UNIVERSIDAD TECNOLÓGICA CENTROAMERICANA

FACULTAD DE INGENIERÍA MECATRÓNICA

PROYECTO:

DESARROLLO DE ROBOT AUTÓNOMO DE LOCALIZACIÓN Y MAPEO SIMULTÁNEO UTILIZANDO UN SENSOR *LIDAR 2D*

PREVIO A LA OBTENCIÓN DEL TÍTULO:

INGENIERO EN MECATRÓNICA

PRESENTADO POR:

21751087 ALBERTO JOSÉ CHINCHILLA SANTOS

41511208 JOSÉ MANUEL MURILLO CASTELLANOS

ASESOR: ING. ALICIA MARÍA REYES DUKE

CAMPUS SAN PEDRO SULA.

DEDICATORIA

JOSÉ MANUEL MURILLO CASTELLANOS

Dedico esta investigación primeramente a Dios y a mis padres José Manuel Murillo y Mirian Lucia Castellanos Salgado quienes me dieron la vida, educación, apoyo y consejos. A Emily Daniela Figueroa Cantarero que gracias a su apoyo emocional e incondicional encontré el camino correcto en los días difíciles, cuando nadie más creía en mí no dudo y siguió creyendo en que podría lograr mis metas y objetivos, mostrando su orgullo hacia mi persona cuando se ameritaba y cuando no, ayudándome de nuevo a esclarecer las dudas. A todos mis hermanos y hermanas que siempre estuvieron cuando más lo necesite. A mi compañero de investigación Alberto José Chinchilla Santos por haberme permitido realizar el proyecto de investigación junto a él, un gran colega y amigo, de las mejores personas que he conocido. A todos y cada una de las personas antes mencionadas les agradezco desde el fondo de mi alma.

ALBERTO JOSÉ CHINCHILLA SANTOS

Dedico esta investigación primeramente a Dios y a mis padres Jorge y Verónica quienes me han dado la vida, amor incondicional, educación, valores, consejos y apoyo a lo largo de toda mi vida y carrera universitaria. A mis dos hermanos Jorge y Josué que siempre han estado apoyándome y motivándome para ser cada día mejor. A toda mi familia que siempre ha creído en mí, tíos, primos, abuelos y en especial a mi abuela Victoria que cuida de mí desde el cielo. A mis amigos por su apoyo incondicional y por último a mi compañero de investigación Manuel por su gran apoyo a lo largo de mi carrera universitaria.

AGRADECIMIENTOS

Agradecemos a nuestros padres por el apoyo y confianza depositada en nosotros para poder seguir nuestros sueños. Agradecemos al ingeniero Josué Pérez y a la ingeniera Alicia Reyes por apoyarnos en la presente investigación.

Agradecemos a todos los docentes de UNITEC que nos brindaron sus conocimientos a lo largo de nuestra carrera.

EPÍGRAFE

"El aspecto más triste de la vida actual es que la ciencia gana en conocimiento más rápidamente que la sociedad en sabiduría."
-Isaac Asimov

RESUMEN EJECUTIVO

En la actualidad la necesidad y uso de los robots autónomos crece exponencialmente tanto en el sector industrial como en el cotidiano, debido a ello surge la implementación de sistemas que faciliten al robot poder operar de manera autónoma y que, a su vez, realice un modelo gráfico del entorno en el cual se moviliza. Con ello surge *SLAM,* los sistemas *SLAM* permiten localizar en tiempo real el robot sin la necesidad de tecnologías como el *GPS* y a su vez realizan un modelo gráfico muy acertado a la realidad. Hay diferentes maneras de implementar un sistema *SLAM*, una de las más populares es haciendo uso de sensores *LIDAR* que mediante el principio de triangulación láser generan la retroalimentación necesaria para que el robot detecte las superficies, objetos, obstáculos y distancia entre ellos. El siguiente proyecto de investigación se realizó con el objetivo de desarrollar un robot autónomo móvil que permita realizar mapas acertados a la realidad y se pueda localizar en tiempo real y de manera precisa, haciendo uso de *ROS* para la comunicación entre los actuadores, microcontroladores y sensores. El diseño de la estructura y sus estudios se han realizado con un *software CAD,* la selección de componentes se realizó con el uso de matrices de multicriterio y decisión, las cuales ayudaron a seleccionar los que mejor se adaptan al proyecto. Se planteó y se demostró que el uso de este sensor es una excelente opción para implementar *SLAM* y con ello desarrollar robots autónomos.

Palabras claves – *LIDAR,* robot autónomo, robot móvil, *ROS, SLAM*.

ABSTRACT

Currently the need and use of autonomous robots is growing exponentially both in the industrial sector and in everyday life, due to this arises the implementation of systems that facilitate the robot to operate autonomously and, in turn, make a graphical model of the environment in which it is mobilized. SLAM systems allow the robot to be located in real time without the need for technologies such as GPS, and at the same time they create a graphic model that is very accurate to reality. There are different ways to implement a SLAM system, one of the most popular is by using LIDAR sensors that through the principle of laser triangulation generate the necessary feedback for the robot to detect surfaces, objects, obstacles and distance between them. The following research project was carried out with the aim of developing a mobile autonomous robot that allows accurate maps to reality and can be located in real time and accurately, using ROS for communication between actuators, microcontrollers and sensors. The design of the structure and its studies have been carried out with CAD software, the selection of components was made with the use of multicriteria and decision matrices, which helped to select those that best fit the project. It was proposed and demonstrated that the use of this sensor is an excellent option to implement SLAM and thus develop autonomous robots.

Keywords - Autonomous robot, *LIDAR,* mobile robot, *ROS, SLAM*.

ÍNDICE DE CONTENIDO

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

ÍNDICE DE ECUACIONES

LISTA DE SIGLAS Y GLOSARIO

2D: Dos dimensiones.

3D: Tres dimensiones.

CNC: *Computer Numerical Control.*

CUDA cores*: Compute Unified Device Architecture*

***Framework*:** Espacio de trabajo o sistema.

GPS: *Global positioning system.*

I2C: *Inter integrated circuits.*

IDE: *Integrated development environment.*

LIDAR: *Light detection and ranging.*

PID: Proporcional, integral y derivativo.

ROS: *Robot operating system.*

SLAM: *Simultaneous localization and mapping.*

I. INTRODUCCIÓN

Los robots autónomos son caso de estudio muy común puesto que cada día se requiere más de ellos tanto en el sector industrial como en el cotidiano, ya que éstos permiten optimizar procesos, ahorrar tiempo, disminuir esfuerzo humano y requieren de una supervisión mínima o nula. Los robots para poder operar de forma autónoma necesitan apoyarse de sensores, cámaras, módulos localizadores, etcétera. Estos dispositivos son lo que le permiten poder evitar obstáculos para que puedan realizar sus recorridos de forma correcta. Una movilidad óptima se deriva de una localización precisa de los robots. Es por ello qué, la presente investigación trata de implementar un sistema *SLAM* basado en un sensor *LIDAR 2D* para el desarrollo de un robot autónomo. Con este sistema el robot logrará evitar obstáculos y obtener una localización exacta dentro del entorno en el que opera, además de esto, proporcionará información necesaria para un mapeo exacto en interiores, todo esto de forma simultánea. Después del mapeo realizado el robot podrá viajar al punto que se requiera de una forma fluida y precisa. El funcionamiento del robot se basará en un *framework* que permita utilizar sensores robustos y variedad de lenguajes de programación, debido a ello se ha decidido implementar *ROS*.

ROS se puede definir como un ecosistema el cual se encarga de comunicar los distintos actuadores y sensores sin importar el lenguaje de programación que éstos poseen. Un robot con este tipo de sistema puede ser diseñado para diferentes usos, como ser: robot carguero, robot de exploración, robot optimizador, robot medidor, etcétera.

A continuación, se presentan los capítulos que conforman la presente tesis y una breve descripción de los mismos:

En el Capítulo II, se presentarán los procedentes del problema, las preguntas de investigación y los objetivos que limitarán la investigación.

El Capítulo III presenta, análisis de la situación actual y definirá las teorías de sustento.

El Capítulo IV: Abordará la metodología que sigue esta investigación, definirá el enfoque, el tipo de alcance, las variables de estudio, hipótesis y la metodología en "V" empleada en la investigación.

El Capítulo V, analizará los resultados obtenidos de las simulaciones y las pruebas realizadas al prototipo de acuerdo con los objetivos planteados.

Por último, Capítulo VI: En este capítulo se presentan las conclusiones a partir de los resultados obtenidos acorde a los objetivos planteados y el capítulo VII en el cuál presentarán las recomendaciones a partir de las conclusiones obtenidas.

II. PLANTEAMIENTO DEL PROBLEMA

En este capítulo se mostrarán los precedentes del problema de investigación, demostrando la necesidad de implementar sistemas *SLAM* para realizar robots autónomos capaces de desempeñarse de forma correcta en entornos desconocidos, se definirá el problema, se justificará el mismo y de igual manera se plantearán una serie de preguntas junto a los objetivos que ayudará a establecer los límites de la investigación.

2.1 PRECEDENTES DEL PROBLEMA

La necesidad de robots autónomos se debe a la creciente demanda que se ha generado en la industria en la últimas dos décadas debido a los nuevos modelos de negocios y a los altos consumos de productos variados como ser: tecnológicos, médicos, alimenticios, los cuales para su producción requieren de este tipo de dispositivos especialmente porque éstos ayudan a reducir el esfuerzo humano, ahorran mucho tiempo y se pueden utilizar para tareas peligrosas evitando así, exponer a los humanos a situaciones riesgosas las cuales pueden llegar a ser fatales.

Ilustración 1 Línea del tiempo de *SLAM*

Fuente: propia con datos recuperados de (Cadena, C., Carlone, L., Carrillo, H., Latif, Y., Scaramuzza, D., Neira, J., … Leonard, J. J., 2016).

Como se puede observar en la ilustración 1. el estudio de *SLAM* surgió en 1986 con el primer problema sugerido por Durrant-Whyte y Bailey, con ello introdujeron las fórmulas principales para *SLAM* incluyendo las aproximaciones realizadas al filtro de Kalman. En este período se basaron más en entender el modelo matemático que posteriormente llevaría a la implementación de este tipo de tecnología; fue hasta 2006, que hubo un acercamiento de probabilidades importante y surgieron los datos asociados a *SLAM.* En 2008, se logró conocer más de los filtros requeridos para el uso de esta tecnología. Para el 2011 se estaba llegando a la mitad del período llamado "análisis de algoritmos" conociéndose más del *back-end,* y con ello lográndose una odometría visual y fue en el año 2016 cuando finalizaba el período de "análisis de algoritmos" con los robots múltiples que se desarrollaron con esta tecnología y más aspectos teóricos que sentaron las bases para lo que hoy conocemos como *SLAM*. (Cadena, C., Carlone, L., Carrillo, H., Latif, Y., Scaramuzza, D., Neira, J., … Leonard, J. J., 2016).

El acceso a los parámetros y características claves de un entorno se vuelve crítico e importante en situaciones en las que intentamos explorar entornos en los que no es posible la intervención humana. (Ashish, R., Sujay, B. M., Caushik, R., Jayashree, S., & Nischal, H. P., 2018)

2.2 DEFINICIÓN DEL PROBLEMA

Un robot sin el sistema de visión y sin un sistema de localización no es capaz de moverse de forma autónoma y mucho menos de operar de forma fluida en un ambiente desconocido. Esto genera una dependencia total en este tipo de dispositivos, como por ejemplo los vehículos de carga, estos para que puedan cumplir con su función de transportar materia prima deben ser manipulados y movilizados en su totalidad por un operario, esta dependencia hace que los procesos sean menos precisos y muchas veces que la materia prima transportada resulte dañada en el camino. La demanda de este tipo de robots aumenta debido a la necesidad de optimizar procesos y especialmente, evitar poner en riesgo la vida humana. La cantidad de estudios que se realizan en zonas de difícil acceso o contaminadas aumenta debido a la necesidad de obtener nuevos conocimientos y generar avances importantes para la ciencia.

Es por ello que en la presente investigación se pretende desarrollar un robot capaz de conocer el entorno en el que opera apoyándose de un sensor *LIDAR* 2D y de codificadores incrementales para posteriormente hacer uso de la inteligencia artificial que le permitirá

desempeñarse en ambientes desconocidos y al mismo tiempo generar un modelo gráfico muy acertado a la realidad utilizando *ROS*.

2.3 JUSTIFICACIÓN

La vida humana es importante, debido a ello los avances tecnológicos van orientados a minimizar el esfuerzo humano y así mismo, el riesgo a los cuales puede enfrentarse una persona en situaciones críticas como en tareas de rescate o en exploraciones en lugares limitados o de difícil acceso. Los robots autónomos capaces de mapear entornos desconocidos históricamente han sido difíciles de desarrollar y costosos, es por ello que se pretende desarrollar un robot autónomo con un sistema *SLAM*.

Con la implementación de este tipo de tecnología se puede realizar un robot autónomo y tele operado al mismo tiempo. Dicho robot puede ser de mucha utilidad en diversos campos debido a la cantidad de información que éste nos puede proveer. Un sistema *SLAM* se puede implementar de diferentes maneras ya sea, utilizando visual *SLAM* que consiste en utilizar una cámara entrenada con inteligencia artificial para detección de objetos o utilizando sensores de medición. Para el diseño de un robot *SLAM* se debe tomar en cuenta cuáles son sus puntos débiles y según mencionan (Ashish, R., Sujay, B. M., Caushik, R., Jayashree, S., & Nischal, H. P., 2018):

"El mayor obstáculo que se encuentra un sistema SLAM es el ruido".

Conociendo dicha limitación se ha decidido desarrollar un robot autónomo con sistema *SLAM* haciendo uso de un sensor *LIDAR 2*D, este sensor utiliza el principio de triangulación laser para su funcionamiento y el *software* con el que éste trabaja que utiliza los filtros de Kalman para minimizar la cantidad de ruido y con el cual al emplearlo se genera una señal pura y precisa.

SLAM basado en el sensor *LIDAR* es muy popular, debido a que el sensor LIDAR tiene las ventajas de alta precisión, posee fuerte resistencia a las interferencias y es robusto a la variación de iluminación. El sensor *LIDAR 2D* disfruta de las ventajas de la medición de rango de alta precisión y de bajo costo en comparación con los costosos escáneres *LIDAR 3D*, lo que lo convierte en una opción más adecuada para escenarios de aplicación ((Ren R., Fu H., & Wu M.,, 2020).

Como se puede observar en la ilustración 2 es de suma importancia que un robot posea un sistema de visión ya que se le permite identificar obstáculos y de esta menara poder evitarlos, generando así una movilidad fluida.

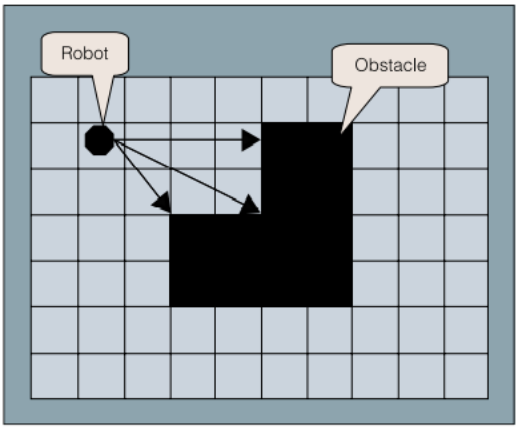

Ilustración 2 Ejemplo de un robot identificando obstáculos.

Fuente: (Manikas, T., Ashenayi, K., & Wainwright, R., 2007)

2.4 PREGUNTAS DE INVESTIGACIÓN

1. ¿Qué limitaciones tendrá el sensor *LIDAR* 2D aplicándolo en un sistema *SLAM*?
2. ¿Cuáles son las características que debe poseer el robot para considerarse autónomo?
3. ¿De qué forma obtendremos la información de la localización del robot en tiempo real?
4. ¿Cómo se integrarán los sensores y actuadores del prototipo para lograr la comunicación y retroalimentación de estos?

2.5 OBJETIVOS

A continuación, los objetivos del proyecto que ayudarán a medir los límites de la investigación.

2.5.1 OBJETIVO GENERAL

Desarrollar un robot autónomo y tele operado capaz de localizar y mapear de forma simultánea un entorno desconocido a través del uso de un sensor *LIDAR 2D*.

2.5.2 OBJETIVOS ESPECÍFICOS

- Identificar las limitaciones del sensor *LIDAR 2D* al aplicarse a un sistema *SLAM*
- Enumerar las características necesarias para desarrollar un robot autónomo.

- Seleccionar el dispositivo necesario para obtener la localización del robot en tiempo real.
- Implementar un *framework* que permita comunicar y obtener los datos de manera remota del prototipo.

III. MARCO TEÓRICO

Durante la ejecución de este capítulo se pretende abordar tanto conceptos básicos como avanzados los cuales ayudarán a desarrollar de una manera correcta la investigación. Con la información obtenida se podrá clasificar diferentes tipos de robots y de esta forma poder elegir el que mejor se adecue a las necesidades propuestas.

Se entiende que estos sistemas están basados en tecnologías inteligentes las cuales se apoyan de una u otra manera de diferentes lenguajes de programación, estos lenguajes son los encargados de darle vida al robot de manera inteligente; pero no solamente se debe basar en la inteligencia del robot, para su funcionamiento éste necesita sensores, actuadores, placas electrónicas etc. Todo este conjunto de elementos crea lo que comúnmente se denomina sistema o máquina autónomos.

Se pueden detallar parámetros y métodos para poder mejorar el rendimiento del robot autónomo a crear, de igual manera se pretende apoyarse de inteligencia artificial la cual se encargará de traducir lo que por consiguiente los sensores captan. De esta forma será posible conocer las características físicas y virtuales de las herramientas a utilizar para la correcta realización del proyecto.

3.1 ANÁLISIS DE LA SITUACIÓN ACTUAL

En las últimas décadas se ha incrementado el uso de robots tanto en fábricas como en los hogares de las personas. La proliferación de la robótica cada vez se va acelerando gracias a los avances de la electrónica, mecánica y otras disciplinas similares. Se están construyendo diferentes aparatos capaces de resolver todo tipo de tareas desde la automatización de cadenas de producción hasta robots capaces de poder ayudar en labores médicas.

La industria manufacturera según los datos obtenido en 2019 había alcanzado un récord mundial que era 113 unidades por cada 10,000 empleados. El país con mayor densidad es Singapur con un total de alrededor de 918 robots, seguidos por Corea del sur con un total de 868 unidades.

A continuación, se muestra los datos que representan la densidad de robots entre el año 2016 al 2019.

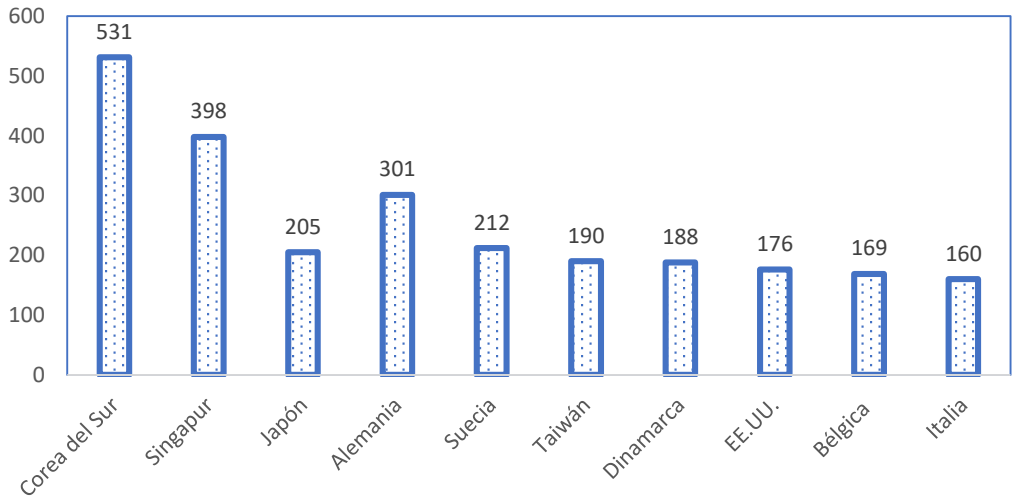

Ilustración 3 Densidad de robots por cada 10,000 habitantes en 2016

Fuente: (IFR (International Federation of Robotics), 2021)

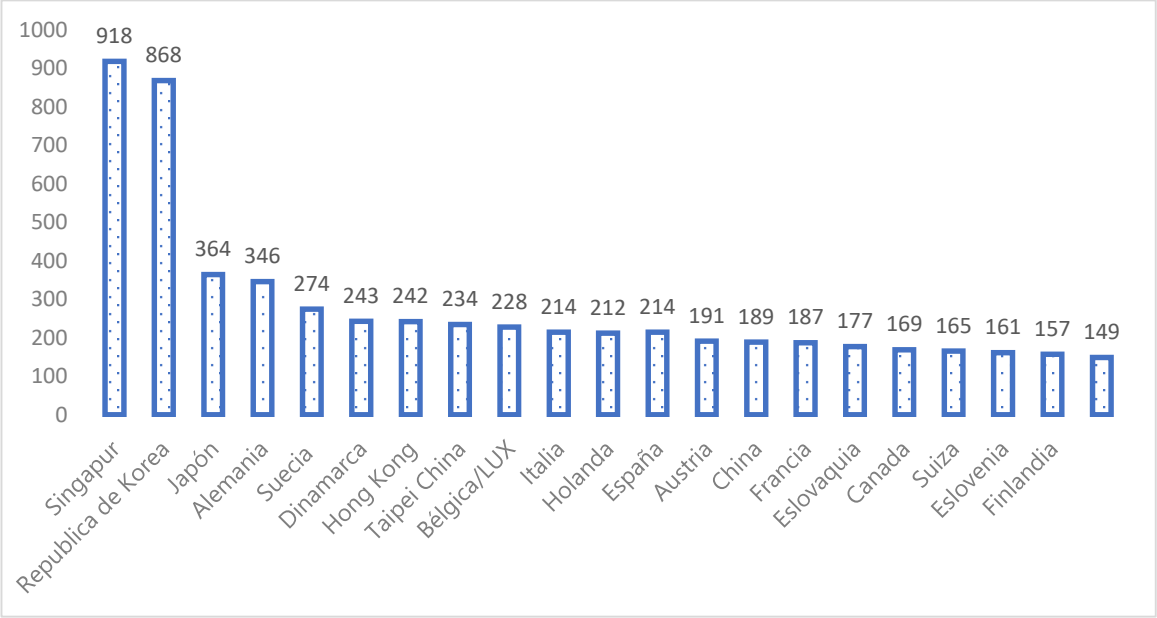

Ilustración 4 Densidad de robots por cada 10,000 habitantes en 2019

Fuente: (IFR (International Federation of Robotics), 2021)

Millton Guerry presidente de la Federación Internacional de Robótica menciona:

> La densidad de robots es el número de robots industriales operativos en relación con el número de trabajadores. Esta medición de nivel permite comparaciones de países

con diferentes tamaños económicos en la carrera de automatización dinámica a lo largo del tiempo. (IFR (International Federation of Robotics), 2021).

Es importante reconocer la tendencia de los robots en este momento histórico. En la actualidad la gama de posibilidades para construir robots es muy amplia. Los componentes y dispositivos necesarios para la construcción de estos cada vez se vuelve más accesible. Las inversiones en tecnología robótica moderna también estarán impulsadas por el requisito de una menor huella de carbono. Los robots modernos son energéticamente eficientes, reduciendo así directamente el consumo de energía de la producción. A través de una mayor precisión, también producen menos rechazos y productos de calidad inferior, lo que tiene un impacto positivo en la relación entre la entrada de recursos y la producción. Además, los robots ayudan en la producción rentable de equipos de energía renovable, como la fotovoltaica o las pilas de combustible de hidrógeno. (IFR (International Federation of Robotics), 2021).

La utilización de nuevas tecnologías ha llegado a diferentes límites. Hoy en día se utilizan diferentes tecnologías para poder reducir el impacto del cambio climático, incluyendo la robótica. Diferentes robots se están creando para esta tarea, desde robots capaces de limpiar ciudades de la basura hasta robots que puedan apoyar en hogares humanos para reducir el consumo de energía.

3.2 TEORÍAS DE SUSTENTO

3.2.1 Robótica

Se conoce a la robótica como una disciplina en auge, la cual se deriva de las ingenierías mecánica, electrónica, eléctrica, biomédica y de ciencias de la computación. Podremos decir que la robótica como tal es la "formación del profesional de la ingeniería, en ramas de automatización, mecánicas e informáticas" (Barrientos, 2012).

3.2.1.1 Robot

(Saha, 2011) asevera:

> El robot se define, de manera formal en la Organización Internacional para la Estandarización (ISO), como un manipulador multifuncional reprogramable, capaz de mover materiales, piezas, herramientas o dispositivos especiales, a

través de movimientos variables programados, para el desempeño de tareas diversas.

Existen muchas más definiciones dadas por diferentes asociaciones del mundo por dar un ejemplo las asociaciones como *Japan Industrial Robot Association* (JIRA) u otras como *Bristish Robot Asocciation* (BRA) que aseguran que un robot es un sistema reprogramable y la multifuncionalidad que estos son capaces de ofrecer a las diferentes áreas alas que pertenecen.

Los robots son máquinas en las que se integran componentes mecánicos, eléctricos, electrónicos y de comunicaciones, y dotadas de un sistema informático para su control en tiempo real, percepción del entorno y programación. La robótica industrial nace en los cincuenta y sólo en los setenta comienzan a impartirse cursos de robótica en un gran número de universidades En la robótica industrial se trata fundamentalmente de dotar de flexibilidad a los procesos productivos manteniendo al mismo tiempo la productividad que se consigue con una máquina automática especializada. En los años ochenta y noventa se han diseñado un gran número de máquinas cuyo objetivo no es sustituir la actividad directa de un trabajador en una cadena de producción. Se trata de realizar tareas en lugares difícilmente accesibles, con riesgo de accidentes, en condiciones peligrosas para la salud, o trabajos que resultan difíciles por el tamaño de los objetos que es necesario manipular. Así, se han abordado aplicaciones en la exploración espacial, actividades subacuáticas, manipulación y transporte de materiales peligrosos, minería, agricultura, construcción, cirugía, etc. La mayoría de estas máquinas son fundamentalmente tele operadas, pero se trata de dotarlas de una mayor autonomía llegando a construir robots. El número de estos robots aumentará de forma importante en los próximos años. Otro sector de importancia creciente en las aplicaciones de la robótica es el de los robots de servicios, entre los cuales se incluyen los robots domésticos, robots de ayuda a los discapacitados, y robots asistentes en general. Tampoco hay que olvidar que los humanos siempre han sentido una gran atracción por las máquinas que imitan los gestos más visibles de los seres vivos en general y de las personas en particular. Por tanto, tampoco hay que extrañarse de la importancia creciente de las aplicaciones recreativas de la robótica.

(Baturone, 2001).

Se pueden clasificar los robots en 3 grandes grupos como ser "*industriales, no industriales o para usos especiales"* (Saha, 2011). Algo que se debe tener en claro siempre es el hecho de que en muchas ocasiones llegamos a confundir diversos términos utilizados en la industria como ser máquinas-herramienta con control numérico computarizado común mente llamadas *CNC*, este tipo de maquina siempre se llega a confundir con el termino robot debido a que esta posee características como la multi-utilidad, si bien esto es cierto no se puede concluir que una *CNC* pertenece a la una de las muchas familias de los robot lo que se debe dejar en claro es que esta no es igual a un robot porque carece de funciones como la reprogramación y multifuncionalidad.

3.2.1.2 Robots Autónomos Y Tele-robótica

De acuerdo con su grado de autonomía, los robots pueden clasificarse en tele operados de funcionamiento repetitivo y autónomos o inteligentes, En los robots tele operados las tareas de percepción del entorno, planificación y manipulación compleja son realizadas por humanos. Es decir, el operador actúa en tiempo real cerrando un bucle de control de alto nivel. Los sistemas evolucionados suministran al operador realimentación sensorial del entorno (imágenes, fuerzas, distancias). En manipulación se emplean brazos y manos antropomórficos con controladores.

Robots autónomos y Tele-robótica automáticos que reproducen los movimientos del operador. Alternativamente, el operador mueve una réplica a escala del manipulador, reproduciéndose los movimientos en éste. Estos robots son interesantes para trabajos en una localización remota (acceso difícil medios contaminados o peligrosos), en tareas difíciles de automatizar y en entornos no estructurados, tales como las que se realizan en la construcción o en el mantenimiento de líneas eléctricas. Las mayores dificultades radican en las limitaciones del hombre en la capacidad de procesamiento numérico y precisión y. sobre todo, en el acoplamiento y coordinación entre el hombre y robot. En algunas aplicaciones, el retraso de transmisión de información juega también un papel importante y su consideración resulta fundamental en el diseño del sistema de control. El diseño de la interfaz persona-máquina suele ser crítico. No obstante, existen limitaciones por el ancho de banda de la transmisión y, eventualmente, por la complejidad de la tarea del operador. Los robots de funcionamiento repetitivo son la mayor parte de los que se emplean en cadenas de producción industrial. Trabajan normalmente en tareas predecibles e invariantes, con una limitada percepción del

entorno. Son precisos, de alta repetitividad y relativamente rápidos; incrementan la productividad ahorrando al hombre trabajos repetitivos y, eventualmente, muy penosos o incluso peligrosos Los robots autónomos o inteligentes son los más evolucionados desde el punto de vista del procesamiento de información. Son máquinas capaces de percibir, modelar el entorno, planificar y actuar para alcanzar objetivos sin la intervención, o con una intervención muy pequeña, de supervisores humanos.

3.2.2 SLAM

Para poder comprender el funcionamiento del proyecto se debe conocer y entender de qué trata este tipo de tecnología llamada *SLAM (Simultaneous localization and mapping)*, derivando de sus siglas en inglés, sabemos que es localización y mapeo simultáneo. Este tipo de tecnología está siendo utilizada para la fabricación de diversos robots autónomos.

La estimación simultánea del estado de un robot equipado con sensores a bordo y la construcción de un modelo (el mapa) del entorno que los sensores están percibiendo. En casos simples, se describe el estado del robot por su pose (posición y orientación), aunque otras cantidades pueden incluirse en el estado, como la velocidad del robot, los sesgos del sensor y los parámetros de calibración. El mapa, por otro lado, es una representación de aspectos de interés (por ejemplo, posición de puntos de referencia, obstáculos) que describen el entorno en el que el robot opera (Cadena, C., Carlone, L., Carrillo, H., Latif, Y., Scaramuzza, D., Neira, J., … Leonard, J. J., 2016).

La implementación de un sistema *SLAM* en los robots autónomos de interiores cobra mayor importancia puesto que de las tecnologías de las cuales normalmente se dispone a la hora de realizar dispositivos autónomos de exteriores son imprecisas y no proporcionan información correcta de lo que pueden llegar a enfrentarse los robots autónomos al ser utilizados en interiores, menciónese el *GPS*. Debido a lo limitado que puede llegar a ser un robot autónomo en interiores haciendo uso del *GPS* aumentó la demanda de sistemas *SLAM* en los robots autónomos, especialmente en los que operan en interiores y es en los cuales se estará enfocado en esta investigación.

La popularidad del problema *SLAM* está relacionada con la aparición de aplicaciones interiores de robótica móvil. Para las operaciones en interiores se descarta el uso de GPS para delimitar la localización error; Además, *SLAM* ofrece una atractiva alternativa a mapas

construidos por el usuario, que muestran que la operación del robot es posible en la ausencia de una infraestructura de localización ad hoc (Cadena, C., Carlone, L., Carrillo, H., Latif, Y., Scaramuzza, D., Neira, J., … Leonard, J. J., 2016).

Para la creación de un sistema *SLAM* se puede hacer uso ya sea de sensores o cámaras con inteligencia artificial para lograr la detección de los obstáculos, al utilizar las cámaras se deben tomar en cuenta muchas variables para que nuestro robot pueda entender qué es un obstáculo y qué no lo es, por ejemplo, si no se llegan a considerar todas las variables y se forma una nube de polvo o algo por estilo, la cámara al detectar las partículas podría interpretar como obstáculo, cuando en realidad es algo que no afectaría en lo absoluto la navegación del robot en interiores. Otra desventaja del uso de cámaras para los sistemas *SLAM*, es que la cámara es más propensa a la interferencia, de igual manera se ven directamente afectadas por la cantidad de luz que se encuentra en el ambiente, sin contar también que el lente puede llegar a detectar ruido en el ambiente por la contaminación ambiental que normalmente hay dentro de la industria. Es por ello que los sensores *LIDAR* se han vuelto muy populares. El sensor *LIDAR 2D* disfruta de las ventajas de la medición de rango de alta precisión y de bajo costo en comparación con los costosos escáneres *LIDAR 3D*, lo que lo convierte en una opción más adecuada para escenarios de aplicación.

Normalmente, un sistema *LIDAR SLAM* consta de un *front-end* de odometría *LIDAR* y un *back-end* que realiza la optimización del gráfico de pose para obtener un mapa optimizado globalmente y la trayectoria completa del robot. Además, la detección de cierre de bucle juega un papel importante que proporciona restricciones entre poses que fueron capturados en el mismo lugar, pero en diferentes momentos Ren, R., Fu, H., & Wu, M. (2019).

Los "cierres de bucles" son muy importantes de comprender puesto que juegan un rol importante para que el *SLAM* funcione de manera correcta, para ello se debe entender qué son los cierres de bucles, es un procedimiento necesario para que funcione correctamente el *SLAM*.

Ayuda a eliminar los errores de odometría que se acumulan en el *front-end* para operaciones a largo plazo. En segundo lugar, la localización en un mapa anterior se vuelve posible con cierre de lazo. Una extensión directa de tal habilidad es para permitir la recuperación del problema del robot secuestrado lo que mejora significativamente la robustez

del sistema. En tercer lugar, el mapeo cooperativo con varios robots requiere algoritmos de cierre de bucle, de modo que los sub mapas de varios los robots se pueden fusionar en un mapa coherente global Li, J., Zhan, H., Chen, B. M., Reid, I., & Lee, G. H. (2017).

Existen algoritmos de optimización que han ayudado a las gráficas realizadas por el *front-end* en estos tipos de sensores como, por ejemplo: RRR (realizing, reversing, recovering).

Está basado en la observación de que el cierre correcto del lazo en conjunto con odometría puede ayudar en la detección de bucle incorrecto cierres. Nuestro método sigue la línea de trabajo en la que el propio proceso de estimación se utiliza para hacer la distinción

entre cierres de bucle correctos y falsos.

Ha sido incluido aquí en aras de la integridad. La formulación basada en gráficos para *SLAM*, el llamado modelos "*graph-SLAM*" que el robot presenta como nodos en un gráfico donde las transformaciones relativas de la odometría o los cierres de bucle forman bordes o "restricciones". (Latif, Y., Cadena, C., & Neira, J., 2012)

La localización por coincidencia de escaneo generalmente implica el punto más cercano iterativo (algoritmo *ICP*). *ICP* es un método utilizado para minimizar la diferencia entre las dos nubes de puntos. El método se basa segmentación de la nube de puntos utilizada para reducir el tiempo de procesamiento seguido de la agrupación. Este algoritmo se utiliza a menudo para reconstruir la superficie de los resultados del escaneo para fines de navegación del robot (Muhammad Rivai, Dony Hutabarat, Zishwa Muhammad Jauhar Nafis, 2020).

3.2.2.1 Principio De Triangulación

El principio de triangulación históricamente es uno de los principios más utilizados por sensores que pretenden medir ambientes tridimensionales, es de los más utilizados debido a que es simple y muy robusto. Dicho principio se centra en medir ambientes los cuales no tengan tantos defectos como pueden ser grietas o perforaciones, es más censar superficies más sólidas. (Dorsch, R. G., Häusler, G., & Herrmann, J. M., 1994).

Los sensores *LIDAR* utilizan este principio, cuentan con un emisor de láser el cual envía un pulso de luz hacia enfrente y cuando el emisor capta dicha luz, el microprocesador que poseen se encarga de realizar los cálculos necesarios y cuantificar las señales, normalmente se basan en el tiempo que tardó el láser en ser reflejado para calcular la distancia a la que se

encuentra el objeto. En el caso del sensor LIDAR 360° tiene una distancia mínima de medición debido a que este sensor se encuentra girando casi de forma permanente permitiendo adquirir datos hasta 8000 veces por segundo y debido a ello podría encontrarse muchos errores de medición si los objetos se encuentran a menos de 10 cm de distancia.

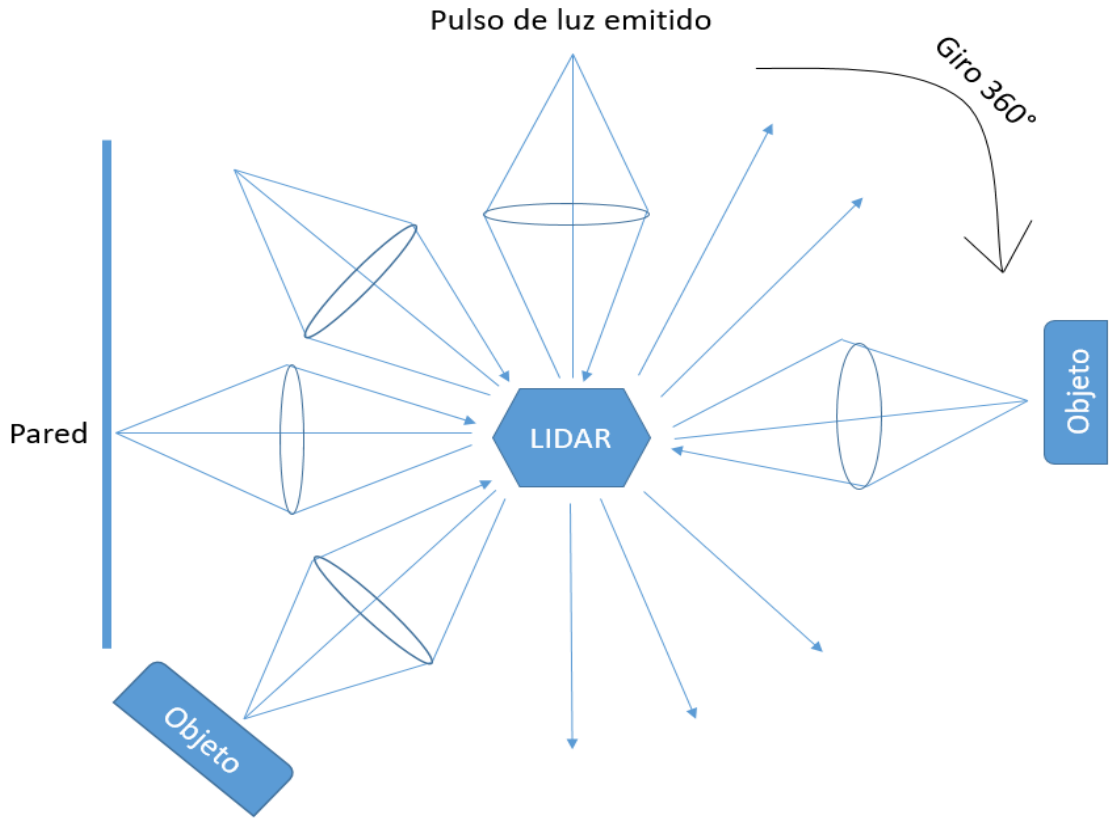

Ilustración 5 Modelo de triangulación

Fuente: propia

Como se puede observar en la ilustración 5 el sensor LIDAR envía pulsos de luz a 360° y cada que el láser retorna, el receptor se encarga de retroalimentar al microprocesador incorporado para que logre cuantificar las señales recibidas y convertirlas en algo gráfico.

3.2.2.2 Filtros De Kalman

Anteriormente se ha mencionado acerca del ruido que puede llegar a afectar la medición cuando están funcionando las cámaras y sensores en un robot autónomo, para minimizar estos ruidos se utilizan los filtros de Kalman.

(Ortiz Bravo, V., Nieto Arias, M., & Castañeda Cardenas, J., 2013)aseveran:

El filtro de Kalman es un algoritmo que se basa en el modelo de espacio de estados de un sistema para estimar el estado y la salida futuros realizando un filtrado optimo a la señal de salida, y dependiendo del retraso de las muestras que se le ingresan puede cumplir la función de estimador de parámetros o únicamente de filtro. Pero en ambos casos elimina ruido, estas ecuaciones son ampliamente utilizadas ya que incluyen probabilidades estadísticas puesto que toma en cuenta la aleatoriedad tanto de la señal como del ruido. El filtro de Kalman es esencialmente una serie de ecuaciones matemáticas que implementan un estimador tipo predictor – corrector que es óptimo en el sentido que minimiza el error estimado de la covarianza, cuando algunas condiciones son dadas.

3.2.2.2.1 Modelos de Espacio de Estado

Los modelos de espacio de estados son esencialmente una noción conveniente para estimulación y control, desarrollado para hacer trazable lo que sería de otra manera un análisis intrazable. Considere un proceso dinámico descrito por una ecuación de forma

(Ortiz Bravo, V., Nieto Arias, M., & Castañeda Cardenas, J., 2013) nos demuestra lo siguiente:

$$y_{i*1} = a_{0,\acute{i}} y_i + \cdots + a_{n-1,\acute{i}}\, y_{i-n+1} + \mu_{i,} \quad i \geq 0$$

Ecuación 1 Modelo de espacio de estados

Donde $\mu_{i,}$ es un ruido blanco (espectralmente) aleatorio de media cero (estadísticamente) correlación:

$$E(\mu_i\, , \omega_j) = R_\mu = Q_i \delta_{ij}$$

Ecuación 2 Correlación del ruido

Con valores iniciales y_0, y_{-1}..., y_{-n+1} son variables aleatorias con media cero con una matriz de covarianza conocida de dimensiones *n*n*.

$$P_0 = E(y_{-j}, y_{-k}), \qquad j, k \in 0, n-1$$

Ecuación 3 Variables aleatorias

También se puede asumir:

$$E(\mu_i\, , y_j) = 0\ para - n + 1 \leq j \leq 0\ y\ i \geq 0$$

Ecuación 4 Correlación de ruido para variables aleatorias

Lo que asegura que:

$$E(\mu_i, y_j) = 0 \; i \geq j \geq 0$$

Ecuación 5 Ruido estimado

En otras palabras, el ruido es estadísticamente independiente del proceso que se va a estimar.

Bajo otras condiciones básicas esta ecuación diferencial puede ser escrita como:

$$\hat{x}_{i+1} = \begin{bmatrix} y_{i+1} \\ y_i \\ y_{i-1} \\ \vdots \\ y_{i-n+2} \end{bmatrix} = \begin{bmatrix} a_0 & a_1 & \cdots & a_{n-2} & a_{n-1} \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \ldots & 1 & 0 \end{bmatrix} \begin{bmatrix} y_i \\ y_{i-1} \\ y_{1-2} \\ \vdots \\ y_{i-n+1} \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \mu_i$$

Ecuación 6 Representación de Ecuación Diferencial en forma de matriz

Que conlleva al modelo de espacio de estados:

$$\hat{x}_{i+1} = A\hat{x}_i + G_{\mu i}$$

$$\hat{y}_i = [1 \quad 0 \quad \ldots \quad 0]\hat{x}_i$$

Ecuación 7 Modelo de espacio de estados

O de forma más general:

$$\hat{x}_{i+1} = A\hat{x}_i + G\mu_i$$

$$\hat{y}_i = H\hat{x}_i$$

Ecuación 8 Modelo de espacio de estado de manera general

3.2.2.2.2 Problema De Diseño Del Observador

El problema básico es determinar los estados internos de un sistema lineal, teniendo acceso a solo las salidas del sistema, éste proceso es descrito por la siguiente ecuación:

$$X_k = Ax_{k-1} + B\mu_k + W_{k-1}$$

Ecuación 9 Estados internos de un sistema lineal

El modelo de medición que describe la relación entre el proceso de estados y las medidas se puede representar de la siguiente manera:

$$Z_k = Hx_k + V_k$$

Ecuación 10 Modelo de medición procesos de estados

Las variables aleatorias W_k y V_k representan respectivamente el ruido del proceso y de la medición.

Las ecuaciones específicas para actualización de tiempo y medida son presentadas de la siguiente manera

$$\hat{x}_k^- = A\hat{x}_{k-1} + B\mu_k$$

$$P_k^- = AP_{k-1}A^T + Q$$

Ecuación 11 Actualización de tiempo

Ecuaciones de actualización de mediciones de filtro de Kalman discreto

$$K_k = P_k^- H^T (HP_k^- H^T + R)^{-1}$$

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-)$$

$$P_k = (I - K_k H)P_k^-$$

Ecuación 12 Filtro de Kalman discreto

Planteado el filtro de Kalman discreto se reasignan las variables utilizando la siguiente fórmula:

$$\hat{x}_k^- = \hat{x}_k$$

$$P_k^- = P_k$$

Ecuación 13 Reasignación de variable

3.2.3 ROS

Robot Operating System (*ROS*) es un sistema flexible que proporciona varias herramientas y bibliotecas para escribir *software* robótico. Ofrece varias funciones potentes para ayudar a los desarrolladores en tareas como el paso de mensajes, la distribución de códigos informáticos, la reutilización y la implementación de algoritmos de última generación para aplicaciones robóticas. El proyecto *ROS* se inició en 2007, con el nombre *Switchyard*, por Morgan Quigley, como parte del proyecto del robot *Stanford STAIR*. L. (Joseph, L., & Cacace, J., 2018)

Estas son algunas de las razones por las cuales *ROS* es una excelente opción para robots autónomos:

- Capacidades de gama alta: *ROS* viene con capacidades listas para usar. Por ejemplo, los paquetes de localización y mapeo simultáneo (*SLAM*) y de localización adaptativa de Monte Ca (*AMC*) en *ROS* se pueden utilizar para realizar navegación autónoma en robots móviles, y el paquete "*move it*" se puede utilizar para la planificación del movimiento de manipuladores de robots.
- Decenas de herramientas: *ROS* es una tecnología con toneladas de herramientas entre ellas depuración, visualización y realización de una simulación. Las herramientas, tales como rqt_gui, *RViz* y *Gazebo* son algunas de las herramientas sólidas de código abierto para depurar la visualización y la simulación. Un *software* de trabajo que tiene tantas herramientas es muy bueno.
- Soporte para sensores y actuadores de gama alta: *ROS* está repleto de controladores de dispositivos y paquetes de interfaz de varios y actuadores en

robótica. Los sensores de gama alta incluyen *Velodyne-LIDAR*, escáner láser, *Kinect*, etc. Y actuadores como servos DYNAMIXEL.

- Interoperabilidad entre plataformas: el *middleware* de paso de mensajes *ROS* permite la comunicación entre diferentes nodos. Estos nodos se pueden programar en cualquier lenguaje que tiene bibliotecas de cliente *ROS*. Podemos escribir nodos de alto rendimiento en C ++ o C y otros nodos en *Python* o *Java*.
- Modularidad: Uno de los problemas que pueden ocurrir en la mayoría de las aplicaciones robóticas independientes es que si alguno de los hilos del código principal falla, toda la aplicación del robot puede detenerse. En *ROS*, la situación es diferente, estamos escribiendo diferentes nodos para cada proceso, y si un nodo falla, el sistema aún puede funcionar. Además, *ROS* proporciona métodos robustos para reanudar las operaciones incluso si algún sensor o motor está fallando.
- Manejo concurrente de recursos: manejar un recurso de hardware a través de más de dos procesos es siempre un dolor de cabeza. Imagine que queremos procesar una imagen de una cámara para detección de rostros y detección de movimiento, podemos escribir el código como una sola entidad que puede hacer ambas cosas o podemos escribir un código de un solo subproceso para la concurrencia, si queremos agregar más de dos características en subprocesos, el comportamiento de la aplicación se volverá complejo y será difícil de depurar. Pero en *ROS*, podemos acceder a los dispositivos usando temas *ROS* desde los controladores *ROS*. Cualquier número de nodos *ROS* puede suscribirse al mensaje de imagen del controlador de la cámara *ROS*, y cada nodo puede realizar diferentes funcionalidades. Puede reducir la complejidad en computación y también aumentar la capacidad de depuración de todo el sistema.
- Comunidad activa: Cuando elegimos una biblioteca o un marco de software, especialmente de una comunidad de código abierto, uno de los principales factores que debe verificarse antes de usarlo es el soporte de *software* y la comunidad de desarrolladores. No hay garantía de soporte de una herramienta de código abierto. Algunas herramientas brindan un buen soporte y algunas herramientas no lo hacen en *ROS*, la comunidad de apoyo está activa (Joseph, L., & Cacace, J., 2018).

Conociendo todas las ventajas que nos ofrece *ROS*, se puede asegurar que la implementación de este tipo de sistema a nuestros robots autónomos es una decisión muy buena, la versatilidad que ofrece este sistema es algo que en otros sistemas no logramos, además agregar lo robusto que puede llegar a ser este sistema, en este tipo de dispositivos muchas veces estamos limitados por el hecho que los sensores un poco más industriales o de usos más avanzados como por ejemplo el sensor *LIDAR*, requieren de un sistema más avanzado y potente para poder generar una retroalimentación correcta, éste tipo de sensores en un microcontrolador común como en un arduino o una pic es complicado que llegue a generar un mapa de forma remota y en tiempo real puesto a las limitantes que tienen éstos microcontroladores.

3.2.3.1 MICROCONTROLADOR

(Dogan, 2008) nos dice:

"*Un microcontrolador es un ordenador de un único chip. La palabra micro indica que el dispositivo es pequeño, y controlador indica que el dispositivo se puede usar en aplicaciones de control*".

Los microcontroladores se programan común mente en lenguajes especiales de programación llamados ensamblador, este tipo de lenguajes se catalogan como difíciles por su alto de compuesto que lo conforman y es más complejo de aprender. Pero no solamente existe esta manera de programar microcontroladores también pueden ser programados en los denominados lenguaje de alto nivel en los cuales se destacan *BASIC*, *PASCAL* y *C*. Estos lenguajes de programación a diferencia del lenguaje ensamblador son más fáciles de aprender y pueden desarrollar proyectos más complejos.

3.2.3.2 I2C

I2C es un protocolo de comunicación en serie que está basado en 8 bits. Este protocolo realiza la transmisión de datos mediante dos líneas de señal. Las líneas de señal son *SCL Y SDA*. *SCL* es la línea del reloj serial, es una línea en serie y *SDA* es en línea bidireccional, ambas líneas permiten transmitir datos entre maestro y esclavos. Después de cada transferencia de datos hay un bit de reconocimiento que es enviado del esclavo al maestro o viceversa para poder comprobar que se han enviado los datos con éxito. Cuando el bus está inactivo las dos líneas

envían un 1 en binario, cuando el maestro envía instrucciones la línea *SDA* pasa de un 1 binario a un 0, mientras que el *SCL* se mantiene en un 1 binario (Kumari, 2017).

3.2.3.3 Python

Como tal *Python* es un lenguaje de programación el cual "fue desarrollado por el matemático Guido Van Rossum a finales de los 80 y a inicios de los 90" (Algar Diaz, 2019). Este lenguaje como tal se puede denominar un lenguaje muy sencillo y versátil de aprender dado que podemos implementar código de una manera más natural dejando a un lado reglas sintácticas de otros lenguajes.

(Algar Diaz, 2019) menciona:

> Python es un lenguaje interpretado. Las instrucciones o código fuente escrito por el programador en este lenguaje de alto nivel son traducidas por el intérprete a un lenguaje entendible por la máquina (lenguaje máquina). Este proceso se repite cada vez que se ejecutan las diferentes instrucciones de que consta un programa. *Python* dispone de un intérprete por la línea de comandos en el que se pueden introducir sentencias. Cada sentencia o instrucción se ejecuta y produce un resultado visible, que ayuda a clarificar la comprensión del código escrito y verifica la validez de los resultados de la ejecución de pequeñas porciones de código de forma inminente

Debido a esto este lenguaje de programación soporta el paradigma de programación orientada a objetos ofreciendo en muchas ocasiones una manera más sencilla y cómoda para elaboración de programas reutilizables.

Otros autores nos dejan en claro lo siguiente de *Python* con respecto a otros lenguajes de programación:

- Los programas escritos en *Python* son más compactos que programas escritos en lenguajes como C, *Java,* etc.
- Está considerado como un lenguaje de muy alto nivel.
- Su sintaxis es simple, clara, legible y visual. Todos estos ingredientes, facilitan la lectura y escritura de código.

3.2.3.4 Lenguaje C

El lenguaje de programación C fue desarrollado por Dennis Ritchie en los Laboratorios Bell de la empresa de comunicaciones *AT&T*, en 1972 (Muñoz, 2016). El lenguaje de programación C fue creado con el fin de crear un sistema operativo *UNIX*. "*UNIX* es el núcleo de un sistema operativo de tiempo compartido" (Marquez, 2015).

(Muñoz, 2016) da a conocer algunas de las características más importantes de este lenguaje:

- **Potencia y flexibilidad**. Se ha usado en contextos tan dispares como el desarrollo de sistemas operativos, procesadores de texto, gráficos, bases de datos, compiladores de otros lenguajes, etc.
- **Popularidad.** Existe una gran variedad de compiladores, librerías, herramientas de apoyo a la programación, etc. Es el lenguaje predominante en el entorno *UNIX*.
- **Portabilidad.** El mismo programa escrito en C puede compilarse y ejecutarse sin prácticamente ningún cambio en diferentes ordenadores.
- **Sencillez.** C utiliza pocas palabras clave, por lo que puede aprenderse fácilmente.
- **Estructura y modularidad.** Los programas en C pueden escribirse agrupando el código en funciones que a su vez se agrupan en distintos módulos.

3.2.3.5 Linux

Linux es un sistema operativo basado en código libre, su primera versión se denomina *MINIX* la cual fue creada por el profesor Andrew Tanenbaum de la universidad de Holanda, esta primera aparición del sistema operativo fue creada para el ámbito educativo, así sus alumnos podrían estudiarlos y modificarlo para que ampliaran sus conocimientos.

No fue hasta agosto de 1991, cuando Linus Torvalds decidiera crear un núcleo basado en *UNIX* para la base de un sistema operativo, fue gracias a él y por su iniciativa que hoy por hoy hemos logrado poseer un sistema operativo de código abierto multiplataforma, robusto y con un amplio sector en la industria. Herramientas (Rodriguez, 2015).

3.2.3.5.1 Licencias de Linux

Linux desde su comienzo fue creado para ser un sistema operativo sin costo alguno, gracias a esto posee una gran comunidad de desarrolladores alrededor del mundo tanto profesionales como entusiastas que se dedican al mejoramiento del sistema.

(Rodriguez, 2015) dice en su libro de *Linux* avanzado lo siguiente a tener en consideración con las licencias de *Linux*:

- **Comercial:** Debe ser comprado, no puede ser distribuido, y solamente está disponible como código binario para los usuarios finales. Un ejemplo de este software es *Microsoft Office*.
- ***Software* de evaluación:** Son versiones con características limitadas de software comercial, que pueden ser distribuidas libremente y que intentan ser propaganda para el software comercial.
- **Uso no comercial:** Es software que es se puede usar gratuitamente por individuos e instituciones educativas. Las corporaciones deben comprar una licencia. Ejemplos son *StarOffice* y *Netscape*.
- ***Shareware*:** Son versiones completas y de libre distribución, pero tienen una licencia que obliga a ser pagada para un uso prolongado del *software*. Ejemplos de esto son *WinZip y WinAmp*.
- ***Freeware*:** Consisten en *software* que puede ser libremente usado y distribuido, pero está disponible solamente en forma binaria. Ejemplos de esto son Internet Explorer y *Netmeeting*.
- **Librerías Gratuitas:** Son *software* que puede ser libremente usado y distribuido como código fuente y como binario, pero no puede ser modificado sin violar la licencia. Un ejemplo son las librerías de clases de
- **Software de Fuentes Abiertas, estilo BSD:** Un grupo cerrado de individuos crea el *software* y permite la libre distribución de los binarios y del código fuente. Aunque los usuarios pueden modificar el código, el grupo de desarrollo generalmente no usa las modificaciones de los usuarios.
- **Software de Fuentes Abiertas, estilo Apache:** Es como el *BSD*, pero el grupo de desarrollo puede usar las modificaciones de los usuarios si son útiles.

- **Software de Fuentes Abiertas, estilo GNU GPL:** Además de las características del estilo *Apache*, la licencia *GPL* (*General Public License*) requiere que todos los trabajos derivados del *software* deben estar también bajo esta licencia. Esta característica adicional, ideada por Stallman, es la que protege al software *GNU* de las empresas comerciales.

3.2.3.5.2 Linux en la actualidad

Se puede encontrar hoy en día entre 7 a 8 millones de ordenadores en todo el mundo que corren bajo un sistema operativo *Linux*. Cabe recalcar que la mayoría de estos sistemas operativos corren diferentes versiones de Linux cada una específica en un área en concreto, otros autores nos dicen que Linux es usado en su mayoría en servidores. Un servidor es un conjunto de sistemas el cual se comprende por un sistema operativo, sistemas de comunicación, estructuras de datos (Palomares, 2017).

IV. METODOLOGÍA

En este capítulo se procederá a definir el enfoque que tendrá la metodología de estudio, los procesos necesarios para elaborar la presente investigación, se definirán las variables de investigación, las técnicas e instrumentación a utilizar para el desarrollo del prototipo y se definirán las diferentes etapas presentadas en la metodología seleccionada.

4.1 HIPÓTESIS

En la presente investigación surgen las siguientes hipótesis:

- El sistema *SLAM* le permitirá al robot operar de manera remota y modelará mapas acertados a la realidad.

4.1.1 Hipótesis Nula

- El sistema *SLAM* no le aportó lo necesario al robot para operar de manera remota y tampoco modeló mapas acertados a la realidad.

4.2 ENFOQUE

En la presente investigación se optó por utilizar un enfoque cuantitativo, ya que se realizará el análisis dinámico de movimiento en el cuerpo del robot, análisis estático para corroborar que el prototipo a realizarse soporta todos los componentes implementados, se realizarán diferentes pruebas al sensor *LIDAR* incluyendo la variación en la frecuencia del motor y a los codificadores rotativos para asegurar que generan la retroalimentación correctamente.

4.3 ALCANCE

En la presente investigación se define un alcance descriptivo. Con el cual se pretende describir características de los diferentes elementos que conforman el sistema. De igual manera se busca detallar lo ocurrido en cada uno de los pasos de la metodología. Se pretende desarrollar un prototipo que ayude a responder las preguntas antes planteadas.

4.4 VARIABLES DE INVESTIGACIÓN

Definido el enfoque, se procede a identificar las variables involucradas en la presente investigación. Las variables están relacionadas entre sí, clasificándose en dependiente e independientes. La variable dependiente se define como la razón de la investigación, las variables independientes son las que pueden perjudicar a las dependientes si éstas se ven alteradas.

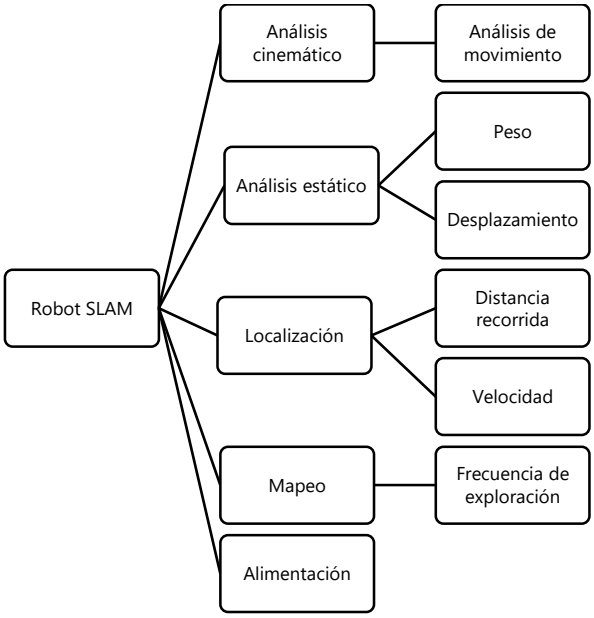

Ilustración 6 Variables dependientes e independientes.

Fuente: propia.

La variable dependiente definida es el robot *SLAM*, su variable independiente análisis cinemático consiste en realizar simulaciones y pruebas para corroborar que el robot se capaz de moverse correctamente con todos los componentes. El análisis estático corroborará que la estructura del robot soporta el peso y no sufre de rupturas de material cuando esté totalmente ensamblado. La localización es la variable que complementa al robot para crear *SLAM*, si el robot se mueve la localización varía, el mapeo dependerá de la frecuencia de exploración del sensor *LIDAR* y la densidad de los puntos, el mapeo tiene que ser claro para lograr una buena navegación autónoma y la variable de alimentación es la que dictará la autonomía y tiempos de carga del robot *SLAM*.

4.5 TÉCNICAS E INSTRUMENTACIÓN

El desarrollo de este proyecto de investigación está sustentado por diversa información adquirida de distintas fuentes como:

- Libros electrónicos
- Artículos publicados en revistas científicas
- Tesis elaboradas en distintas universidades en todo el mundo.

Para el desarrollo del prototipo se hará uso de los siguientes *softwares*:

- *Solidworks*.
- *Fritzing*.
- *Mapper by SLAMTEC*.
- *Hector SLAM* para *Linux*.
- Rviz.
- *Arduino* IDE.
- *Cloud Shell editor by Google*.
- *Matlab*.
- *VNC Viewer*.

El software de modelado 3*D Solidworks* nos permitirá realizar las pruebas dinámicas de movimiento a la estructura del robot, *Fritzing* se utilizará para realizar los diagramas de conexión de los diferentes componentes del prototipo, *Mapper* es el *software* que utiliza el sensor *LIDAR, Hector SLAM* es un *software* de simulaciones para mapeos, Rviz es el *software* de visualización en tiempo real del robot y mapa, *ARDUINO IDE* se utilizará para programar lógica a los *Arduino* nanos y para poder realizar pruebas a los codificadores, *Cloud Shell by Google* es el software utilizado para programar en lenguaje *Python, Matlab* será utilizado para simular los filtros de Kalman y *VNC Viewer* es el programa que permite acceder y manipular remotamente el jetson nano. En cuanto al material de la estructura se realizará un ensamblado con láminas de aleación de aluminio.

4.6 METODOLOGÍA DE ESTUDIO

Para el desarrollo del prototipo se optó por utilizar la metodología en "V" creada por *Vasic & Lazarevic*. La metodología en "V" está enfocada a la producción de dispositivos mecatrónicos los cuales involucran componentes electrónicos, eléctricos, mecánicos y microcontroladores.

La metodología en "V" está compuesta por dos ciclos: ciclo A y ciclo B. En el ciclo A se realiza el diseño y análisis al diseño, una vez fabricado el prototipo, se procede a realizar el ciclo B en el que se realizan las pruebas y correcciones al prototipo con el fin de obtener un prototipo final con la menor cantidad de errores (Vasilije S. Vasić, 2008).

Para la presente investigación en el ciclo A se realizará el diseño para el robot, los componentes a utilizar y se hará un análisis de estos. En el ciclo B se realizarán las correcciones para poder desarrollar el robot autónomo *SLAM* con la menor cantidad de errores.

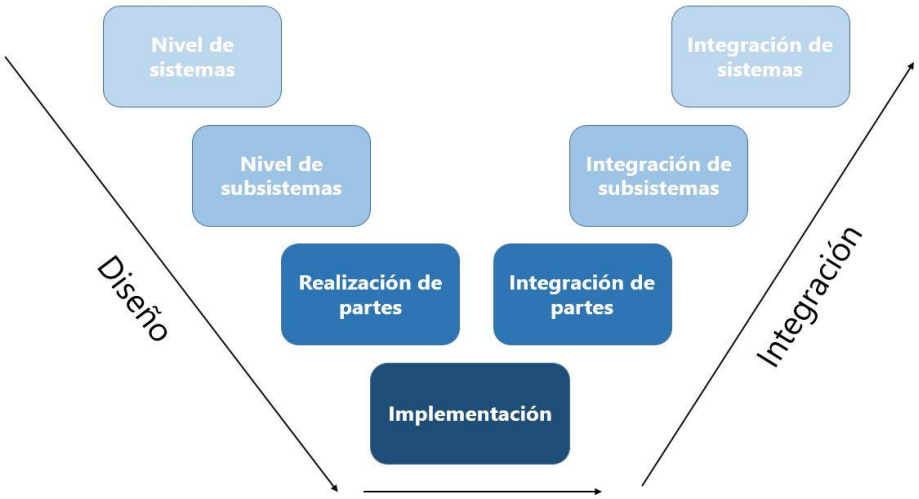

Ilustración 7 Metodología en V

Fuente: propia.

El ciclo A estará compuesto por las etapas descritas en la ilustración 7, cada etapa tiene un objetivo y proceso para el desarrollo de la presente investigación. A continuación, se explica la función de cada etapa, así como, sus niveles y subniveles.

4.6.1 ETAPA I: NIVEL DE LOS SISTEMAS

Para el diseño del robot autónomo con sistema *SLAM* se definen los siguientes sistemas: sistema de control, sistema de alimentación y sistema mecánico cada uno con su respetivo subsistema.

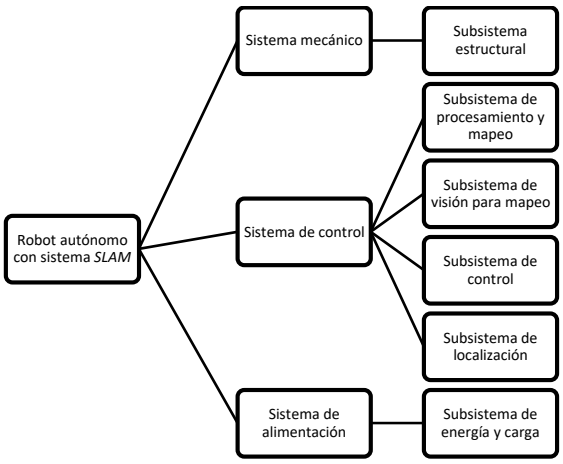

Ilustración 8 Nivel de sistemas y subsistemas.

Fuente: propia.

Sistema mecánico: este sistema está dividido en un subsistema: estructural. En el subsistema estructural se diseña la estructura en la cual estarán apoyados todos los subsistemas del robot, evaluándose la cantidad de peso la cual soportará sin llegar a puntos de ruptura o de tensiones que afecten el movimiento del robot.

Sistema de control: este sistema está dividido en cuatro subsistemas: procesamiento y mapeo, de control, de visión y localización. Este sistema está compuesto por componentes electrónicos, microprocesadores y un minicomputador que se encargará de leer la información enviada por el sensor *LIDAR* y de los codificadores rotativos. Dicha información será procesada y con la lógica programada en los diferentes microcontroladores y el minicomputador tomará decisiones para gestionar el movimiento haciendo uso de puentes H para el control de los motores.

Sistema de alimentación: este sistema está compuesto únicamente por el subsistema de energía y carga, es importante tomar en cuenta la capacidad y el tipo de batería utilizada para alimentar el robot. En este sistema se toma en consideración la capacidad de voltaje que suministra y la capacidad con la que éste cuenta.

4.6.2 ETAPA II: NIVEL DE LOS SUBSISTEMAS.

Al haber definido cada uno de los sistemas se procede a definir cada uno de los subsistemas.

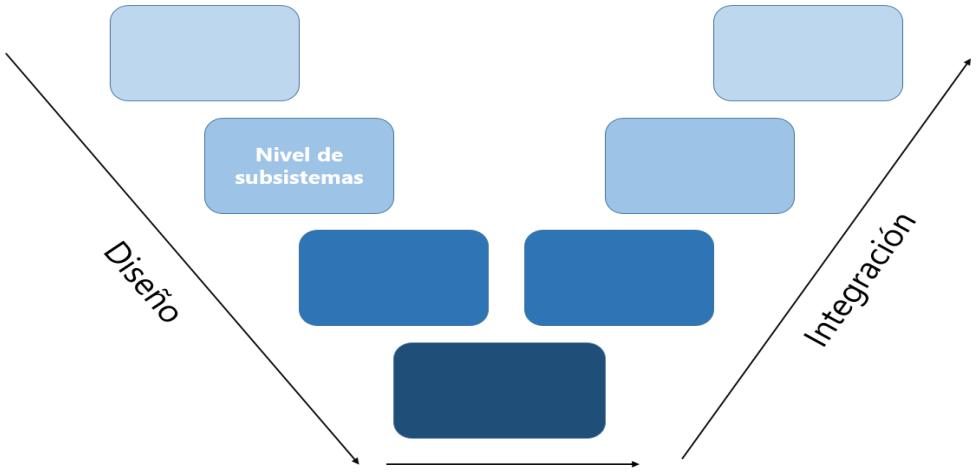

Ilustración 9 Etapa 2: nivel de subsistemas.

Fuente: propia.

4.6.2.1 Subsistema estructural

Está conformado por todas las piezas que se encuentren en el robot, que permita acoplar todos los componentes de forma correcta y que permita tener una movilización fluida al robot. Dicho subsistema requerirá de diferentes uniones y acoples que permita centralizar todo lo necesario para el funcionamiento autónomo del robot. Se tomará en consideración el material que mejor se adapte a las necesidades del prototipo.

4.6.2.2 Subsistema de procesamiento y mapeo.

Se requiere de microcontroladores que interpreten los pulsos enviados por los codificadores rotativos y también de un minicomputador que será el maestro el cual recopilará dicha información para posteriormente analizarla e interpretarla para poder crear un mapa de manera gráfica y tomará decisiones gracias al algoritmo implementado.

4.6.2.3 Subsistema de visión para el mapeo

Este subsistema se encargará de retroalimentar al sistema de procesamiento con el uso de su láser que le permitirá detectar obstáculos y estructuras de su entorno.

4.6.2.4 Subsistema de control

Está conformado por puentes H que interpretarán los pulsos enviados a sus entradas para activar los motores en el sentido necesario para que pueda movilizarse al lugar requerido.

4.6.2.5 Subsistema de localización

La localización es parte fundamental del robot para lograr un mapeo más acorde a la realidad, de igual manera le permitirá saber al robot la distancia que debe recorrer y con eso optimizar su recorrido. Para ello se requerirá de codificadores rotativos que proveerán al subsistema de procesamiento toda la información requerida para posterior toma de decisiones.

4.6.2.6 Subsistema de energía y carga

Se compone por la forma de obtención de energía con la que contarán los componentes electrónicos que se encontrarán en los otros subsistemas.

4.6.3 ETAPA III: REALIZACIÓN DE PARTES

Se ha definido cada uno de los subsistemas y los requisitos que poseen los mismos, se procedió a especificar los distintos materiales y componentes que se requieren para el desarrollo de los subsistemas.

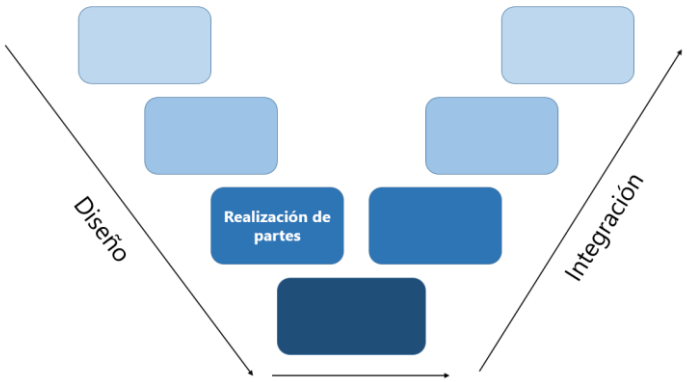

Ilustración 10 Etapa III: realización de partes

Fuente: propia.

4.6.3.1 Subsistema estructural

Para este subsistema se compararon los distintos materiales para la selección del chasis del robot, considerando su precio, resistencia y versatilidad a la hora de modificarlo. Uno de los materiales más utilizados para estos prototipos es el *ABS*, pero la desventaja es que este tipo de plástico es más frágil y como se pretende utilizar este robot para exploración, no sería la mejor decisión; es por ello por lo que se ha optado por una estructura que utiliza láminas delgadas de aleación de aluminio, el aluminio es muy utilizado en la creación de robots por sus propiedades físicas, químicas y mecánicas. Su densidad es muy reducida por ser un metal y presenta una excelente resistencia a la corrosión.

Material	DENSIDAD [g/cm3]	Resistencia máxima a la tensión [Mpa]	Módulo de elasticidad [Gpa]	Temperatura máxima [°C]	Elongación %
ABS	1.3	41.4	2.1	71-93	25
Aleación de aluminio 3003	2.8	200	71.7	660	30
NYLON 11	1.26	55.2	1.3	82-149	300

Tabla 1 Propiedades mecánicas de las aleaciones de aluminio.

Fuente: propia con datos recuperados de ingemecanica.com

En la tabla 1, se muestra una comparación entre dos de los materiales más utilizados en impresión 3D y la aleación de aluminio 3003. Como se puede observar las propiedades mecánicas de esta aleación permiten una mayor durabilidad, más resistencia al calor y una elongación más que aceptable. Agregar a ello que la ventaja de este aluminio en contra de otros metales es su gran resistencia a la corrosión.

La elongación de un material se entiende como el aumento de longitud que éste presenta al estar sometido a un esfuerzo. Entre menos elongación tiene el material, menos posibilidades de llegar a la rotura.

4.6.3.2 Subsistema de procesamiento y mapeo

Para este subsistema se seleccionó los microcontroladores que mejor se adaptan al proyecto. Para su selección se tomó en cuenta especialmente la compatibilidad con las librerías de *PID* y *ROS*, voltaje de entrada, cantidad de puertos y además de ello era de suma importancia que sean lo más compactos posibles debido a que eran necesario implementar uno para cada llanta.

Microcontrolador	Voltaje de entrada	Puerto *I2C*	Compatibilidad con Ros	Librería
Arduino Nano	5V	2	Sí	Sí
Pic18f45k22	5-9 V	2	No	Sí
Raspberry pi pico	3.3 V	22	Únicamente con ROS2	Sí

Tabla 2 Microcontroladores

Fuente: propia

En la tabla 2 se detallan las características de tres microcontroladores, *Arduino* nano, pic18f45k22 y *raspberry* pi pico. El microcontrolador seleccionado es el *Arduino* nano debido a que es compatible con *ROS*, posee una librería de *PID* para el control de los motores y de igual manera tiene lo necesario para realizar una comunicación por medio de *I2C* al maestro. Necesita de 5v de alimentación los cuales se obtendrán del puerto *USB* que posee el minicomputador.

Para el procesamiento de todos los nodos es necesario de un minicomputador el cuál cuente con un sistema operativo compatible con *ROS*, en este caso, *Linux (Ubuntu 18.02)*. Dicho microcomputador debe tener la potencia necesaria para ejecutar diversos programas tanto en segundo como en primer plano. Para ello se ha realizado una comparación entre dos de los mejores minicomputadores en la actualidad de bajo costo que cumplen con las características antes mencionadas: *Raspberry pi 4* y *Nvidia Jetson nano*.

	Raspberry pi 4	*Nvidia Jetson Nano*
CPU	Quad-core ARM Cortex-A72 64-bit @ 1.5 Ghz	Quad-core ARM Cortex-A57 64-bit @ 1.42 Ghz
GPU	Broadcom VideoCore VI (32-bit)	NVIDIA Maxwell w/ 128 CUDA cores @ 921 Mhz
Memoria RAM	4Gb DDR4	2GB DDR4
Conectividad	Gigabit ethernet, wifi 802.11ac	Ethernet incorporado, adaptador *USB* para wifi
USB	2x *USB 3.0* 2x *USB 2.0*	1x USB 3.0 Tipo A, 2x USB 2.0, 1x USB 2.0 micro-B
GPIO	40	40
OS	*Raspbian, Ubuntu 20.04, Windows 10 IoT*	*Ubuntu (18.02)*
Precio	*$55*	*$59*

Tabla 3 Minicomputadores

Fuente: propia.

Como se puede observar en la tabla 3, es una comparación muy reñida puesto que el parecido de estos dos minicomputadores es muy evidente. El *raspberry* posee un procesador de nueva generación el cual alcanza una frecuencia de reloj un poco mayor que la del *jetson* nano, también posee una mayor cantidad de memoria *RAM* y más diversidad de sistemas operativos, sin embargo, se optó por un *Nvidia jetson* nano debido a que posee una tarjeta gráfica mucho mejor que la del *raspberry*, en concreto una Maxwell con 128 *CUDA cores*. Esta es una gran diferencia puesto que para poder realizar los mapas el minicomputador deberá apoyarse de una gráfica más potente, agregar a ello que gracias a esta gráfica se puede programar más inteligencia artificial al robot. La diferencia de la *RAM* no es tan importante puesto que gracias a la optimización de *Nvidia*, se puede utilizar parte de la memoria *ROM* para compensar la *RAM*.

4.6.3.3 Subsistema de visión para mapeo

El subsistema de visión es parte fundamental del robot, éste le permitirá al robot ver los obstáculos y proveerá la información necesaria para el mapeo. La característica más importante es que este subsistema le permita al robot una visión acertada y que no sea propenso al ruido. Es por ello se comparan dos sensores *LIDAR* de bajo costo para la implementación al prototipo.

Sensor	Rango de detección [m]	Voltaje de entrada [V]	Frecuencia de exploración[Hz]	Ángulo de aceptación	Tipo
TFmini-S *LIDAR*	0.1 – 12 m	5V	-	2 °	Unidireccional
RPLIDAR A1M8 2D	0.15 – 12 m	5V	2-10 Hz	1 °	360°

Tabla 4 Sensores LIDAR

Fuente: propia.

El *RPLIDAR* A1M8 2D es un claro ganador de esta comparativa, éste permite al robot una visión en 360 y con ello un mapeo más rápido, a diferencia que e *TFmini* por ser unidireccional el robot necesitará moverse en todos los sentidos para que se realice el mapa. Con el *RPLIDAR* será instantáneo por ende permitirá que el minicomputador tome decisiones con mayor velocidad y no descuidará ni una sola parte del entorno.

4.6.3.4 Subsistema de control

Para el subsistema de control se necesita de un módulo de control para los motores DC 12v que incorpora el prototipo. Para este sistema se ha seleccionado el módulo L298N por su alta compatibilidad con los microcontroladores Arduino y además ofrece un control bastante robusto debido que permite motores de 5 a 35 V.

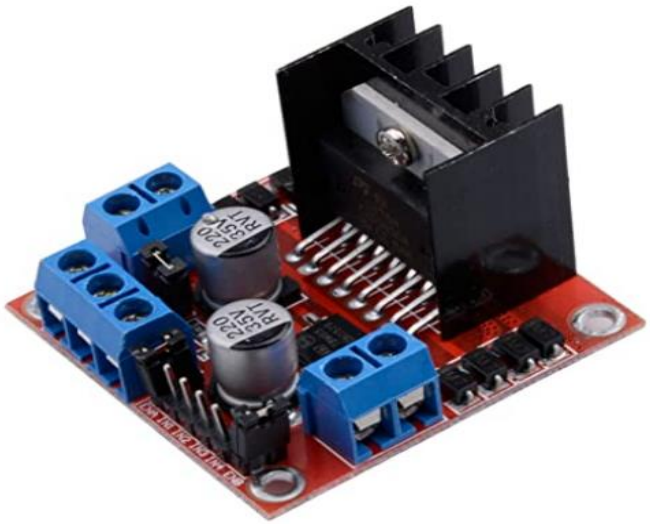

Ilustración 11 Módulo L298N

Fuente: Amazon (2020)

4.6.3.5 Subsistema de localización

Este sistema es fundamental debido a que el mapeo no se puede realizar sin la localización del robot, para ello se han implementado codificadores rotativos debido a que estos por medio de sus pulsos nos dan los datos de odometría necesarios para el diseño del prototipo. Otra forma de obtener odometría es con una cámara programada con inteligencia artificial, pero se ha descartado ya que las cámaras son muy propensas al ruido ambiental y dependen de tener un sistema de visión nocturna en condiciones de poca luz.

Modelo	**Voltaje de entrada [V]**	**Número de pulsos**	**Frecuencia de respuesta [HZ]**
Codificador rotativo incremental HN3806	5 – 24 V	100/200/360/400/600	30 Khz
Codificador rotatorio KY-040	5 V	20	30 Khz

Tabla 5 Codificadores incrementales

Fuente: propia.

El codificador incremental HN3806 brinda una lectura más precisa debido a la cantidad de pulsos que este codificador puede generar, una codificación mayor es equivalente a una localización más acertada.

4.4.3.6 Subsistema de alimentación

Para el subsistema de alimentación se tomó en cuenta la capacidad de la batería, el voltaje suministrado, las dimensiones y el peso. La batería debe ser recargable y suministrar 12v necesarios para alimentar los motores *DC*. El *jetson* nano necesita 5v a 2.5 Amps es por ello por lo que se hará uso de un *powerbank* de 10,000 mah, con el *jetson* nano alimentado se energizará por medio de sus puertos USB el sensor *LIDAR,* los *arduinos nano* y el módulo wifi.

Modelo	Voltaje suministrado [V]	Amperios/hora [mah]	Dimensiones [pulgadas]	Peso [g]	Precio
Goldbat 3S	11.1 – 12.5 v	5200 mah	5.35x1.65x1.18	368 g.	$39.99
Povway lipo	11.1 – 12.5 v	4500 mah	5.31x1.57x0.98	312 g.	$31.99
E-Flite	11.1 – 12.5 v	2200 mah	7.3 x 2.4 x 1.1	204 g.	$27.99

Tabla 6 Baterías LIPO 12v.

Fuente: propia.

La batería seleccionada es la *E-FLite* porque a pesar de que no es la que mayor capacidad posee, es la más barata y con menor peso, haciéndola mejor opción para la implementación en el prototipo.

4.6.4 ETAPA IV: INTEGRACIÓN DE PARTES.

Definidas las partes de todos los sistemas que componen el prototipo y definidos los componentes a utilizar en el desarrollo del proyecto se comienza a realizar la segunda parte del ciclo "A" que consiste en realizar las pruebas que corroboren el correcto funcionamiento de cada componente.

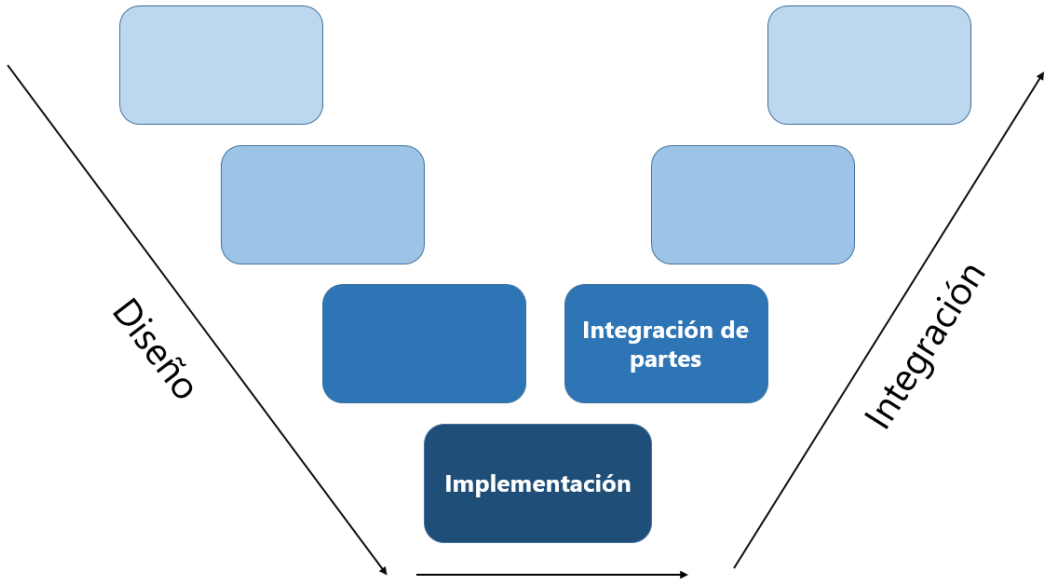

Ilustración 12 Integración de partes.

Fuente: propia.

4.6.4.1 Subsistema estructural

Para las pruebas del diseño estructural del prototipo se ha utilizado el *software* de modelado 3D *Solidworks,* ya que este *software* nos permite realizar estudios de movimiento, pruebas de tensiones y peso de la estructura y cada parte que la conforman. Se obtiene el peso mediante la ecuación de Newton que nos dice:

$$W = m * g$$

Ecuación 14 Ecuación de peso

Fuente: Mott (2006)

Donde:

W= Peso del objeto

m= masa del objeto

g= fuerza de gravedad.

4.6.4.2 Subsistema de procesamiento y mapeo

Se realizaron pruebas a los microcontroladores utilizando la librería de *PID* y en el monitor serial leyendo los valores que nos arrojaban los codificadores incrementales. Las pruebas en el jetson nano consistieron en la ejecución del sistema operativo *Ubuntu 18.04* y la instalación de *ROS*. Se ejecutó el simulador *Hector SLAM* y el compilador *CloudShell editor* para la programación de lógica en *Python*.

```
#include <PinChangeInt.h>
#include <Wire.h>
#include <PID_v1.h>
#define encodPinA1      2
#define encodPinB1      8
#define M1              9
#define M2              10
double kp =1, ki =20 , kd =0;
double input = 0, output = 0, setpoint = 0;
unsigned long lastTime,now;
volatile long encoderPos = 0,last_pos=0,lastpos=0;
PID myPID(&input, &output, &setpoint, kp, ki, kd,DIRECT);
void setup() {
  pinMode(encodPinA1, INPUT_PULLUP);
  pinMode(encodPinB1, INPUT_PULLUP);
  attachInterrupt(0, encoder, FALLING);
  TCCR1B = TCCR1B & 0b11111000 | 1;

  myPID.SetMode(AUTOMATIC);
  myPID.SetSampleTime(1);
  myPID.SetOutputLimits(-255, 255);

  Wire.begin(8);
  Wire.onRequest(requestEvent);
  Wire.onReceive(receiveEvent);
  Serial.begin (9600);

}
void loop() {
   now = millis();
   int timeChange = (now - lastTime);
   if(timeChange>=500 )
   {
      input = (360.0*1000*(encoderPos-last_pos)) /(1856.0*(now - lastTime));
      lastTime=now;
      last_pos=encoderPos;
   }

  myPID.Compute();
  pwmOut(output);
  delay(10);
}
```

```
void pwmOut(int out) {
  if (out > 0) {
    analogWrite(M1, out);
    analogWrite(M2, 0);
    Serial.println(out);
  }
  else {
    analogWrite(M1, 0);
    analogWrite(M2, abs(out));
    Serial.println(out);
  }
}
void encoder()  {
  if (PINB & 0b00000001)    encoderPos--;
  else                      encoderPos++;
}
void requestEvent() {
  int8_t s;

  s= (360.0*(encoderPos-lastpos))/1856.0;
  lastpos=encoderPos;
  Wire.write(s);
}
void receiveEvent(int howMany)
{
  uint8_t a,b;
  a = Wire.read();
  b = Wire.read();
  setpoint= (double)((b<<8)|a);
}
```

Ilustración 13 Programa encoder Izquierdo ARDUINO IDE

Fuente: Propia

```
#include <PinChangeInt.h>
#include <Wire.h>
#include <PID_v1.h>
#define encodPinA1      2
#define encodPinB1      8
#define M1              9
#define M2              10

double kp =1, ki =20 , kd =0;
double input = 0, output = 0, setpoint = 0;
unsigned long lastTime,now;
volatile long encoderPos = 0,last_pos=0,lastpos=0;
```

```
PID myPID(&input, &output, &setpoint, kp, ki, kd,DIRECT);
void setup() {
  pinMode(encodPinA1, INPUT_PULLUP);
  pinMode(encodPinB1, INPUT_PULLUP);
  attachInterrupt(0, encoder, FALLING);
  TCCR1B = TCCR1B & 0b11111000 | 1;

  myPID.SetMode(AUTOMATIC);
  myPID.SetSampleTime(1);
  myPID.SetOutputLimits(-255, 255);

  Wire.begin(9);
  Wire.onRequest(requestEvent);
  Wire.onReceive(receiveEvent);
  Serial.begin (9600);
}
void loop() {
   now = millis();
   int timeChange = (now - lastTime);
   if(timeChange>=500 )
   {
      input = (360.0*1000*(encoderPos-last_pos)) /(1856.0*(now - lastTime));
      lastTime=now;
      last_pos=encoderPos;
   }

  myPID.Compute();
  pwmOut(output);
  delay(10);
}
void pwmOut(int out) {
 // to H-Bridge board
  if (out > 0) {

    analogWrite(M1, 0);
    analogWrite(M2, out);
    Serial.println(out);
  }
  else {
    analogWrite(M1, abs(out));
    analogWrite(M2, 0);
    Serial.println(out);// drive motor CCW
  }
}
```

```
void encoder()  {
  if (PINB & 0b00000001)    encoderPos++;
  else                      encoderPos--;
}

void requestEvent() {
  int8_t s;

  s= (360.0*(encoderPos-lastpos))/1856.0;
  lastpos=encoderPos;
  Wire.write(s);
}
void receiveEvent(int howMany)
{
  uint8_t a,b;
  a = Wire.read();
  b = Wire.read();
  setpoint= (double)((b<<8)|a);
}
```

Ilustración 14 Programa encoder derecho *ARDUINO IDE*

Fuente: Propia

En la ilustración 13 y 14 se observa la programación implementada en cada codificador para obtener la retroalimentación deseada y con ello, lograr la localización del robot.

4.6.4.3 Subsistema de visión para el mapeo.

Se realizaron pruebas al sensor *LIDAR* con el *software* dedicado que éste posee: *Mapper by Slamtec.* Las pruebas fueron exitosas ya que nos creó un pequeño mapa del entorno en el cual estaba funcionando, utilizando el parámetro estándar de frecuencia que está dado en 6Hz.

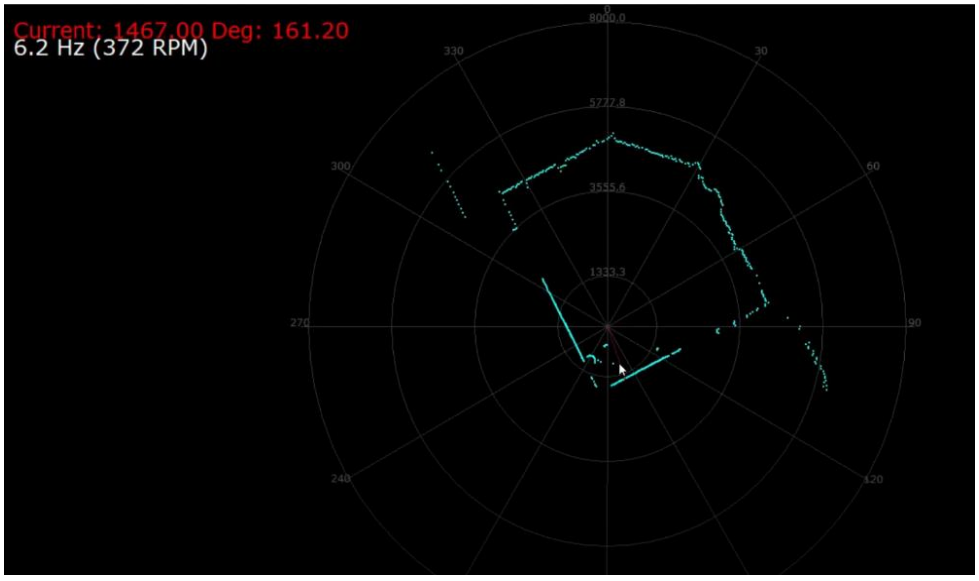

Ilustración 15 Mapeo realizado en el programa nativo Mapper.

Fuente: propia.

4.6.4.4 Subsistema de control

Se energizó el módulo L298N utilizando los 12v de la batería, se comprobó con un multímetro que hacía el paso de corriente a los motores cuando el *Arduino* le enviaba la señal.

4.6.4.5 Subsistema de localización

Con la librería y el programa en el microcontrolador se energizaron los codificadores rotativos y se comprobó que estos generaban los pulsos necesarios para interpretar su movimiento, una precisión más que correcta comprobada en el monitor serial de *Arduino IDE*.

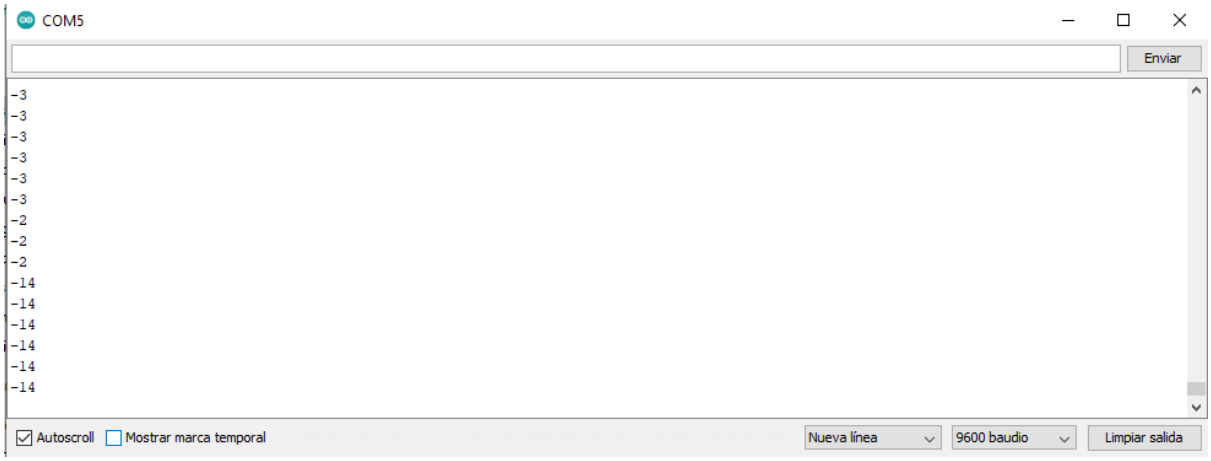

Ilustración 16 *Encoder* Izquierdo

Fuente: Propia

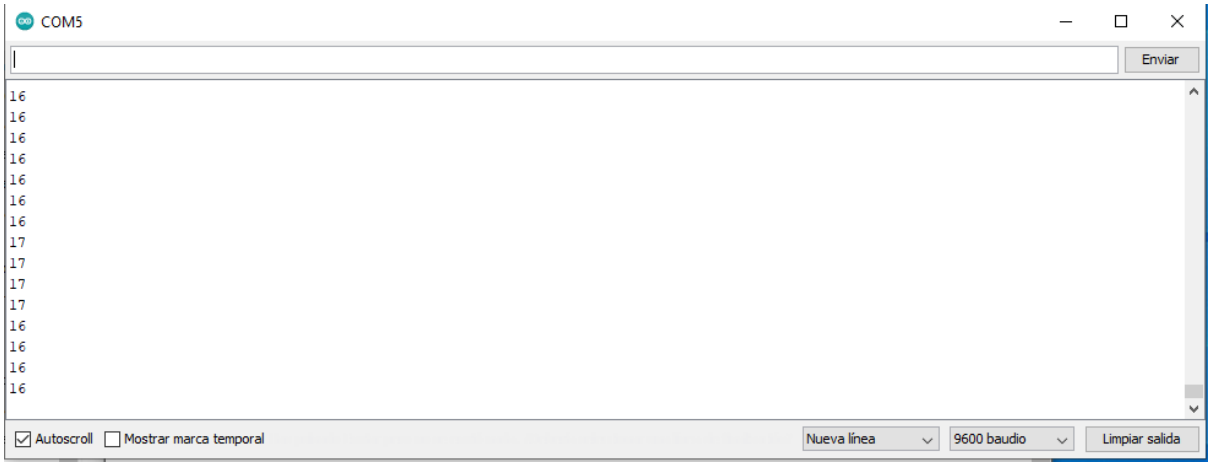

Ilustración 17 *Encoder* Derecho

Fuente: Propia

4.6.4.6 Subsistema de alimentación

Con un multímetro se corroboró que la batería daba el voltaje deseado y junto a ello se estimó el tiempo que necesita la batería y el *powerbank* para recargarse totalmente.

4.6.5 ETAPA V: INTEGRACIÓN DE LOS SUBSISTEMAS

Posterior a realizar las pruebas de los componentes que utilizan cada subsistema se procede a la etapa V que consiste en observar el funcionamiento de los subsistemas en conjunto y utilizando parte de los algoritmos implementados para el resultado final

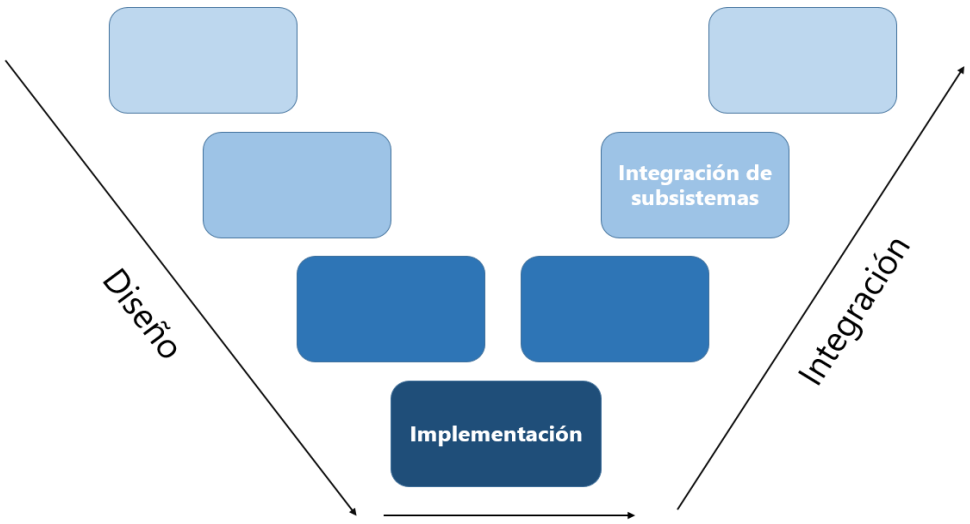

Ilustración 18 Integración de los subsistemas.

Fuente: propia.

4.6.5.1 Subsistema estructural

Posterior a los estudios realizados a la estructura del robot se procedió a realizar el ensamblado completo incluyendo las diferentes uniones para asegurar el acople correcto de las piezas requeridas para dicho subsistema.

4.6.5.2 Subsistema de procesamiento y mapeo

Se procedió a realizar el diagrama de flujo el cual orientará el proceso de la programación del microcontrolador y del minicomputador. Para el minicomputador al utilizar *ROS* se realizó el diagrama de flujo especificando cada nodo y enumerando los paquetes que serán utilizados para la lógica del robot, dichos paquetes contienen las instrucciones que el robot recibirá y éstas serán programadas en diferentes lenguajes de programación.

4.6.5.3 Subsistema de visión para mapeo

Se procedió a realizar una representación gráfica del trazado que realiza el sensor *LIDAR* para obtener la información de los obstáculos.

4.6.5.4 Subsistema de control

Se realizó el esquema de conexión que conecta a dicho subsistema con los otros que se encargan de darle instrucciones.

4.6.5.5 Subsistema de localización

Se realizaron cálculos necesarios para realizar una estimación de la posición en tiempo real del robot considerando las velocidades individuales de cada motor. Se consideró la velocidad lineal, la velocidad angular, longitud del robot y el radio de cada llanta.

$$w = R\frac{v_r - v_l}{L}$$

Ecuación 15 Velocidad individual de cada llanta

Fuente: (L. E. Solaque, M. A. Molina, E. L. Rodríguez., 2014).

Donde:

- W = velocidad angular.
- R = radio de los engranes.
- L = Longitud entre ruedas.
- V_r = Velocidad de la rueda derecha.
- V_L = Velocidad de la rueda izquierda.

Para obtener la distancia recorrida se calcula con la siguiente ecuación:

$$D = (\frac{R * 2\pi}{ppv})N$$

Ecuación 16 Distancia recorrida

Fuente: (VALENCIA V., JHONNY A., & MONTOYA O., ALEJANDRO, & RIOS, LUIS HERNANDO, 2009).

Donde:

- D = Distancia recorrida.
- R = Radio.
- ppv = Pulsos por vuelta.
- N = Número de pulsos recorridos.

Para obtener la posición en el eje x del robot se calcula con la siguiente ecuación:

$$\dot{x} = vcos\emptyset$$

Ecuación 17 Posición en eje x del robot

Fuente: (L. E. Solaque, M. A. Molina, E. L. Rodríguez., 2014).

Donde:

- $\dot{x}$ = Posición en el eje x del robot.
- v = Velocidad.
- $\emptyset$ = Angulo.

Para obtener la posición en el eje y del robot se calcula con la siguiente ecuación:

$$\dot{y} = vsen\emptyset$$

Ecuación 18 Posición en eje y del robot

Fuente: (L. E. Solaque, M. A. Molina, E. L. Rodríguez., 2014).

Donde:

- $\dot{y}$= Posición en el eje y del robot.
- v = Velocidad.
- $\emptyset$ = Angulo.

4.6.5.6 Subsistema de alimentación

Se realizó el cálculo correspondiente para determinar la autonomía de la batería seleccionada utilizando los consumos que presentan cada uno de los componentes de todo el sistema.

$$H = \frac{Vb(Ib)}{Vb(Ic)}$$

Ecuación 19 Autonomía de la batería

Fuente: (Coelectrix, 2019)

Donde:

H=autonomía en horas

Vb= voltaje de la batería

Ib= Miliamperios hora de la batería

Ic= consumo de los componentes

4.6.6 ETAPA VI: INTEGRACIÓN DE LOS SISTEMAS

Finalizadas las etapas de prueba realizadas a cada componente, parte y subsistema se procede a realizar la etapa final del ciclo A, etapa VI integración de los sistemas.

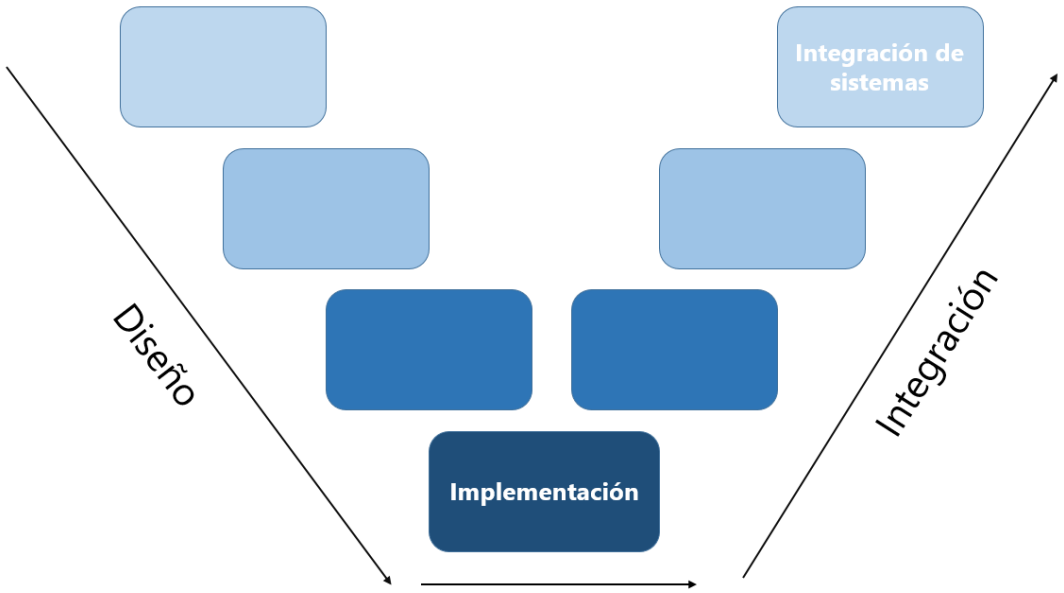

Ilustración 19 Etapa VI: integración de sistemas.

Fuente: propia.

En esta última etapa se acoplan todos los sistemas que componen el robot y con ello el diseño de lo que se pretende será el diseño final del robot autónomo con sistema *SLAM*.

4.7 CRONOGRAMA DE ACTIVIDADES

Se realizó un cronograma de actividades detallando cada tarea a realizar a lo largo de la presente investigación en base a un tiempo de trabajo de 10 semanas. Formato: MM/DD/AA

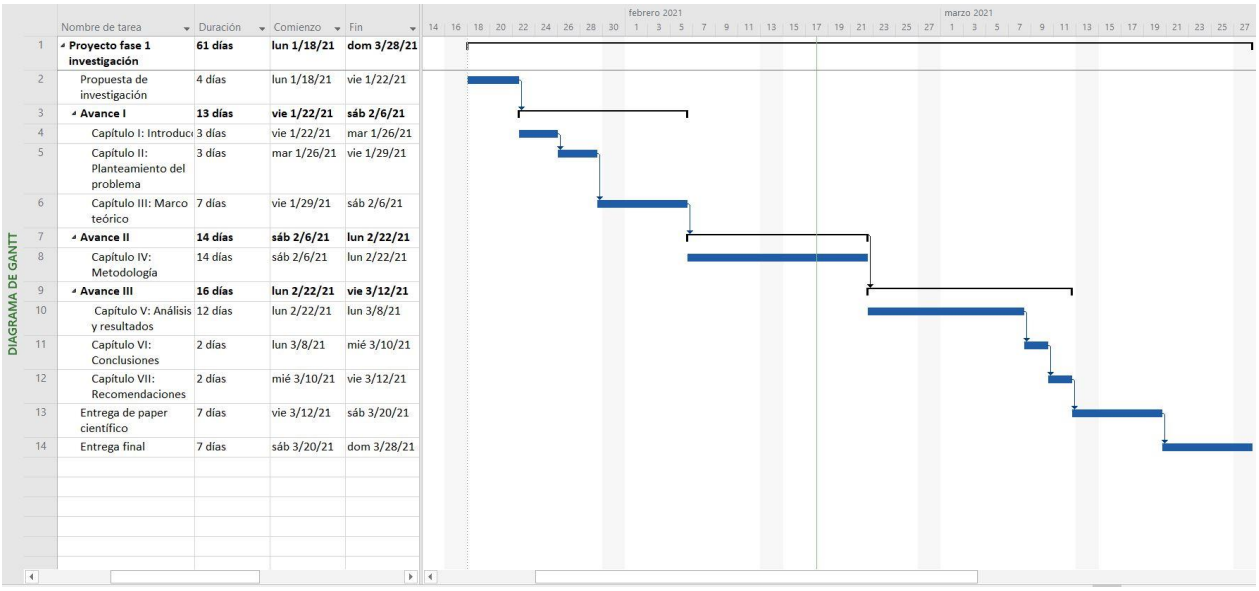

Ilustración 20 Cronograma de actividades.

Fuente: propia.

V. ANÁLISIS Y RESULTADOS

En este capítulo se mostrarán los resultados obtenidos en la presente investigación, haciendo énfasis en las preguntas de investigación antes planteadas. Para los resultados de la investigación se apoyará de las simulaciones previamente realizadas a cada uno de los componentes y sistemas necesarios para el prototipo, de igual manera se mostrarán los resultados teóricos de los cálculos realizados que ayudarán a demostrar el funcionamiento del proyecto.

5.1 SISTEMA ESTRUCTURAL

El sistema estructural es en donde van acoplados todos los componentes que conforman el robot, para dicho sistema se realizó el diseño en *Solidworks* al igual que el análisis estático y dinámico. En cuanto a la estructura se utilizó una de aleación de aluminio tipo tanque para permitir al robot navegar en zonas un poco más complicadas, antes de adquirir dicha estructura se le realizaron pruebas en Solidworks para estar seguros que se logrará un funcionamiento correcto.

5.1.1 Análisis estático

El análisis estático se basó en los cálculos realizados utilizando la ecuación #14, la estructura se subdivide en dos partes, parte superior y parte inferior, en la parte superior está acoplado el sensor *LIDAR* 2D y la batería. El resultado es un peso de 4.56 N. Para la parte inferior se acoplan los arduinos, el jetson nano, el puente H y el regulador de voltaje.

En las simulaciones se utilizó un peso de 5 N para calcular los desplazamientos en [mm].

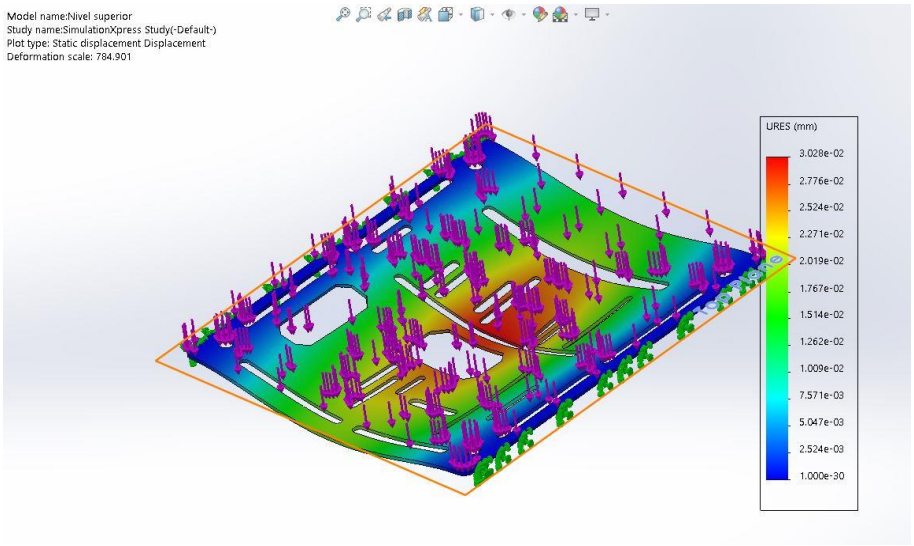

Ilustración 21 Desplazamiento en la parte superior

Fuente: propia.

Como se puede observar en la ilustración 21 el desplazamiento es muy pequeño y no se encuentra ningún punto de ruptura en la estructura. Su valor máximo ocurre en el centro dando un resultado de 3.02×10^{-2} mm y su valor mínimo de desplazamiento ocurre en las orillas con un valor de 2.543×10^{-3} mm.

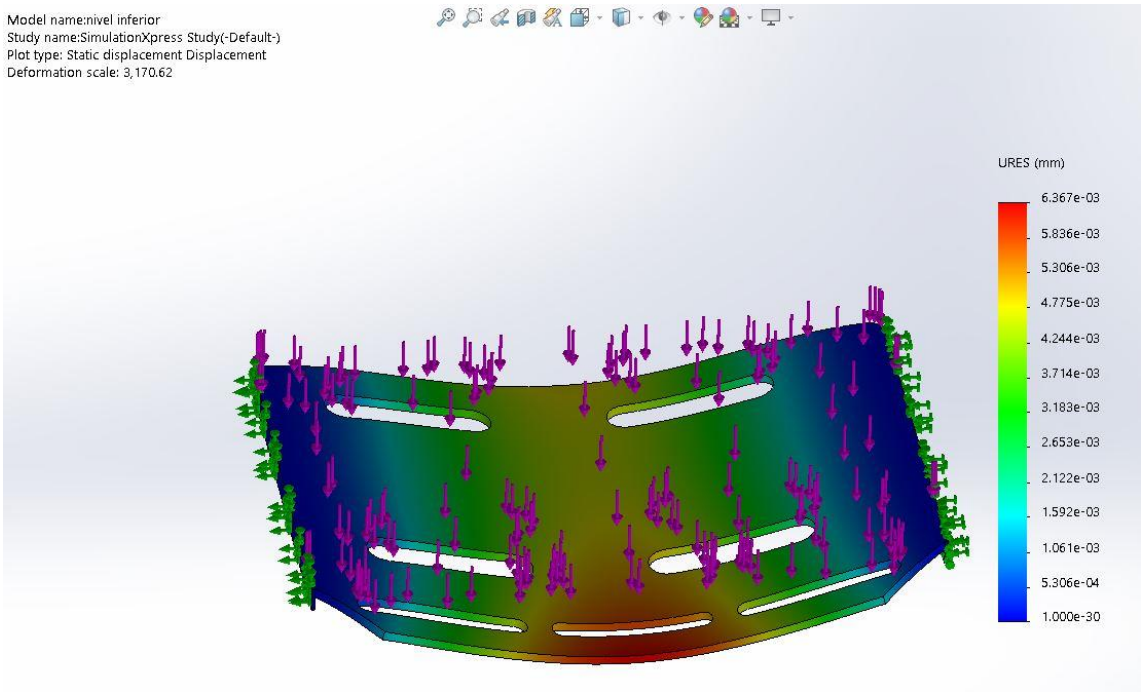

Ilustración 22 Desplazamientos en la parte inferior

Fuente: propia.

Como se puede observar en la ilustración 22 el desplazamiento en la parte inferior es muy pequeño y no se encuentra ningún punto de ruptura en esta parte de la estructura. Su valor máximo es de 6.357×10^{-3} mm, dicho desplazamiento ocurre en una pequeña parte del frente a la mitad de la pieza y su valor mínimo de desplazamiento que ocurre en la orilla es de 5.306×10^{-4} mm.

Conocer los desplazamientos es importante al igual que las tensiones a las cuales están sometidas las partes de la estructura y es por ello que se realizó una prueba de Von misses a cada pieza en la cual estarán acoplados los componentes.

Se utilizó para la simulación la aleación de aluminio 3003 y los resultados se muestran en las ilustraciones 23 y 24 respectivamente.

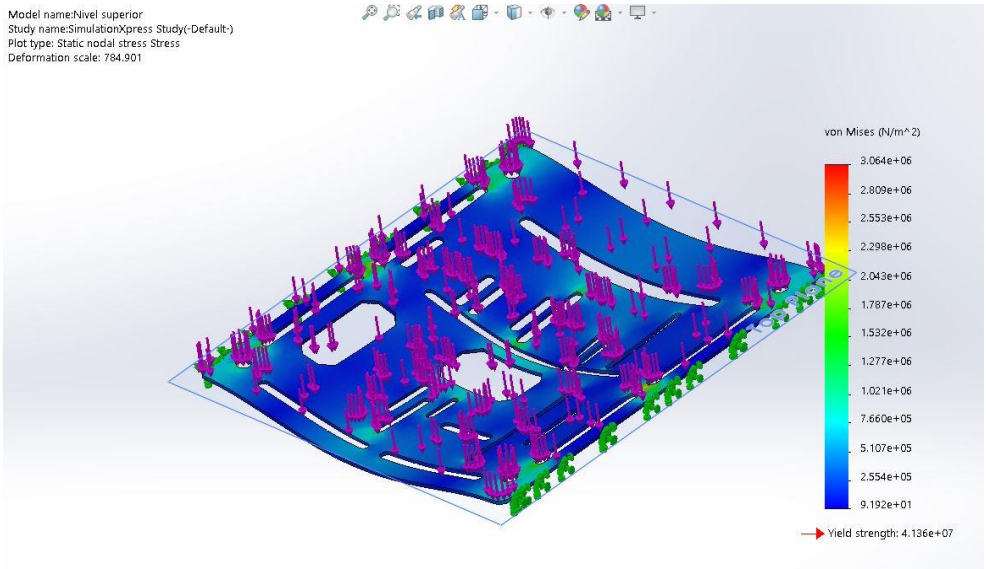

Ilustración 23 Prueba de Von Misses en parte superior

Fuente: propia.

En la ilustración 23 se observan los resultados de las pruebas de von misses en la pieza que conforma la parte superior del robot. Su valor máximo es de 3.064×10^{-6} N/m^2 y el valor mínimo de 2.554×10^{-6} N/m^2.

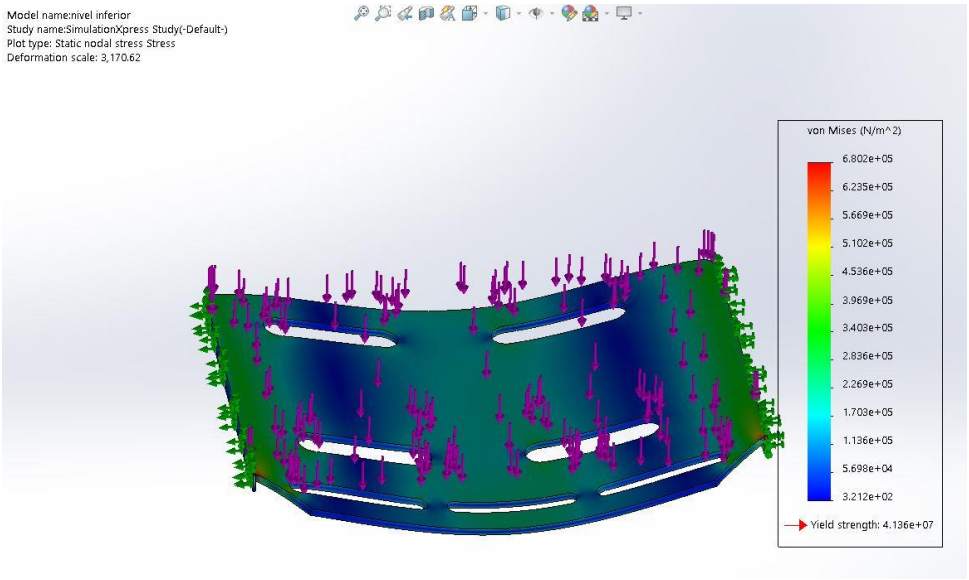

Ilustración 24 Prueba de Von Misses en la parte inferior

Fuente: propia.

En la ilustración 24 se observan los resultados de las pruebas de von misses en la pieza que conforma la parte inferior del robot. Su valor máximo es de 6.802×10^{-5} N/m^2 y el valor mínimo de 5.698×10^{-4} N/m^2.

5.1.2 Análisis dinámico

Para el análisis dinámico se utilizó la herramienta de *Solidworks* de estudios de movimiento, con los datos obtenidos en dichas simulaciones se procedió a realizar los cálculos en la ecuación #15 para conocer la velocidad individual y la ecuación #16 para calcular la distancia recorrida.

Los datos son los siguientes:

Radio 1.5 cm, longitud entre ruedas o engranes 20 cm y velocidades lineares promediadas de 1.43 mm/sec. Utilizando la ecuación #15 tenemos una velocidad angular individual de 0.0107 deg/segundos o 38.61 deg/hora.

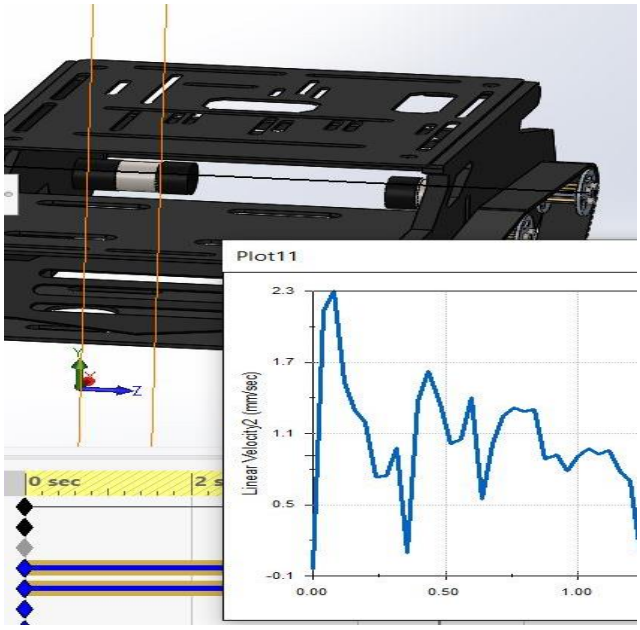

Ilustración 25 Velocidad linear en el engrane

Fuente: propia.

Como se puede observar en la ilustración 25 los resultados de la velocidad linear en cada engrane tiene un pico de 2.3 mm/seg al inicio y progresivamente baja hasta 0.5 mm/seg, promediando una velocidad linear de 1.43 mm/seg.

Para la distancia recorrida se utiliza la ecuación 16 con los siguientes datos: radio 1.5 cm, 30 pulsos por vuelta y 20 número de pulsos recorridos, con estos datos se obtiene una distancia de 6.28 cm, todo ello a nivel teórico.

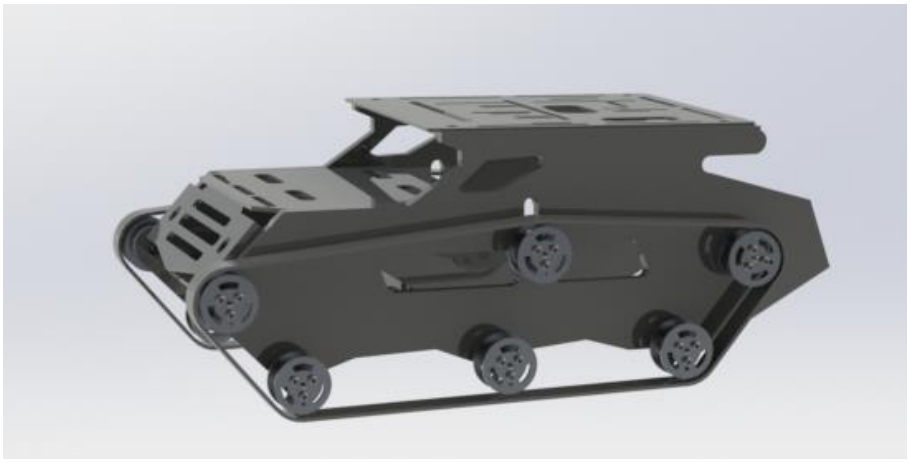

Ilustración 26 Estructura final del prototipo

Fuente: propia.

En la ilustración 26 se puede observar un renderizado del diseño final del sistema estructural del prototipo en el cual se ha basado para realizar todos los análisis y cálculos antes descritos.

5.2 SENSOR *LIDAR 2D* Y SUS LIMITACIONES

El sensor *LIDAR* es uno de los componentes más importantes del prototipo, su funcionamiento ha sido más que correcto y a pesar de sus limitaciones, ha permitido realizar modelos gráficos muy acertados a la realidad. Se realizaron distintas pruebas al sensor LIDAR 2D el cual se implementó en la investigación, según los parámetros en los cuales se basa el este sensor, se obtuvieron los siguientes resultados en las diferentes pruebas realizadas.

Un aspecto a considerar es el rango del sensor *LIDAR 2D* implementado, en este caso el sensor es de 360°, sin embargo, hay partes en las cuales la precisión del sensor no es la más correcta como es el caso cuando el láser se encuentra en el grado 90.

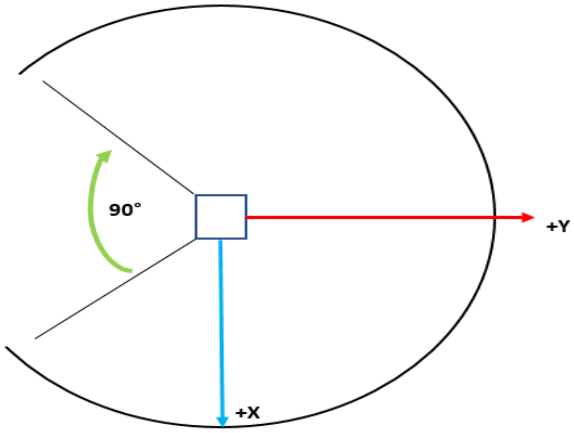

Ilustración 27 Rango del sensor LIDAR 2D

Fuente: Propia

El sensor LIDAR 2D utilizado en la investigación hace uso de un solo plano de láseres para capturar las dimensiones X e Y. Esto limita a que sea mayormente utilizado para tareas de detección y alcance, pero esto no significa que no puede ser utilizado en tareas más complejas.

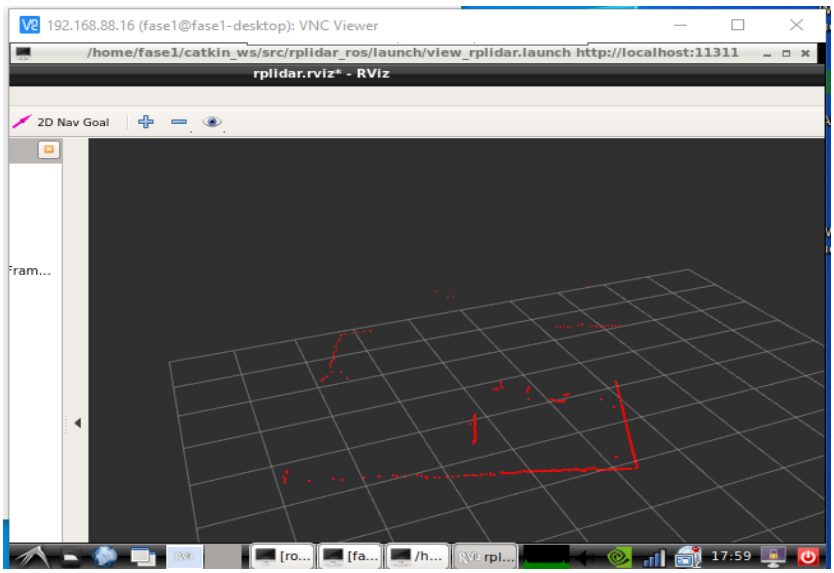

Ilustración 28 Sensor LIDAR 2D utilizando ros para obtener una vectorización del lugar en tiempo real.

Fuente: Propia

Como se puede observar en la ilustración 28 el sensor LIDAR 2D se ve limitado a solo vectorizar el área cercana a él por lo cual es preciso estarlo moviendo hacia las nuevas áreas para que sean vectorizadas. Con ello se asegura que no tendrá muchos problemas cuando se esté movilizando el robot. Cabe destacar que para la lectura del sensor *LIDAR* es muy fácil de obtener ya que el sensor es catalogado "*plug n play*" ya que solo fue necesario descargar el archivo ejecutable de Ubuntu que el mismo proveedor del sensor brinda, de igual manera, si se desea utilizar Windows el sensor cuenta con un programa base llamador Mapper

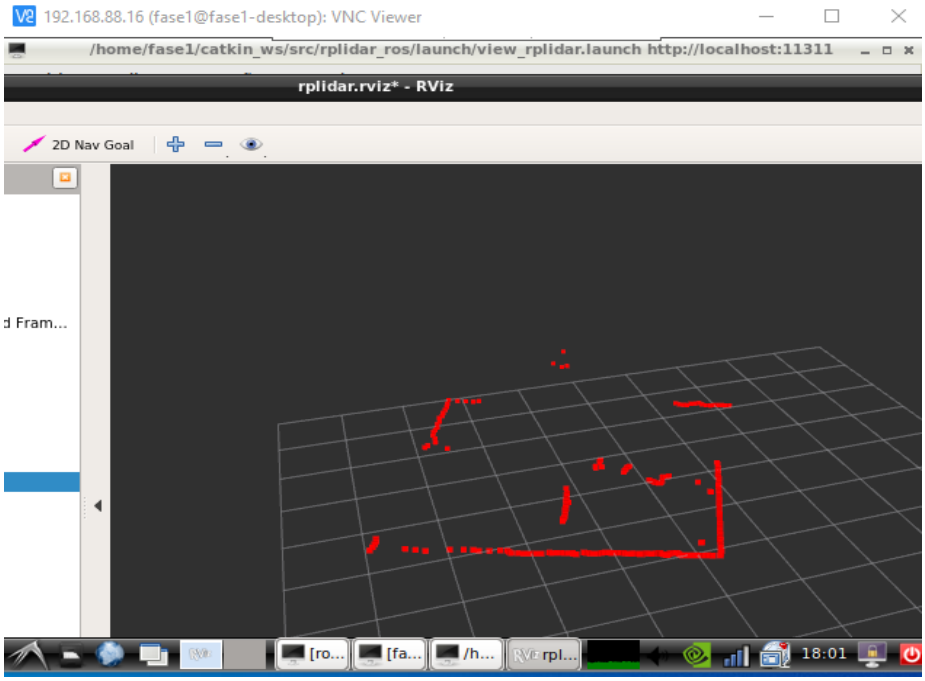

Ilustración 29 Sensor LIDAR 2D Utilizando ROS para obtener una vectorización de lugar en tiempo real con una densidad de escaneo aumentada.

Fuente: Propia

Como se puede observar en la ilustración 29 el sensor LIDAD 2D al variar la densidad en su modo de escaneo el entorno se vuelve más claro esto debido a que el láser del sensor toma menos puntos, pero con una densidad mayor, dicha densidad fue manipulada mediante el *software* de Ubuntu llamado *Rviz*, que es el cual provee de más herramientas para obtener un mapa más realista.

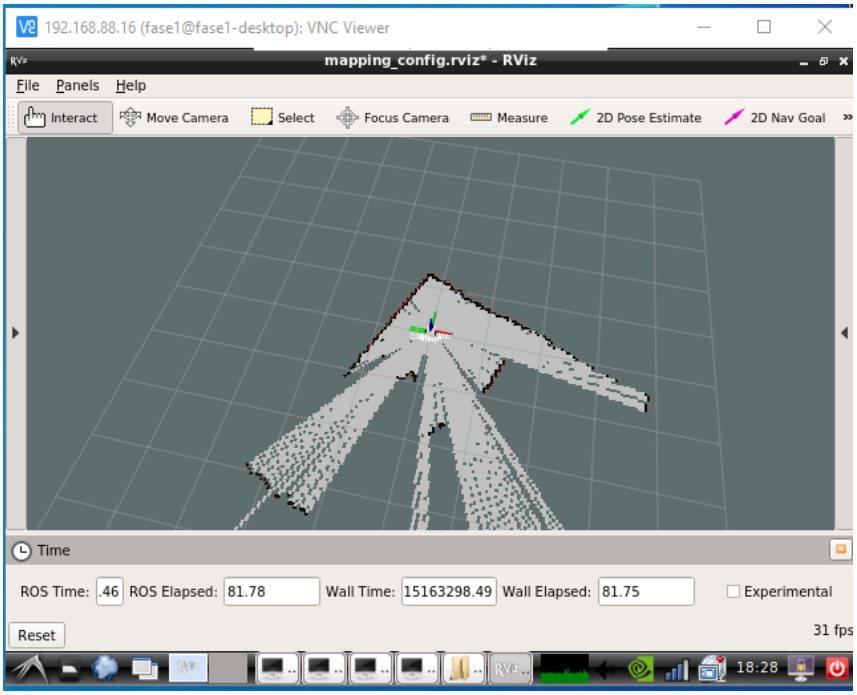

Ilustración 30 Sensor LIDAR 2D utilizando el programa de mapeo y haciendo uso de ROS para la interpretación de los datos.

Fuente: propia.

Gracias al uso del programa de mapeo y ejecutando la librería ROS a la vez podemos crear un entorno más detallado, a continuación, se mostrará una imagen donde se puede observar el puesto del sensor y como ha interpretado éste el entorno.

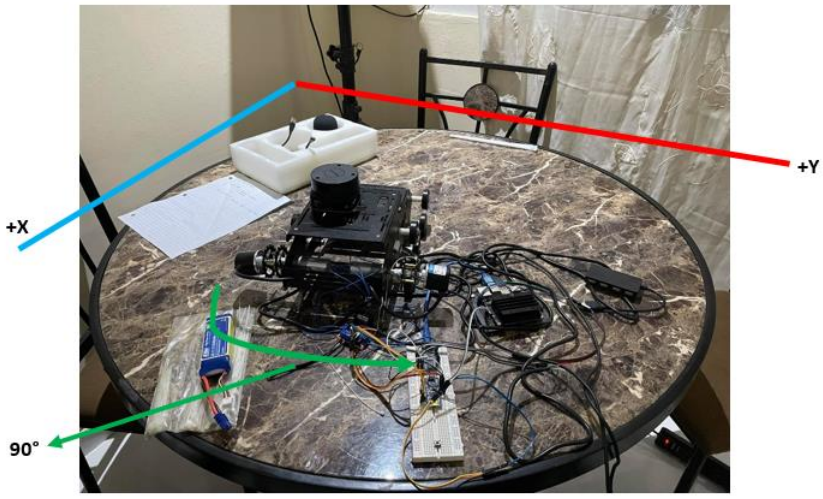

Ilustración 31 Ambiente en el cual se está realizando el mapeo

Fuente: propia.

Se describe las coordenadas que el sensor está captando en ese momento dando como resultado el mapa del entorno mostrado en la ilustración 31 que se puede observar anteriormente se observa lo descrito al inicio que el sensor no tiene acceso a información 90° detrás de su módulo principal esto sucede, aunque el láser este haciendo una rotación de 360°

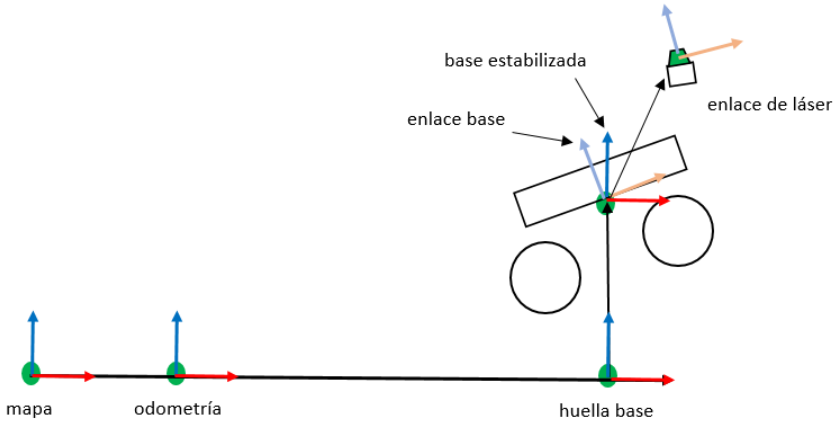

Ilustración 32 Funcionamiento del láser en un robot

Fuente: propia con datos recuperados de *ROS* (2020)

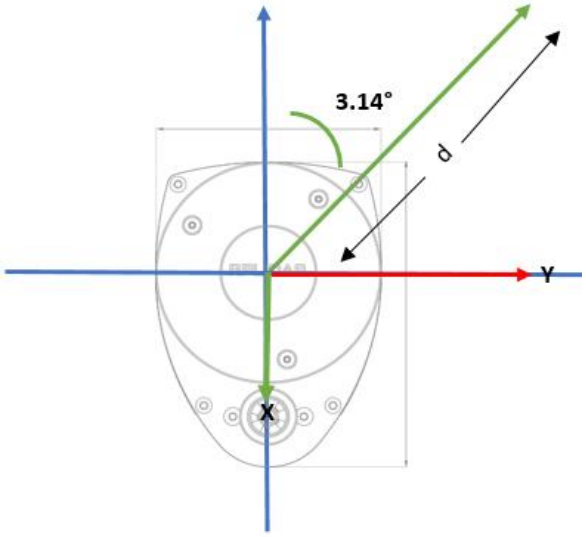

Ilustración 33 Ángulo interno del sensor LIDAR 2D

Fuente: *ROS* (2020)

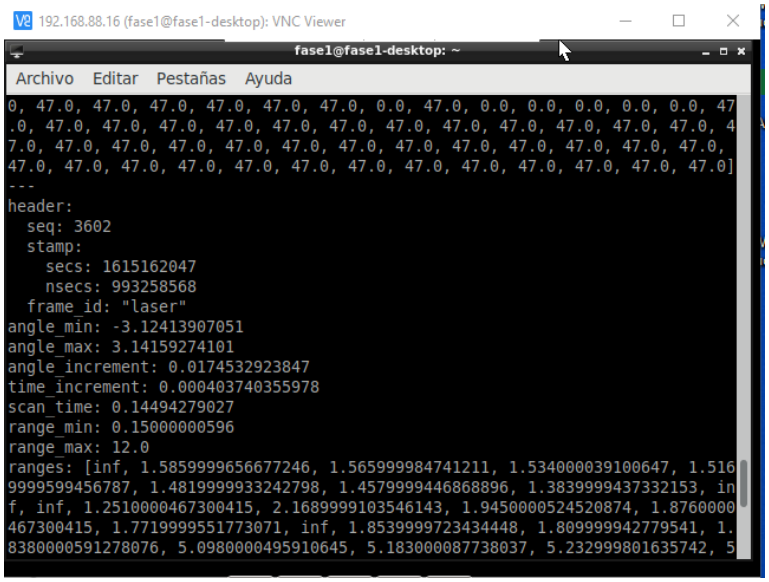

Ilustración 34 Configuración Interna de escaneo del sensor LIDAR 2D

Fuente: propia.

En la ilustración 34 se logra apreciar los ángulos mínimos y máximos que puede soportar el sensor también los incrementos que este puede tener en el tiempo establecido, también se observa los otros valores que pertenecen a los datos leídos en ese momento.

5.3 SISTEMA DE CONTROL

En cuanto al sistema de control se implementó el jetson nano apoyado de arduinos nanos, su funcionamiento dependerá de la retroalimentación e información proporcionada por los sensores y actuadores del robot. Todo ello está conectado de forma que el jetson nano sea el máster del prototipo, ya que éste se encarga de tomar decisiones e interpretar las señales enviadas desde los arduinos los cuales cuantifican la información proporcionada por los encoders para obtener la localización.

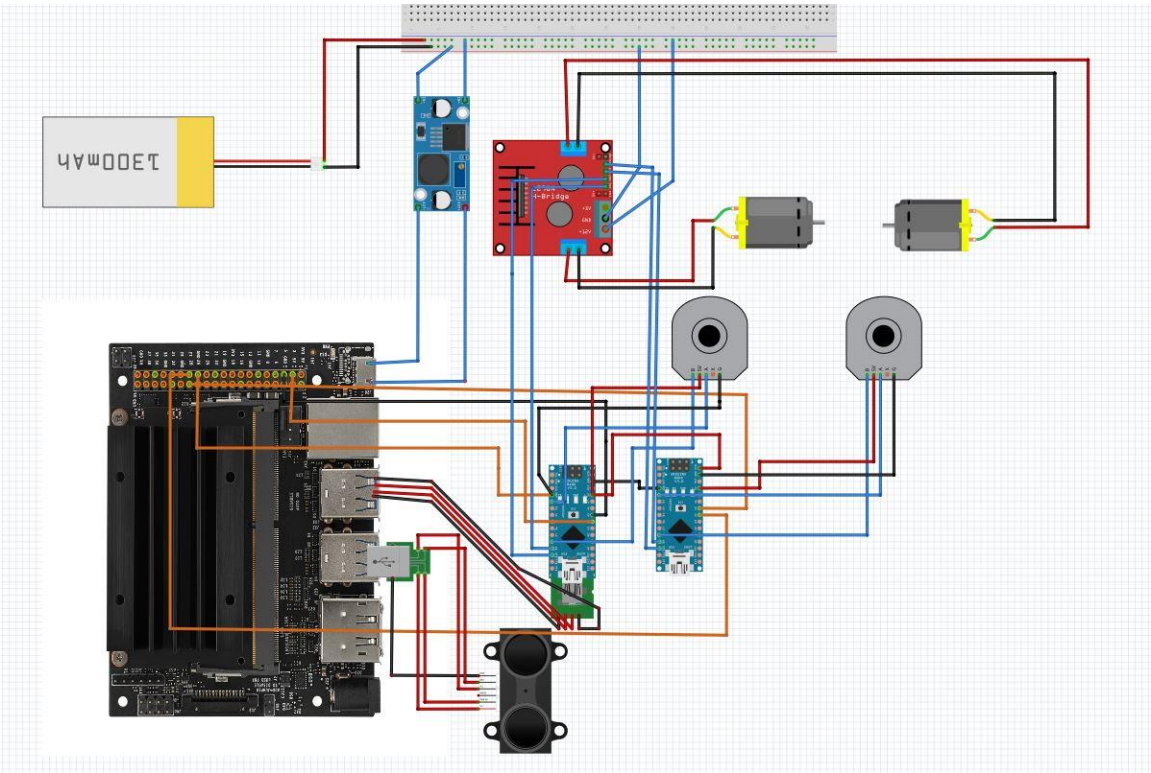

Ilustración 35 Diagrama de conexión

Fuente: propia.

La ilustración 35 muestra el diagrama de conexiones del prototipo, como se puede observar el prototipo cuenta con un jetson nano 2gb, dos arduinos nano, dos codificadores rotativos, dos motores dc 12v, un puente H, un regulador descendente de voltaje y el sensor *LIDAR 2D*. En cuanto a las conexiones, los arduinos nanos se alimentan con *USB,* pero envían la información proporcionada por los codificadores mediante los puertos GIO del jetson nano utilizando los puertos con el protocolo de comunicación *I2C*. En cuanto al sensor *LIDAR* se alimenta y comunica mediante *USB*.

Para el control de los codificadores se utiliza un arduino nano para cada uno programado con lógica de un PID para obtener la información de forma precisa, a la hora de realizar las conexiones al puente H se encontró el problema que los motores no se activaban a pesar de que recibían la señal de parte del jetson nano, dicho problema se derivaba de la librería de PID programada en los arduinos ya que inicialmente el circuito no contaba con un GND independiente para ese circuito y al estar el circuito abierto la lectura de PID y de los arduinos era errónea, por ende no permitían al puente H hacer el paso de corriente a los motores para su activación, la solución fue colocar a tierra de forma independiente el circuito

del puente H, dicho circuito fue aterrizado apoyándose de un puerto GND del jetson nano. En cuanto a los motores DC utilizados, mediante la librería de PID pasaron a operar a 1,500 rpm ya que naturalmente son de 3,000 rpm, pero se detectó que al moverse tan rápido el sensor *LIDAR* no podía mapear correctamente y se tenían que realizar muchas vueltas en el mismo mapa, con esta velocidad moderada es posible para el sensor mapear en una única ruta.

5.4 SISTEMA DE ALIMENTACIÓN

De forma teórica se realizaron los cálculos de la autonomía de la fuente de alimentación presente en el robot. La fuente de alimentación es una batería de 12v tipo lipo de 2,200mah.

5.4.1 Tiempo de carga

El tiempo de carga de la batería de 2,200 mah con el cargador proporcionado que carga las celdas de la batería a 1 amp, da un resultado que se carga la batería en 2 horas y 12 minutos.

5.4.2 Autonomía

En cuanto a la autonomía del robot se obtiene mediante la ecuación 21, el consumo total del robot es de 1,233 mah considerando el consumo de cada componente y realizando la sumatoria. Al tener estos datos la autonomía de la batería nos da un resultado de 1 hora con 47 minutos. Esto puede variar dependiendo el uso del robot.

5.5 FRAMEWORK IMPLEMENTADO Y FILTRADO

Para el desarrollo del prototipo el *framework* es una parte esencial puesto que éste se encarga de interpretar todas las señales y ejecutar los nodos encargados de la lógica del prototipo al igual que el tipo de filtrado de las señales para obtener la señal más pura.

5.5.1 Framework ROS

El *framework* implementado en el prototipo es *ROS,* debido a la versión de sistema operativo que posee el jetson nano, se implementó *Melodic ROS* basado en *Ubuntu 18.02*. Para el funcionamiento del prototipo se deben definir los nodos, tópicos, mensajes enviados e interpretados en el espacio de trabajo de *ROS* y compatibilidades con los distintos actuadores y sensores que conforman el sistema. Con el apoyo de librerías como "ROS_differential_drive_controller", "gmapping", "robot_hardware_interface", "rplidar_ros", "i2c_ros" y "mobile_robot_urdf" se logró la comunicación entre los componentes y la

implementación de una lógica apoyada de *machine learning* para que el robot logre operar de manera autónoma.

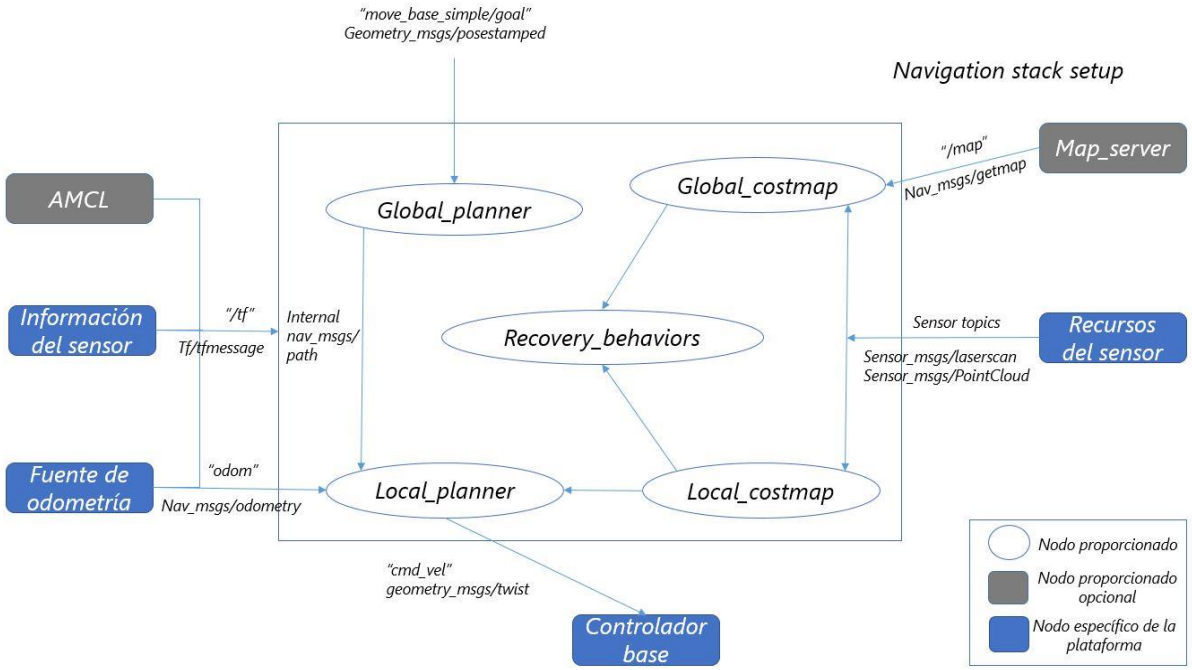

Ilustración 36 Navigation stack setup

Fuente: propia con datos recuperados de *ROS* (2020)

La ilustración 36 muestra la configuración de los nodos, tópicos y mensajes enviados para poder operar el "navigation stack". Los nodos azules son los específicos implementados en el prototipo en el que se incluye: controlador base, recursos del sensor, información del sensor y la fuente de odometría. El controlador base es el encargado del control de los motores mediante la librería PID implementada en los arduinos nanos, dicho nodo depende de la ejecución de los nodos "*global_planner*" y "*local_planner*", cada uno de ellos se retroalimenta de diferentes tópicos, mensajes y comandos que le permiten la implementación de la lógica. Los recursos del sensor se encargan de enviar los tópicos específicos del *LIDAR* mediante los mensajes del escaneo láser y los puntos específicos enviados, con ello informan a los nodos encargados de los mapas, locales y globales que posteriormente son almacenados en el nodo "*map_server*". Información del sensor es la información que envía el sensor para poder navegar, básicamente es el encargado de detectar todos los obstáculos a la hora de querer navegar de forma autónoma. La fuente de odometría es el encargado de proporcionar la ubicación del

robot, en cuanto el prototipo se mueva los codificadores conocen cuánta distancia se ha recorrido y mediante cálculos envía los tópicos y mensajes de odometría que dictan la posición en tiempo real del robot. *AMCL* es el nodo que engloba la odometría y el mapeo para poder realizar *SLAM*. Todos estos nodos, mensajes y tópicos están ordenados e implementados en una carpeta llamada "robot_autonomous_navigation", dicha carpeta cuenta con un archivo tipo "*launch"* que es el llamado a la hora de querer ejecutar la navegación autónoma, al ser ejecutado por *ROS* se ejecuta el *software Rviz,* que como se mencionó anteriormente es el software que se utilizará para la visualización en tiempo real del robot en el mapa creado y así mismo proporciona las herramientas de "2D *navigation" "2D pose estimate" y "measure"*.

La programación del robot ha sido realizada totalmente apoyándose de las librerías antes mencionadas, dichas librerías fueron instaladas en el espacio de trabajo de *ROS*, al estar todas dentro del espacio de trabajo *ROS* solo fue necesario ajustar parámetros en la programación de navegación autónoma, entre ellos la distancia entre ejes y el radio de lo engranes. Un obstáculo que se encontró a lo largo de implementar *ROS* fue la creación de un solo espacio de trabajo ya que ocurrieron muchos errores a la hora de crear debido a que el espacio de trabajo no había sido creado de forma correcta por errores en el sistema operativo, para resolver dicho problema se descargó el SO más estable y mediante una *SD* de alta velocidad de lectura totalmente limpia. Otro problema que surgió fue que las direcciones ip del jetson nano cambiaban cada vez que se hacía un reinicio del minicomputador y se perdían las conexiones de los nodos del espacio de trabajo, dicho problema se resolvió asignando una dirección ip estática.

La ejecución de los nodos se realizó utilizando diferentes terminales ya que dentro de una sola terminal solo se realizaba un "*launch"*. Para el funcionamiento del robot se requiere de realizar en primer lugar la navegación tele dirigida con el nodo de "diff_drive_ct" y "teleop_twist_keyboard" para controlarlo con el teclado de la computadora, al ser ejecutados se habilita el control tele operado y comienza a realizar el mapa, al obtener el mapa con la mayor cantidad de información se procede a guardarse en el nodo "map_server", al obtener el mapa ya se habilita la navegación autónoma y para ello se ejecuta el *launch* de "robot_autonomous_navigation" y con ello se ejecuta el mapa realizado con las herramientas de navegación autónoma ya habilitadas para su funcionamiento.

5.5.2 Diagramas de funcionamiento

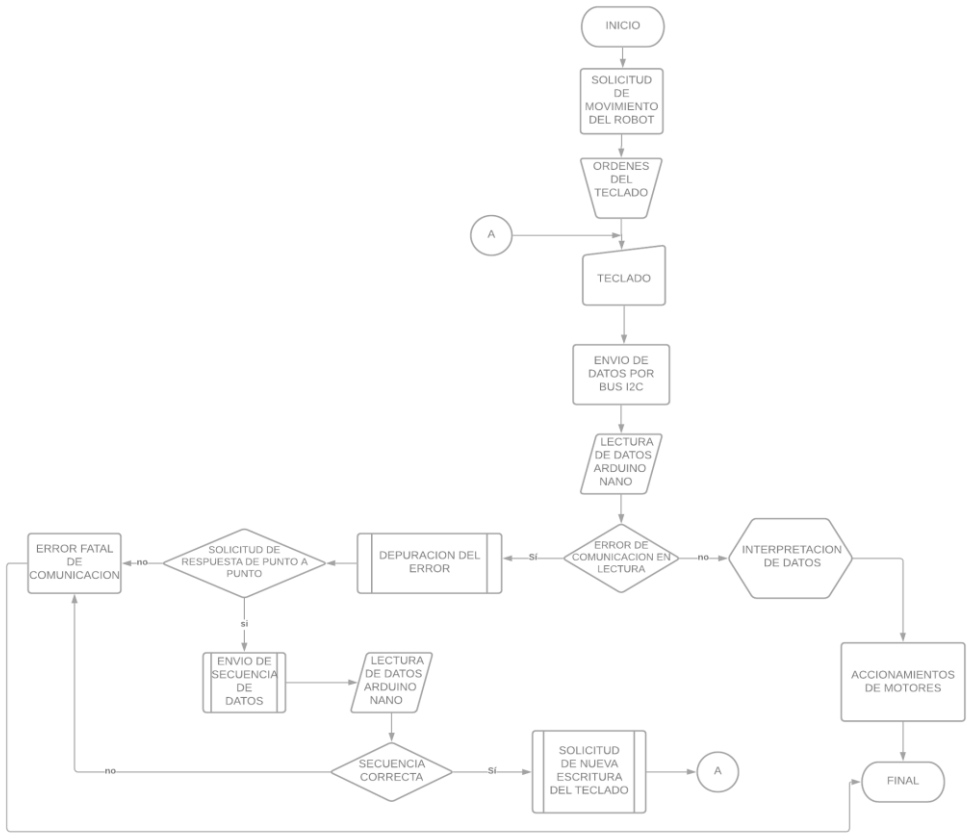

Ilustración 37 Diagrama de movimiento y comunicación

Fuente: propia.

En la ilustración 37 se observa el diagrama de movimiento y comunicación, el cual funciona de la siguiente manera: inicia con la solicitud del movimiento del robot, dicha acción es enviada desde el teclado de la pc remota donde estamos controlando el robot, el jetson comprende las órdenes del teclado, con ello envía los datos obtenidos por el bus I2C del jetson nano, se realiza la lectura de los datos enviados por los arduinos, posteriormente se comprueba si hay error en la comunicación, si lo hay se depura el error y se envía una solicitud de respuesta punto a punto, si se logra la comunicación envía la secuencia de datos, la lectura de los arduinos y se corrobora que llegue a la secuencia correcta, al asegurarse que es la secuencia correcta se realiza una solicitud de nueva escritura del teclado y comienza nuevamente el ciclo, de lo contrario se envía un aviso de error fatal de comunicación y finaliza el programa. En caso de que no exista un error de comunicación en lectura, se procede a la interpretación de datos que posteriormente lleva al accionamiento de los motores para lograr el movimiento.

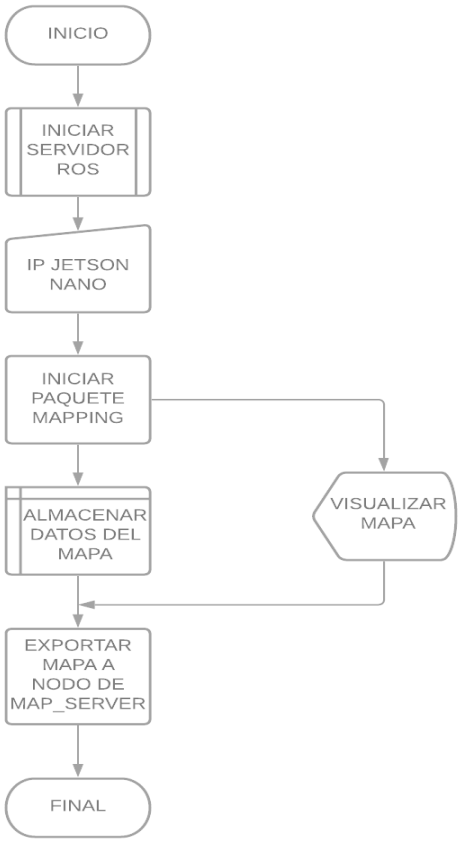

Ilustración 38 Diagrama de inicio de servidor y nodos

Fuente: propia.

El inicio del servidor ROS y nodos se inicia con la puesta en marcha del servidor ROS, posterior a ello se configura la ip del jetson nano para que comunique con la red local, una vez configurada la ip se inicia el paquete de mapping, simultáneamente se puede observar el mapa que se está creando en tiempo real y así mismo se van almacenando los datos del mapa, una vez terminado el mapping se exporta el mapa en el nodo map_server.

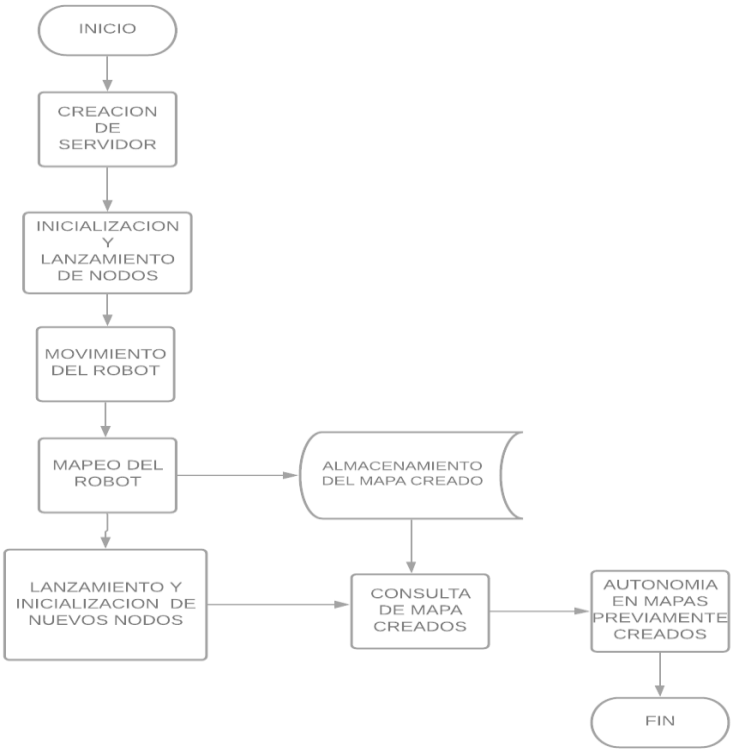

Ilustración 39 Diagrama de funcionamiento general

Fuente: propia.

El funcionamiento general del robot, se inicia el robot y con ello la creación del servidor, al estar correctamente iniciado se comienza con los lanzamientos de nodos, entre ellos se inicia el teleopt_keyboard para poder enviarle instrucciones de movimiento al robot utilizando el teclado, ya en movimiento simultáneamente el robot mapea el ambiente en el que se encuentra, una vez teniendo el mapeo se procede a almacenarlo en el map_server, con ello se permite el lanzamiento de nuevos nodos y así mismo la consulta de mapas creados para habilitar la navegación autónoma en caso de existir mapas creados.

5.5.3 Filtro de Kalman

En cuanto al filtrado de las señales, *ROS* implementa el algoritmo matemático de *Kalman* para poder generar la señal más pura y deseable de parte del sensor *LIDAR*. Para comprender dicho algoritmo se ha realizado una simulación en *Matlab* para poder

observar el filtrado en una señal con ruido aleatorio utilizando el siguiente código:

```
close all
clear all
clc
wn = 10;
Hs = tf ([1 2], [1 2 wn^2])
h = 0.1/sqrt(wn);
Hz = c2d (Hs, h)
n = [0.03111 -0.02918];
d = [1 -1.843 0.9387];
[A, B, C, D] = tf2ss (n, d);
uk =1;
Xem1= [0;0];
Pkm1=1e6;
Q =0.5;
R = 0.5;
Y =step (Hs, 10);
Yest=zeros (1,200);
for k=1:200
   Ys(k)=Y(k)+0.1*(0.2-rand);
   Xem=A*Xem1+B*uk;
   Pkm=A*Pkm1*A'+Q;
   Kk=(Pkm*C')/(C*Pkm*C'+R);
   Xe=Xem+Kk*(Ys(k)*C*Xem);
   Pk= (eye (2) *Kk*C) *Pkm;
   Xem1=Xem;
   Pkm1=Pk;
   Yest(k)=C*Xe;
end
figure (1)
subplot (3,1,1)
plot(Y)
title ('Señal Original')
axis ([0 200 -0.15 0.15])
subplot (3,1,2)
plot(Ys,'g')
title ('Señal con ruido Aleatorio')
axis ([0 200 -0.15 0.15])
subplot (3,1,3)
plot(Yest,'r')
title ('Señal Recuperada')
axis ([0 200 -0.15 0.15])
figure (2)
hold on; grid on
plot(Y)
plot(Ys,'g')
plot(Yest,'r')
axis ([0 200 -0.15 0.15])
```

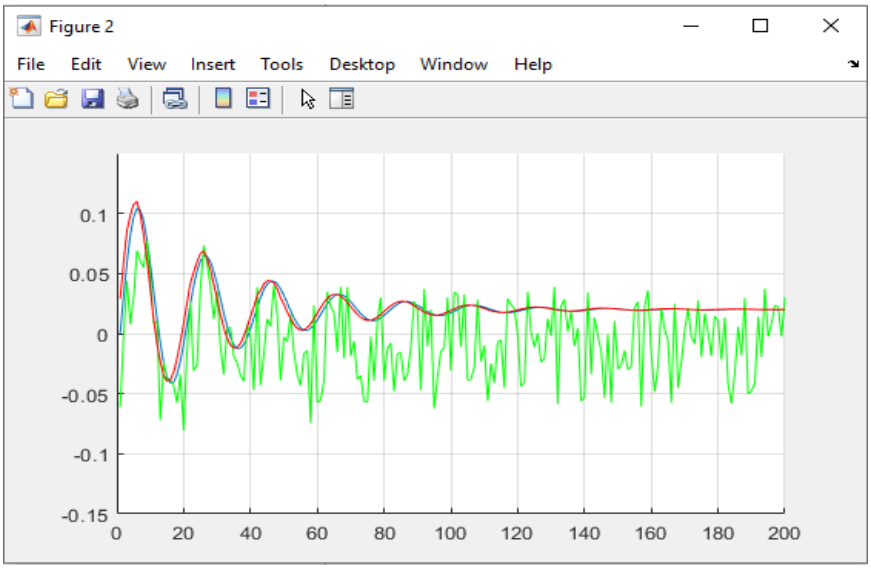

Ilustración 40 Filtro de Kalman simulado en Matlab

Fuente: propia.

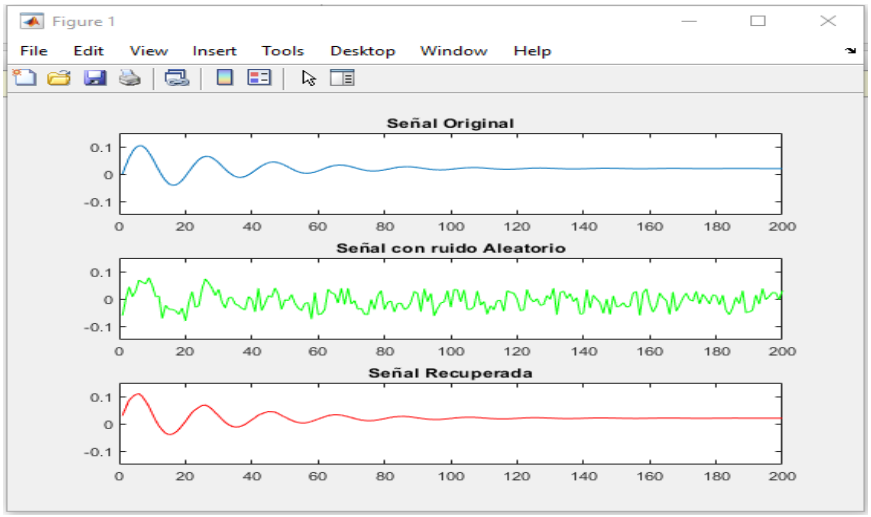

Ilustración 41 Comparación de señal filtrada con filtro de Kalman con presencia de ruido

Fuente: propia.

Como se puede observar en la ilustración 40 y 41 la señal recuperada mediante un filtro de Kalman es prácticamente idéntica a la original, sin dicho filtrado el ruido hubiese generado distorsión en la señal original al punto de dejarla incomprensible o entregando un mensaje muy diferente al enviado. Gracias a éste algoritmo los sistemas *SLAM* generan mapas muy acertados a la realidad.

5.6 DISEÑO FINAL DEL PROTOTIPO Y MAPAS REALIZADOS POR *SLAM*

Una vez realizadas las diferentes pruebas a los sistemas que conforman al robot, se procedió a realizar el ensamblaje completo del robot, en dicho proceso se detectó un problema que a la larga puede afectar el funcionamiento del robot y son los codificadores, éstos se ensamblan en el eje del engrane del motor pero por el lado de la banda y al realizarse muchas pruebas estos codificadores movieron la banda debido a tanta vibración dejando al robot sin opción de movimiento, dicho problema se resolvió de forma temporal dejando los codificadores pegados a la parte del eje para evitar que la vibración los sacara del sitio donde operan. En cuanto al sistema de control es necesario aislarlos para evitar cortocircuitos y para ello se apoyó de cinta adhesiva doble cara y fajillas plásticas resistentes para apoyar en la sujeción, el sensor *LIDAR* cuenta con roscas para el ensamblaje con tornillos a la estructura del robot. En cuanto al sistema de alimentación se colocó todo de forma ordenada y no se encontraron problemas de sobrecalentamiento ni vibraciones indebidas en esa parte del robot. El prototipo final cuenta con una altura máxima de 18.8 cm, ancho de 25.2 cm incluyendo los encoders ensamblados en la parte lateral de la banda y una longitud de 32 cm. Los mapas realizados van acorde al entorno real en el que opera el robot, utilizando la frecuencia estándar dictada por el fabricante de 6 Hz.

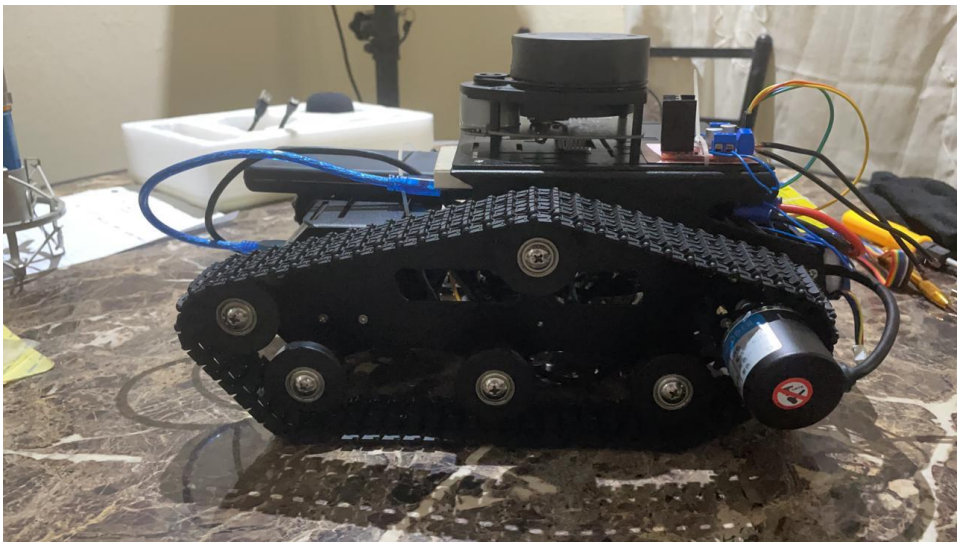

Ilustración 42 Diseño final del prototipo con todos los componentes ensamblados

Fuente: propia.

En la ilustración 42 se puede observar el robot *SLAM* ensamblado, en la parte de la banda el engrane en el cual van acoplados los motores se colocaron los codificadores

incrementales que son los encargados de localizar al robot, el sensor *LIDAR 2D* se encuentra en la parte más alta de la estructura, debido a sus limitaciones de detectar superficies y objetos en un solo plano, se utilizó la estructura tipo tanque para que pequeños objetos no sean impedimento para que se pueda movilizar con fluidez. Una vez ensamblado el robot se procedió a operarlo de manera remota para la creación de mapas, todo ello haciendo uso de *ROS* y ejecutando el programa *RVIZ* para monitorear en tiempo real cuando éste se moviliza y lo que está detectando.

Ilustración 43 Robot mapeando un entorno controlado

Fuente: propia.

Como se puede observar en la ilustración 43 el robot se encuentra en un entorno controlado para poder corroborar que la calidad del mapeo es la adecuada y a su vez comprobar que dicho mapeo es acertado a la realidad. Se colocaron los obstáculos a modo que no sea tan fácil para el robot mapear. Una vez realizado dicho entorno se procedió a la ejecución de la operación tele operada y mapeo.

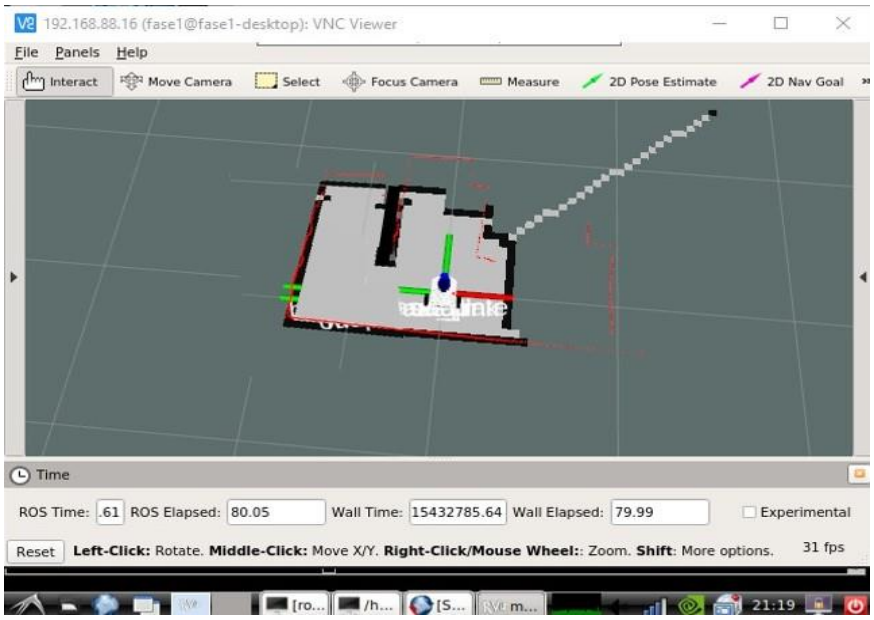

Ilustración 44 Mapa realizado en el entorno controlado

Fuente: propia.

La ilustración 44 demuestra el mapeo final realizado por el robot, dicho mapeo es muy acorde a la realidad, detecta en las esquinas hasta los subniveles debido a que se colocaron las cajas a modo de que no fuera tan plano dicho entorno y, por ende, no sea tan fácil de detectar, sin embargo, el mapeo del robot es preciso y ha logrado detectar las diferentes irregularidades en las esquinas del entorno en el que se encuentra. En cuanto a la navegación autónoma se habilita al ejecutar el nodo del mapa y el de "robot_autonomous_navigation", cuando el robot ya conoce el entorno en el que opera se puede enviar a cualquier punto del mapa y de manera autónoma él se mueve y dibuja la trayectoria en el instante. Así mismo se habilitan diferentes herramientas que funciona para cambiar parámetros que ayuden a modificar la visualización del mapa.

COMPONENTE	CANTIDAD	PRECIO POR UNIDAD	PRECIO TOTAL
ESTRUCTURA	1	$90	$90
JETSON NANO 2GB	1	$59	$59
ARDUINO NANO	2	$8	$16
MÓDULO L298N (PUENTE H)	1	$8	$8
CODIFICADOR ROTATIVO INCREMENTAL HN3806	2	$24	$48
SENSOR *LIDAR* A1M8 2D	1	$100	$100
BATERÍA LIPO 12V	1	$30	$30
MOTORES DC 12V	2	$10	$20
OTROS	1	$40	$40
			TOTAL: $411

Tabla 7 Costo del prototipo final.

Fuente: propia.

VI. CONCLUSIONES

Finalizada la investigación se puede concluir lo siguiente:

- Se rechaza la hipótesis nula.
- Se identificaron las limitaciones y formas de operación del sensor *LIDAR 2D* al ser implementado en un sistema *SLAM*, así mismo, las ventajas que dicho sensor posee.
- Se enumeraron e implementaron las características para el desarrollo de un prototipo considerado robot autónomo, conociendo las diferencias entre un inteligente y tele operado, se concluye que el prototipo realizado es un robot inteligente y tele operado.
- Se seleccionaron los codificadores rotativos para apoyar al robot en el tema de la localización en tiempo real debido a que éstos dispositivos mediante el movimiento del eje cuantifican la cantidad de pulsos para posteriormente transformarlo en una señal medible que retroalimenta al sistema con la posición exacta del robot.
- Se implementó *ROS* como *framework* debido a su flexibilidad, capacidad de comunicación y obtención de datos proporcionados por cada componente que conforma el robot autónomo *SLAM*.

VII. RECOMENDACIONES

Con los resultados obtenidos y finalizada la investigación se plantean las siguientes recomendaciones para investigaciones futuras relacionadas con la presente:

- Utilizar el sensor *LIDAR 2D* para aplicaciones sencillas y de bajo costo.
- Se recomienda utilizar un minicomputador más potente y con un SO actualizado para la implementación de *ROS2* y ejecución de más nodos simultáneamente.
- Para aplicaciones más complejas, como por ejemplo robots de rescate, se recomienda utilizar un sensor *LIDAR 3D* debido a que éste obtiene la información de los 3 planos de forma simultánea.
- Se recomienda utilizar motores con codificadores rotativos ya incorporados en la misma estructura para poder ahorra espacio y evitar los problemas en la banda antes expuestos.

VIII. REFERENCIAS

Joseph, L., & Cacace, J. (2018). Mastering ROS for Robotics Programming - Second Edition: Design, build, and simulate complex robots using the Robot Operating System . *Packt Publishing*.

L. E. Solaque, M. A. Molina, E. L. Rodríguez. (2014). Seguimiento de trayectorias con un robot móvil de configuración diferencial. *Ing. USBMed, Vol. 5*, 26-34.

Algar Diaz, M. F. (2019). *Introduccion practica a la programación con Python.* Universidad de Alcala.

Arkin, R. C., & Murphy, R. R. (1990). Autonomous navigation in a manufacturing enviromment. *IEEE transactions on Robotics and Automartion*, 445-454.

Ashish, R., Sujay, B. M., Caushik, R., Jayashree, S., & Nischal, H. P. (2018). 2D Based Indoor Slam and Autonomous Navigation Using a Terrain Robot. *2018 3rd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*.

Barrientos, A. (2012). *Fundamentos de robótica.* Madrid, España: McGraw-Hill.

Baturone, A. O. (2001). *Robótica: manipuladores y robots móviles.* Barcelona, España: GyERSA.

Cadena, C., Carlone, L., Carrillo, H., Latif, Y., Scaramuzza, D., Neira, J., … Leonard, J. J. (2016). Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age. *Transactions on Robotics*, 1309-1332.

Dogan, I. (2008). *Programacion de microcontroladores PIC.* . Barcelona: Marcombo.

Dorsch, R. G., Häusler, G., & Herrmann, J. M. (1994). Laser triangulation: fundamental uncertainty in distance measurement. *Applied Optics*, 33-40.

IFR (International Federation of Robotics). (27 de Enero de 2021). *Internacional Federation of Robotics*. Obtenido de https://ifr.org/ifr-press-releases/news/robot-race-the-worlds-top-10-automated-countries

Kumari, R. S. (2017). Interfacing of MEMS motion sensor with FPGA using I2C protocol. *IEEE*, 2.

L. E. Solaque, M. A. Molina, E. L. Rodríguez. (2014). Seguimiento de trayectorias con un robot móvil de configuración diferencial. *Ing. USBMed, Vol. 5, No. 1*, 26-34.

Latif, Y., Cadena, C., & Neira, J. (2012). Realizing, reversing, recovering: Incremental robust loop closing over time using the iRRR algorithm. *IEEE/RSJ International Conference on Intelligent Robots and Systems*.

Manikas, T., Ashenayi, K., & Wainwright, R. (2007). Genetic algorithms for autonomous robot navigation. *IEEE Instrumentation & Measurement Magazine*, 26-31.

Marquez, F. (2015). *Unix: programacion avanzada.* RA-MA.

Muñoz, E. (2016). *Aprendiedno a programar paso a paso con C.* Madrid: Budok Publishing.

Neira, J., Davison, A. J., & Leonard, J. J. (2008). Guest Editorial Special Issue on Visual SLAM. *IEEE Transactions on Robotics,* , 929-931.

Ortiz Bravo, V., Nieto Arias, M., & Castañeda Cardenas, J. (2013). ANALISIS Y APLICACIÓN DEL FILTRO DE KALMAN A UNA SEÑAL CON RUIDO ALEATORIO. *Scientia Et Technica,*, 267-274.

Palomares, C. (2017). *Instalacion y configuracion del software de servidor Web.* Madrid: CEP.

Ren R., Fu H., & Wu M.,. (2020). Large-Scale Outdoor SLAM Based on 2D Lidar. *TELKOMNIKA Telecommunications, Computing, Electronics and Control Vol. 18*.

Rivai, M., Hutabarat, D., & Jauhar Nafis, Z. M. . (2020). 2D mapping using omni-directional mobile robot equipped with LiDAR. *ELKOMNIKA Telecommunication, Computing, Electronics and Control*, 1467-1470.

Rodriguez, P. (2015). *Linux Avanzado.* ICB.

Saha, K. (2011). *Introducción a la robótica.* Ciudad de Mexico: McGraw-Hill.

VALENCIA V., JHONNY A., & MONTOYA O., ALEJANDRO, & RIOS, LUIS HERNANDO. (2009). ODELO CINEMÁTICO DE UN ROBOT MÓVIL TIPO DIFERENCIAL Y NAVEGACIÓN A PARTIR DE LA ESTIMACIÓN ODOMÉTRICA. *Scientia Et Technica, XV*, 191-196.

Vasilije S. Vasić, M. P. (2008). Standard Industrial Guideline for Mechatronic Product Design. *FME transactions*, 6.