguridad inalámbrica"* elegir del muenú desplegable de *"Seguridad"* la sexta

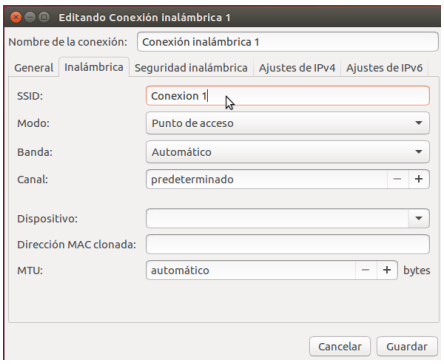


Figura **A-4**.: Editar conexión inalámbrica 2.

opción, en este caso es *"WPA y WPA2 personal"* como se observa en la figura **A-5**, después se configura la contraseña y listo, damos clic en *Guardar* la red.

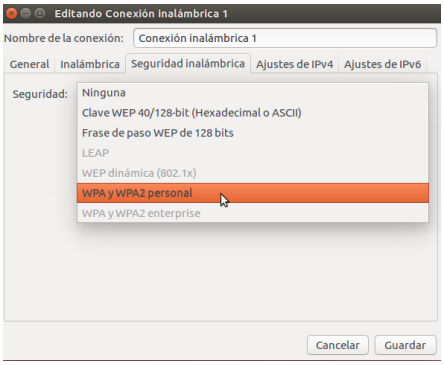


Figura **A-5**.: Editar conexión inalámbrica 3.

A.4. Habilitar el puerto SSH de la Raspberry Pi

Habilitar la conexión por medio del puerto *ssh* permite controlar la Raspberry desde una computadora remota, para ello es necesario ejecutar la siguiente línea de comando en una terminal del sistema, esto hará que la conexión remota sea posible:
sudo service ssh restart

Debido a que no es práctico realizar esta acción cada vez que el dispositivo se encienda es necesario crear un *script* que se ejecute automáticamente cada vez que se encienda la tarjeta. Este proceso se realiza como se muestra a continuación.

En primer lugar se debe crear un archivo con extensión *".sh"* y lo guardamos en nuestra carpeta personal. En este caso se le puso el nombre de *"puerto.sh"*, pero se le puede poner el nombre que gustes.

Abrimos el archivo y escribimos en la primera linea del código: *#!/bin/bash*, esta linea

nos permite que el documento sea reconocido como *script*, en la segunda linea del código ponemos la instrucción anterior para establecer la conexión remota, en este caso *sudo service ssh restart* y lo guardamos.

Abrimos una terminal y ejecutamos las siguientes lineas de código para copiar el documento a la carpeta *" etc"* y para otorgar los permisos para ejecutar este archivo, eso hará que el sistema lea y ejecute el archivo que hemos insertado en la carpeta.
sudo mv /home/usuario/puerto.sh /etc/init.d/
sudo chmod +x /etc/init.d/puerto.sh
sudo update-rc.d puerto.sh defaults 80

Con estas instrucciones ya es posible realizar la conexión remota con la Raspberry sin necesidad de usar periféricos extras ni configurar cada vez que se inicia.

A.5. Instalación de la librería de TurtleBot

Ahora finalmente queda instalar la paquetería de TurtleBot en la Raspberry, para lo cual debemos abrir una terminal y seguir los siguientes comandos:

- *sudo apt-get install -y linux-headers-generic*
- *sudo sh -c 'echo "deb-src http://us.archive.ubuntu.com/ubuntu/ xenial main restricted*
- *sudo apt-get update*
- *sudo apt-get install -y ros-kinetic-librealsense*
- *sudo apt-get install -y ros-kinetic-librealsense-camera*
- *sudo apt-get install -y ros-kinetic-turtlebot*

De este modo se obtiene la instalación completa. Si en algún momento marca errores relacionados con la fecha de compilación se puede ejecutar el siguiente código para actualizar la fecha de la tarjeta de forma manual solo cambiando los valores por la fecha y hora actuales:
date -s "03/26/2018 22:00"

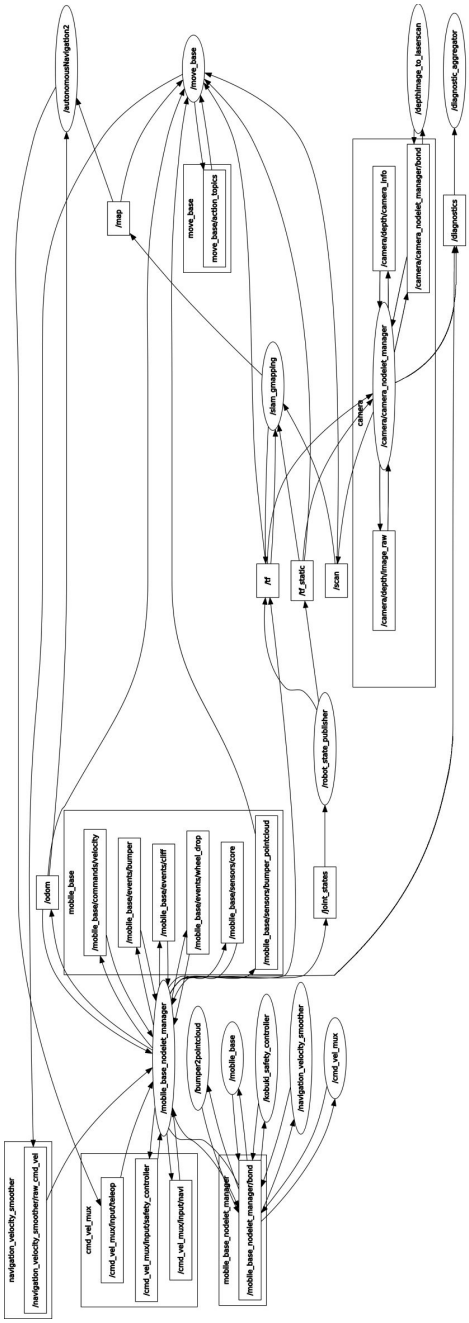
Si se tienen errores o se desea consultar más información puede visitar las siguientes páginas web que le ayudarán a resolver sus dudas.

1. Para instalar Ubuntu Mate Xenial:
https://ubuntu-mate.org/download/

2. Para instalar ROS Kinect:
https://www.intorobotics.com/how-to-install-ros-kinetic-on-raspberry-pi-3-ubuntu-mate/

3.Para instalar TurtleBot: *https://answers.ros.org/question/246015/installing-turtlebot-on-ros-kinetic/*

B. Anexo: Conexión de nodos en ROS



Bibliografía

Aagela, Hamza, Maha Al-Nesf y Violeta Holmes (2017). «An Asus_xtion_probased indoor MAPPING using a Raspberry Pi with Turtlebot robot Turtlebot robot». En: *Automation and Computing (ICAC), 2017 23rd International Conference on*. IEEE, págs. 1-5.

Acevedo, Hernando González y Carlos Alberto Mejia Castañeda (2007). «Estudio comparativo de tres técnicas de navegación para robots móviles». En: *Revista UIS Ingenierias* 6.1, págs. 77-88.

Ahmed Omara, H. I. y K. S. Mohamed Sahari (2015). «Indoor Mapping using Kinect and ROS». En: *International Symposium on Agents, Multi-agent Systems and Robotics (ISAMSR)*, págs. 110-116.

Andrade, Federico y L Martin (2007). *SLAM Estado del arte*.

Armstrong, DG y CD Crane (1998). «A modular, scalable, architecture for intelligent, autonomous vehicles». En: *3rd IFAC Symposium on Intelligent Autonomous Vehicles, Universidad Carlos III, Madrid, Spain*, págs. 62-66.

Bailey, Tim y Hugh Durrant-Whyte (2006). «Simultaneous localization and mapping (SLAM): Part II». En: *IEEE Robotics & Automation Magazine* 13.3, págs. 108-117.

Ballesta, M et al. (2010). «Analisis de detectores y descriptores de caracteristicas visuales en slam en entornos interiores y exteriores». En: *Revista Iberoamericana de Automática e Informática Industrial RIAI* 7.2, págs. 68-80.

Barraquand, Jerome y Jean-Claude Latombe (1991). «Robot motion planning: A distributed representation approach». En: *The International Journal of Robotics Research* 10.6, págs. 628-649.

Blaza, David y Semiconductor Marketing Insider Tech (2015). «It's not just Raspberry Pi». En: URL: `https://www.embedded.com/electronics-blogs/say-what-/4439231/It-s-not-just-Raspberry-Pi` (visitado 14-11-2018).

Bledsoe, Woodrow W (1961). «The use of biological concepts in the analytical study of systems». En: *ORSA-TIMS national meeting*.

Boucher, Sol (2012). «Obstacle detection and avoidance using turtlebot platform and xbox kinect». En: *Department of Computer Science, Rochester Institute of Technology* 56.

Box, George EP (1957). «Evolutionary operation: A method for increasing industrial productivity». En: *Applied statistics*, págs. 81-101.

Bratko, Ivan (2001). *Prolog programming for artificial intelligence*. Pearson education.

Bravo, Salao y José Raúl (2010). «Estudio de las Técnicas de Inteligencia Artificial mediante el apoyo de un Software Educativo». B.S. thesis.

Bremermann, Hans J (1962). «Optimization through evolution and recombination». En: *Self-organizing systems* 93, pág. 106.

Čapek, Karel (1968). *Divadelníkem proti své vůli: recenze, stati, kresby, fotografie*. Vol. 8. Orbis.

Capito, Linda et al. (2016). «Experimental comparison of control strategies for trajectory tracking for mobile robots». En: *International Journal of Automation and Control* 10.3, págs. 308-327.

Carreras, Marc et al. (2012). «Inspección visual subacuática mediante robótica submarina». En: *Revista Iberoamericana de Automática e Informática industrial* 9.1, págs. 34-45. ISSN: 1697-7920. DOI: 10.1016/j.riai.2011.11.011. URL: https://polipapers.upv.es/index.php/RIAI/article/view/9626.

Cheein, F et al. (2010). «Slam-based cross-a-door solution approach for a robotic wheelchair». En: *International Journal of Advanced Robotic Systems* 7.2, págs. 155-164.

Chong, Edwin KP y Stanislaw H Zak (2013). *An introduction to optimization*. Vol. 76. John Wiley & Sons.

Chui, M., J. Manyika y M. Miremadi (2015). «Four fundamentals of workplace automation». En: *McKinseyQarterly*.

Claessens, Rik, Yannick Müller y Benjamin Schnieders (2013). *Graph-based Simultaneous Localization and Mapping on the TurtleBot platform*.

Cohen, William, Russell Greiner y Dale Schuurmans (1991). «Probabilistic hill climbing». En: *Proceedings of CLNL-91, Berkeley*.

Correa, C. y L. Vásquez (2012). «Algoritmos para la planificación y seguimiento de trayectorias en robots agrícolas». En: *X Congreso Latinoamericano y del Caribe de Ingeniería Agrícola e XLI Congresso Brasileiro de Engenharia Agrícola CLIA/CONBEA 2012*.

Correa Farias, Christian y Lorenzo Vásquez Alfaro (2012). «Algoritmos de Planificación y Seguimiento de Trayectorias para Robots Agrícolas». En:

Cruz, Pedro Ponce y Alejandro Herrera (2011). *Inteligencia artificial con aplicaciones a la ingeniería*. Vol. 1. Marcombo.

Cuchango, Helbert Eduardo Espitia y Jorge Sofrony Esmeral (2012). «Algoritmo para planear trayectorias de robots móviles, empleando campos potenciales y enjambres de partículas activas brownianas». En: *Ciencia e Ingeniería Neogranadina* 22.2, pág. 3.

De Jong, Kenneth Alan (1975). «Analysis of the behavior of a class of genetic adaptive systems». En:

Dissanayake, Gamini, Hugh Durrant-Whyte y Tim Bailey (2000). «A computationally efficient solution to the simultaneous localisation and map building (SLAM) problem». En: *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*. Vol. 2. IEEE, págs. 1009-1014.

Dissanayake, MWM Gamini et al. (2001). «A solution to the simultaneous localization and map building (SLAM) problem». En: *IEEE Transactions on robotics and automation* 17.3, págs. 229-241.

Duan, H. y P. Qiao (2015). «Pigeon-inspired optimization: a new swarm intelligence optimizer for air robot path planning». En: *International Journal of Intelligent Computing and Cybernetics* 7, págs. 24-37.

Duque, Juan Pablo Urrea y Emmanuel Ospina (2004). «Implemetación de la transformada de Hough para la detección de lineas para un sistema de visión de bajo nivel.» En: *Scientia et technica* 1.24, págs. 79-84.

Durrant-Whyte, Hugh y Tim Bailey (2006). «Simultaneous localization and mapping: part I». En: *IEEE robotics & automation magazine* 13.2, págs. 99-110.

Elfes, Alberto (1989). «Using occupancy grids for mobile robot perception and navigation». En: *Computer* 6, págs. 46-57.

Elizondo, J. C. et al. (2016). «Multi-robot Exploration Using Self-Biddings under Constraints on Communication Range». En: *IEEE Latin America Transactions* 14.

Engelbrecht, Andries P (2006). *Fundamentals of computational swarm intelligence*. John Wiley & Sons.

Fernández, Yerai Berenguer (2018). «Uso de descriptores holísticos para la localización y creación de mapas: una aproximación al graph-slam mediante apariencia visual». Tesis doct. Universidad Miguel Hernández de Elche.

Fernández-Vega, Iván et al. (2017). «Desarrollo de un entorno de programación para un robot simulado Turtlebot-2 con brazo manipulador WIDOWX mediante la conexión V-REP y MATLAB». En:

Fogel, Lawrence J, Alvin J Owens y Michael J Walsh (1966). «Artificial intelligence through simulated evolution». En:

Friedman, George J (1959). «Digital simulation of an Evlutionary Process». En: *General Systems Yearbook*, págs. 171-184.

Fu, King Sun, Ralph Gonzalez y CS George Lee (1987). *Robotics: Control Sensing. Vis.* Tata McGraw-Hill Education.

Galdeano, Mónica Ballesta (2011). «Slam visual cooperativo mediante fusión de mapas 3d adquiridos por robots móviles». Tesis doct. Universidad Miguel Hernández de Elche.

Gallardo, Nicolas et al. (2016). «Formation control implementation using Kobuki TurtleBots and Parrot Bebop drone». En: *World Automation Congress (WAC), 2016*. IEEE, págs. 1-6.

Ganeshmurthy, M. S. y G. R. Suresh (2015). «Path planning algorithm for autonomous mobile robot in dynamic environment». En: *2015 3rd International Conference on Signal Processing*, págs. 1-6.

García, Miguel Ángel Gutiérrez (2004). «Cálculo de caminos óptimos para manipuladores articulados». En: *Departamento de Informática y Automática. Salamanca, Universidad de Salamanca*.

Georgoulas, Christos et al. (2012). «An AmI environment implementation: Embedding TurtleBot into a novel robotic service wall». En: *Robotics; Proceedings of ROBOTIK 2012; 7th German Conference on*. VDE, págs. 1-6.

Gerkey, Brian P, Sebastian Thrun y Geoff Gordon (2005). «Parallel stochastic hill-climbing with small teams». En: *Multi-Robot Systems. From Swarms to Intelligent Automata Volume III*. Springer, págs. 65-77.

Gil, A et al. (2007). «Influencia de los parámetros de un filtro de partículas en la solución al problema de SLAM». En: *IEEE Latin America* 6.1, págs. 18-27.

Gonzáles, V y R Parkin (2005). «Evadiendo Obstáculos con Robots Móviles». En: *Revista Digital Universitaria, Universidad Nacional Autónoma de México* 6.1.

González, Rafael C y Richard E Woods (1996). *Tratamiento digital de imágenes*. Vol. 3. Addison-Wesley New York.

González Arjona, D. (2011). «Generación automática de mapas en espacios cerrados mediante robots móviles». En: *Universidad Autónoma de Madrid: Escuela Politécnica Superior*.

Graetz, G. y G. Michaels (2015). «Robots at Work». En: *Discussion paper series* IZA DP No. 8938.

Guzmán Luna, J. A., R. E. Arango Sánchez y L. D. Jiménez Pinzón (2012). «Búsqueda de la ruta óptima mediante los algoritmos: genético y dijkstra utilizando mapas de visibilidad». En: *Scientia et Technica Año XVII. Universidad Tecnológica de Pereira* 51, págs. 107-112.

Hamzeh, O. y A. Elnagar (2015). «A Kinect-based Indoor Mobile Robot Localization». En: *IEEE*.

Holland, J. H. (1992). «Complex Adaptive Systems». En: *Daedalus, JSTOR* 121, págs. 17-30.

Hoy, M., A. S. Matveev y A. V. Savkin (2015). «Algorithms for collision-free navigation of mobile robots in complex cluttered environments: a survey». En: *Cambridge University Press 2014* 33, págs. 463-497.

Hulbert, Caleb (2016). «Biologically Inspired Autonomous Robotic Navigation Using High-Level Object Detection». En:

Hwang, Yong y Narendra Ahuja (1989). «Path planning using a potential field representation». En: *Computer Vision and Pattern Recognition, 1989. Proceedings CVPR'89., IEEE Computer Society Conference on*. IEEE, págs. 569-575.

Ioakeimidis, Paris A (2018). «Integration of ROS, Turtlebot, RPLIDAR, RFID technologies and algorithm implementation for navigation and RFID tag detection in a warehouse.» En:

Kaczmarek, W et al. (2018). «Industrial and ROS-enabled mobile robots cooperation». En: *24 th International Conference ENGINEERING MECHANICS 2018 Svratka* 74, 349–352. DOI: 10.21495/91-8-349.

Kenneth, R (1979). «Castleman Digital Image Processing, 1996 Prentice-Hall». En: *Chapter* 14, págs. 313-346.

Khatib, Oussama (1986). «Real-time obstacle avoidance for manipulators and mobile robots». En: *Autonomous robot vehicles*. Springer, págs. 396-404.

Koditschek, Daniel (1987). «Exact robot navigation by means of potential functions: Some topological considerations». En: *Robotics and Automation. Proceedings. 1987 IEEE International Conference on*. Vol. 4. IEEE, págs. 1-6.

Koren, Y. y J. Borenstein (1991). «Potential field methods and their inherent limitations for mobile robot navigation». En: *1991 IEEE International Conference on Robotics and Automation, Sacramento* 2, págs. 1398-1404.

Korf, Richard E (2002). «A new algorithm for optimal bin packing». En: *AAAI/IAAI*, págs. 731-736.

Lapeña, P et al. (2014). «Swarm Algorithm Implementation in Mobile Robots for Security and Surveillance». En: *TENCON 2014 - 2014 IEEE Region 10 Conference*, págs. 1-5.

Latombe, JC (1991). «Robot motion planning Kluwer Academic Publishers Boston Google Scholar». En:

Laumond, J-P et al. (1994). «A motion planner for nonholonomic mobile robots». En: *IEEE Transactions on Robotics and Automation* 10.5, págs. 577-593.

Lavalle, S. M. (1998). «Rapidly-Exploring Random Trees: A New Tool for Path Planning». En: *Iowa State University*.

LaValle, Steven M (1998). «Rapidly-exploring random trees: A new tool for path planning». En:

Liang, Y. et al. (2017). «Optimization of robot path planning parameters based on genetic algorithm». En: *2017 IEEE International Conference on Real-time Computing and Robotics (RCAR)*, págs. 529-534.

López, D et al. (2010). «Planificación de trayectorias con el algoritmo RRT. Aplicación a robots no holónomos». En: *Revista Iberoamericana de Automática e Informática Industrial* 3.3, págs. 56-67.

López Torres, Patricia (2016). «Análisis de algoritmos para localización y mapeado simultáneo de objetos». En:

López, D. et al. (2006). «Planificación de trayectorias con el algoritmo RRT. Aplicación a robots no holónomos.» En: *Revista Iberoamericana de Automática e Informática Industrial*, págs. 56-67.

Marczyk, Adam (2004). «Genetic algorithms and evolutionary computation». En: *The Talk Origins Archive: http://www. talkorigins/faqs/genalg/genalg. html*.

Melián, Belén, José A Moreno Pérez y J Marcos Moreno Vega (2003). «Metaheurísticas: Una visión global». En: *Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial* 7.19, pág. 0.

Milford, Michael y Gordon Wyeth (2007). «Spatial mapping and map exploitation: a bio-inspired engineering perspective». En: *International Conference on Spatial Information Theory*. Springer, págs. 203-221.

Moya Carrasco, Á. (2016). «Generación de trayectorias en tiempo real a partir de diagramas de Voronoi». En: *Universidad de Sevilla: Escuela Técnica Superior de Ingeniería*.

Muñoz, V (1995). «Planificación de trayectorias para Robots móviles». Tesis doct. Tesis Doctoral. Universidad de Málaga.

Nakadate, R. y M. Hashizume J. Arata (2015). «Next-generation robotic surgery - from the aspect of surgical robots developed by industry». En: *Center for Advanced Medical Innovation*.

Nehmzow, U. (2012). *Mobile Robotics: A Practical Introduction*. Springer London. ISBN: 9781447100256. URL: https://books.google.com.mx/books?id=hrjkBwAAQBAJ.

Nguyen, Van-Duc (2012). «Obstacle avoidance using the Kinect». En: DOI: https://es.scribd.com/document/80464002/Obstacle-Avoidance-Using-the-Kinect.

Olivera, Alfredo (2004). «Heurísticas para problemas de ruteo de vehículos». En: *Reportes Técnicos 04-08*.

Park, FC y Bahram Ravani (1995). «Bezier curves on Riemannian manifolds and Lie groups with kinematics applications». En: *Journal of Mechanical Design* 117.1, págs. 36-40.

Parra Tsunekawa, S. I. (2014). «Generación de mapa de entorno para navegación de vehiculo autónomo terrestre.» En: *Universidad de Chile: Facultad de Ciencias Físicas y Matemáticas.*

Pérez, Lucia Hilario (2012). «Técnicas computacionales y álgebra tensorial para su aplicación en robótica móvil y simulación numérica». Tesis doct. Universidad CEU-Cardenal Herrera.

Pierce, R. M. (2015). «Increasing Transparency and Presence of Teleoperation Systems Through Human-Centered Design». En: *Publicly Accessible Penn Dissertations 1947*.

Porta García, Miguel Ángel (2007). «Planeación de trayectorias para robótica móvil mediante optimización por colonias de hormigas». Tesis doct. Instituto Politécnico Nacional.

Prabha, S Sathya et al. (2014). «Smart cloud robot using raspberry Pi». En: *Recent Trends in Information Technology (ICRTIT), 2014 International Conference on*. IEEE, págs. 1-5.

Quinonez, Y., I.Tostado y C. Burgueno (2015). «Aplicación de técnicas evolutivas y visión por computadora para navegación autónoma de robots utilizando un TurtleBot 2». En: *RISTI [online] n.spe3*, págs. 93-105.

Quiñonez, Yadira, Iván Tostado y Carlos Burgueño (2015). «Aplicación de técnicas evolutivas y visión por computadora para navegación autónoma de robots utilizando un TurtleBot 2». En: *RISTI-Revista Ibérica de Sistemas e Tecnologias de Informação* SPE3, págs. 93-105.

Randy, L (2004). «Haupt, Sue Ellen Haupt,"Practical genetic algorithmsPublished by John Wiley & Sons, Inc.» En:

Rascón Crespo, J. F. (2014). «Implementación de un algoritmo de seguimiento de trayectorias en el framework ROS». En: *Universidad Carlos III de Madrid: Escuela Politécnica Superior*.

Raspberry, Pi (3). «model B». En: *Raspberrypi.org: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/. Hakupäivä* 11, pág. 2018.

Ray, Partha Pratim, Lekhika Chettri y Nishant Thapa (2018). «IoRCar: IoT Supported Autonomic Robotic Movement and Control». En: *2018 Internat2018 International Conference on Computation of Power, Energy, Information and Communication (ICCPEIC) ional conference on computation of power, energy, Information and Communication (ICCPEIC)*. IEEE, págs. 077-083.

Rechenberg, Ingo (1965). «Cybernetic solution path of an experimental problem». En: *Royal Aircraft Establishment Library Translation 1122*.

Restrepo, Carlos et al. (2009). «Localización y construcción de un mapa en forma simultánea utilizando el Filtro Extendido de Kalman». En: *Revista INGENIERÍA UC* 16.1.

Riisgaard, Søren y Morten Rufus Blas (2004). «SLAM for Dummies A Tutorial Approach to Simultaneous Localization and Mapping By the 'dummies'». En:

Romero, Juan José Flores (2016). «Búsqueda a Ciegas (No Informada)». En: 4.

Russell, Stuart J y Peter Norvig (2004). *Inteligencia Artificial: un enfoque moderno*. Vol. 2. Pearson Education, pág. 1242.

Sanchez, Celso Marquez et al. (2016). «Trajectory generation for wheeled mobile robots via Bézier polynomials». En: *IEEE Latin America Transactions* 14.11, págs. 4482-4490.

Schwager, M. et al. (2017). «A Multi-robot Control Policy for Information Gathering in the Presence of Unknown Hazards». En: *Springer International Publishing Switzerland*, págs. 455-472.

Siegwart, R. et al. (2011). *Introduction to Autonomous Mobile Robots*. Intelligent robotics and autonomous agents. MIT Press. ISBN: 9780262015356. URL: `https://books.google.com.mx/books?id=4of6AQAAQBAJ`.

Skiena, Steven S (1998). *The algorithm design manual: Text*. Vol. 1. Springer Science & Business Media.

Sobhaninehad, Gholamreza, Assadollah Noorzad y Anooshirvan Ansari (2007). «Genetic algorithm (GA): A new approach in estimating strong ground motion attenuation relations». En: *Proceedings of the Fourth International Conference on Earthquake Geotechnical Engineering*.

Soille, Pierre (2013). *Morphological image analysis: principles and applications*. Springer Science & Business Media.

Song, B., Z. Wang y L. Sheng (2016). «A new genetic algorithm approach to smooth path planning for mobile robots». En: *Assembly Automation* 36, págs. 138-145.

Sosa Cortés, Luis Raúl (2017). «Control *PID* para el seguimiento de trayectoria de vehículos autónomos mediante algoritmos bio-inspirados». Tesis de mtría. Universidad Politécnica de Tulancingo.

Thrun, S. et al. (2005). *Probabilistic Robotics*. Intelligent robotics and autonomous agents. MIT Press. ISBN: 9780262201629. URL: `https://books.google.com.mx/books?id=2Zn6AQAAQBAJ`.

Thrun, Sebastian, Wolfram Burgard y Dieter Fox (1998). «A Probabilistic Approach to Concurrent Mapping and Localization for Mobile Robots». En: *Autonomous Robots* 5.3,

págs. 253-271. ISSN: 1573-7527. DOI: 10.1023/A:1008806205438. URL: https://doi.org/10.1023/A:1008806205438.

Torres, Wuilian J. y Roger J. Bello (2002). «Procesamiento de imágenes a color utilizando morfología matemática». En:

Torres Soler, Luis Carlos y Néstor Manuel Garzón Torres (2015). «La inteligencia artificial y la resolución de problemas». En: *Revista Clepsidra* 11.20, págs. 75-88.

Trevelyan, J., W. R. Hamel y S. C. Kang (2016). «Robotics in Hazardous Applications». En: *Springer Handbook of Robotics*.

Velásquez Hernández, C. A., J. J. Chávez Chávez y E. Córdoba Nieto (2015). «Implementación de sistema de navegación autónomo en robot móvil experimental para reconstrucción y exploración de entornos desconocidos». En: *Congreso Internacional de Ingeniería Mecatrónica y Automatización - CIIMA* IV, págs. 73-84.

Viñals, J (2012). «Localización y generación de mapas del entorno (SLAM) de un robot por medio de una Kinect». En: *Universidad Politécnica de Valencia*.

Vincentelli S., Guillermo J. y Troy H. Hughes (2016). «Roaming Spirit: A Framework for Localization and Navigation on a Raspberry Pi». En:

Wang, P.C. y T.E. Shoup (2011). «Parameter sensitivity study of the Nelder–Mead Simplex Method». En: *Advances in Engineering Software* 42, págs. 529-533.

Yu, Wen, Erik Zamora y Alberto Soria (2016). «Ellipsoid SLAM: a novel set membership method for simultaneous localization and mapping». En: *Autonomous Robots* 40, págs. 125-137. DOI: 10.1007/s10514-015-9447-y.

Zannatha Juárez, Angel Gabriel (2010). *Generación automática de mapas de entorno usando un robot móvil con visión por computadora*. Tesis de maestría IPN.