UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Diseño e implementación de un sistema de control y navegación automático para el robot secador de café

Trabajo de graduación presentado por Alvaro Javier Torres González
para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,

2022

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

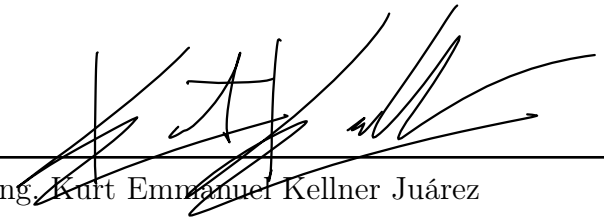
Diseño e implementación de un sistema de control y navegación automático para el robot secador de café

Trabajo de graduación presentado por Alvaro Javier Torres González
para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,

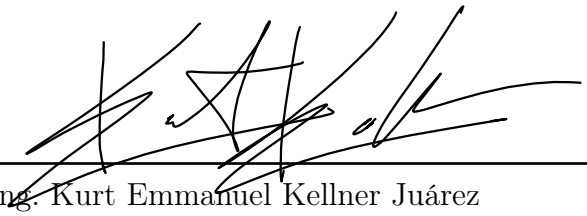
2022

Vo.Bo.:

(f) 

Ing. Kurt Emmanuel Kellner Juárez

Tribunal Examinador:

(f) 

Ing. Kurt Emmanuel Kellner Juárez

(f) 

Ing. Pedro Iván Castillo Rivera

(f) 

Ing. Miriam Carolina Solares López

Fecha de aprobación: Guatemala, miércoles 5 de Enero de 2022.

Prefacio

Este proyecto de graduación surgió a partir de la necesidad de automatizar un proceso agrícola, en este caso el secado de café. La finca El Injerto, la cual se ubica en el departamento de Huehuetenango - Guatemala, le compartió a la Universidad del Valle de Guatemala el deseo de implementar tecnología y automatizar su proceso de secado de café, sin perder la técnica que los ha caracterizado durante muchos años.

Debido a esto, el departamento de Ing. Electrónica, Mecatrónica y Biomédica de la Universidad del Valle de Guatemala vio la oportunidad para apoyar a la finca El injerto diseñando y construyendo un robot móvil con un sistema de control que cumple con las necesidades de la finca.

Este es el segundo trabajo enfocado en el sistema de control de este proyecto. Bajo la tutela del Ing. Kurt Emmanuel Kellner, con ayuda del departamento de Ing. Electrónica, Mecatrónica y Biomédica de la UVG y con el apoyo de la finca El Injerto, se presenta el tercer prototipo del robot secador de café.

Sin más qué agregar, se agradece el valioso aporte del Ing. Kurt Emmanuel Kellner, del departamento de Ing. Electrónica, Mecatrónica y Biomédica de la Universidad del Valle de Guatemala y de la finca El Injerto para que este proyecto se realizara con éxito.

También quiero hacer un reconocimiento especial a mi familia y amigos que me apoyaron anímicamente en todo este proceso y me ayudaron a seguir adelante. Lucrecia Carolina González, Marco Augusto Torres, José Gabriel Torres y David Ernesto Torres, este trabajo es por ustedes.

Índice

Lista de figuras

Lista de cuadros

Resumen

En las últimas décadas hemos tenido grandes avances tecnológicos alrededor de todo el mundo en todas las áreas. Estos avances permitieron una evolución significativa y cambios en los métodos de realizar las cosas. En el área de la agricultura, específicamente en la cafetalera, también podemos encontrar el uso de la tecnología y robótica.

En este trabajo se abarcan tema cómo el secado de café y la implementación de un robot para la realización de estas tareas, también se habla del sistema de control y el uso de sensores para la recolección de datos y el uso de estos para el manejo "autónomo" del robot. Como resultado final del robot secador de café se logró que este sea capaz de circular de manera remota controlado por un operario por medio de un control remoto con comunicación de radio frecuencia. También se consiguió la lectura de posición y distancia a partir de un sistema de localización en tiempo real (RTLS) utilizando módulos DWM1001.

Para un robot móvil autónomo, conocer la posición/ubicación en tiempo real es esencial para poder tomar mediciones y con ellas navegar en el espacio indicado, por lo que el uso de sensores cómo la unidad de medición inercial (IMU) y los DWM1001 que conforman al sistema de localización en tiempo real (RTLS) son los componentes ideales y los utilizados en el robot secador de café.

Al trabajar con motores con grandes torques se pierde la regularidad de giro o avance debido a diferentes factores como el deslizamiento de las ruedas o la irregularidad del terreno. Por lo que fue necesario la implementación de un sistema de control para conseguir un movimiento constante y recto por parte del robot. Para esta tarea se implementó el uso de encoders ópticos en el robot.

En el siguiente trabajo también se habla del impacto y la eficiencia que se obtiene al implementar robots para realizar tareas de secado y del impacto que tiene el café en un país dónde una de sus principales fuentes de ingreso es la exportación de productos agrícolas y textiles. El sistema de control presentado en este trabajo pertenece al tercer prototipo del Robot secador de café desarrollado por estudiantes de la Universidad del Valle de Guatemala, por lo que en este trabajo se tocarán temas relevantes, sin embargo, pueden complementarse algunos otros temas con la información brindada en trabajos previos a este [1].

Abstract

In the last decades we have had great technological advances around the world in all areas. These advances allowed for significant evolution and changes in the methods of doing things. In the area of agriculture, specifically in the coffee plantation, we can also find the use of technology and robotics.

This covers the topic of how the coffee drying work and the implementation of a robot to carry out these tasks, it also talks about the control system and the use of sensors for data collection and the use of these for handling .ªutonomous "of the robot. As a final result of the coffee drying robot, it was achieved that it is able to circulate remotely controlled by an operator by means of a remote control with radio frequency communication. Position and distance reading was also achieved starting from a real-time location system (RTLS) using DWM1001 modules.

For an autonomous mobile robot, knowing the position / location in real time is essential to be able to take measurements and with them navigate in the indicated space, so the use of sensors such as the inertial measurement unit (IMU) and the DWM1001 that make up to the Real Time Location System (RTLS) are the ideal components and those used in the robot coffee dryer.

When working with motors with high torques, the regularity of rotation or advance is lost due to different factors such as the slip of the wheels or the unevenness of the ground. Therefore, it was necessary to implement a control system to achieve a constant and straight movement by the robot. For this task, the use of optical encoders was implemented in the robot.

The following work also talks about the impact and efficiency obtained by implementing robots to carry out drying tasks and the impact that coffee has in a country where one of its main sources of income is the export of agricultural and textile products. The control system presented in this work belongs to the third prototype of the Coffee Dryer Robot developed by students of the Universidad del Valle de Guatemala, so relevant topics will be addressed in this work, however, some other topics can be complemented with the information provided in previous works to this [1].

CAPÍTULO 1

Introducción

En este trabajo de graduación se presenta el proceso de diseño, implementación, pruebas y resultados de fabricar el sistema de control autónomo de un robot móvil para cumplir con una tarea específica, en nuestro caso, el secado de café.

El sistema de control cuenta con un juego de sensores y actuadores los cuales permiten al microcontrolador ATMega2560 recopilar la información del sistema y el entorno y con ella poder actuar conforme a nuestros objetivos. Lo que se busca es recorrer de forma autónoma un área cubierta con café siguiendo un patrón de ida y vuelta a lo ancho del campo. En la finca El Injerto se utiliza esta técnica actualmente, con la diferencia que es una persona la encargada de realizar este recorrido.

El robot utiliza dos motores DC de 2.9 N.m de torque alimentados por una batería de carro, estos son controlados por el microcontrolador ATMega2560. Los sensores que se utilizaron para la retroalimentación del sistema son cuatro DWM1001-DEV, uno de ellos anclado al robot (*TAG*) y los otros tres en las esquinas del campo de trabajo (*Anchors*), estos conforman el sistema de localización en tiempo real o por sus siglas en inglés *RTLS*. También cuenta con dos encoders para obtener la odometría del robot, y por último un LIS3MDL y un LSM6DS33 que en conjunto conforman una unidad de medición inercial (*IMU*) la cual nos brinda los ángulos ROLL-PITCH-YAW de nuestro robot.

También se solicitó por parte de la finca que un operario fuera capaz de controlar al robot manualmente, por lo que se implementó la opción de control remoto en la cual el operario toma el completo control del robot.

Por último, se implementó un mecanismo que eleva las aspas del robot las cuales se encargan de mover el café. Este mecanismo se diseñó para evitar manipular el café cuando el robot se esté desplazando por fuera del terreno cubierto de café.

En este trabajo podrán encontrar, modelos a escala, diseños, resultados de las pruebas individuales de los módulos, códigos y cálculos realizados durante el trabajo.

CAPÍTULO 2

Antecedentes

Con el paso de los años se han desarrollado nuevas tecnologías enfocadas en diversas áreas tales cómo agricultura, ganadería, construcciones, diseño, automatización, transporte, entre otras. Esto con el fin de mejorar, facilitar y eficientizar los trabajos del día a día o incluso dar un mayor confort.

De manera más específica, podemos mencionar los grandes avances que se han desarrollado en el área de la robótica en las últimas décadas. De pasar de un simple robot que podía ejecutar un repertorio de tareas muy limitado, a tener robots capaces de entablar conversaciones, reconocer emociones, caminar, hacer piruetas, manejar de manera autónoma, entre otras tareas mucho más complejas.

Los robots sin duda alguna han llegado al mundo para cambiarlo y en el área de la industria no es una excepción. Hoy en día muchas industrias apuestan fuertemente por la automatización de procesos debido a las grandes ventajas que conlleva que un robot o una máquina realice la tarea necesaria. Entre las mayores ventajas que se pueden encontrar es el no arriesgar vidas humanas en procesos peligrosos cómo manipulación de cierras eléctricas o trabajos con elementos tóxicos. Entre otras ventajas podemos mencionar la constancia con la que se realiza la misma tarea, la rapidez, la precisión, el tiempo de trabajo, entre otras. Todo esto se a conseguido gracias a que cada vez hay nuevas y mejores maneras de que una máquina o un robot interactúe con el mundo real por medio de sensores, actuadores y cámaras que les ayudan a recopilar datos y, de manera muy eficiente, poder manipularlos para ejecutar tareas de la mejor y más precisa manera [2].

Los robots o vehículos autónomos son una buena muestra de esto. Para dar un ejemplo podemos observar a los vehículos TESLA y el gran impacto que han tenido en la industria automotriz. Gran parte de su éxito se debe a su sistema de manejo “autónomo” el cual resuelve una de las mayores problemáticas en el transporte terrestre, la inseguridad en las carreteras. Esto lo logra debido a sensores, cámaras, sistemas de reconocimiento de posicionamiento global (GPS) entre otras cosas, consiguiendo así, con ayuda de sistemas de control que manejen la información recopilada, un vehículo “autónomo” que es aproximadamente 8 veces más seguro que un vehículo normal [3].

En el año 2003 la universidad de Wageningen creó un evento anual llamado FRE (por sus siglas en inglés provenientes de *Field Robot Event*) en el cual distintos grupos de estudiantes diseñan y fabrican un robot autónomo el cual deberá de cumplir con distintas tareas de campo con el fin de encontrar soluciones creativas para la implementación de robots en la agricultura [4].

En este evento, los robots se colocan dentro de un terreno de cultivos los cuales tendrá que navegar de manera acertada cumpliendo/superando desafíos que los agricultores se enfrentan día a día. Los robots deben de ser capaces de navegar por el terreno completo, recoger "basura", sortear obstáculos, entre otras actividades. Cabe resaltar que estos deben de ser pilotados de manera autónoma, esto quiere decir, no deben de ser controlados por ningún operario [5] [6].

El robot secador de café se inspiró en esta competencia, debido a la similitud de la finalidad de los robots los cuales deben de trabajar en un campo siguiendo un camino de manera autónoma y cumpliendo con algunas tareas. A diferencia de los robots presentados en la competencia, el robot secador de café tiene la finalidad de mover el café siguiendo una trayectoria especifica durante 7 horas continuas, una diferencia notoria comparada con los cortos periodos de tiempo que se evalúan en la competencia. Esto genera la necesidad de implementar una mayor capacidad de batería en el diseño, sin embargo, el uso de cámaras y visión por computadora no es relevante para nosotros, pero si el poder ubicar de manera precisa el posicionamiento del robot por lo que el uso de diferentes sensores de posicionamiento es necesario, entre otras diferencias [7].

Este es el tercer prototipo realizado del robot secador de café, el primer y segundo prototipo lo elaboró el ingeniero Frank Jonathan Saenz Paz y el ingeniero Alejandro José Windevoxhel Hibjan los cuales establecieron las bases para el diseño, la estructura, el sistema de control y los módulos eléctricos/electrónicos a utilizarse para el tercer prototipo [1].

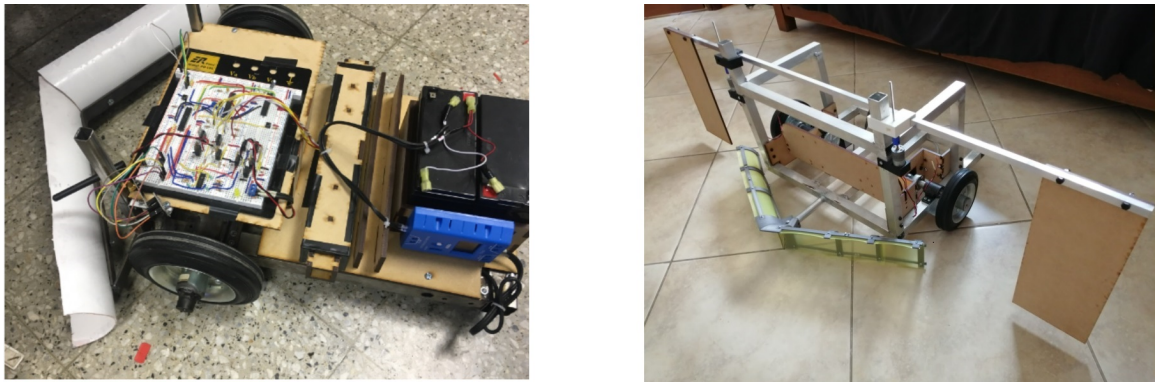


Figura 1: Prototipo 1 y 2 - Robot secador de café

CAPÍTULO 3

Justificación

La economía de Guatemala, al ser un país agrícola, depende mucho de las exportaciones de productos entre los cuales podemos mencionar como principales productos a los artículos de vestuario, cardamomo, banano, café, grasas y aceites comestibles, azúcar, entre otros. Esto provoca que aproximadamente el 14.57 % del PIB guatemalteco provenga de exportaciones [8] [9].

El café guatemalteco es reconocido a lo largo de todo el mundo como "café único" debido a su variedad de sabores, granos, aroma, post sabor, acidez, entre otras características, lo que hace ser a Guatemala uno de los mayores proveedores de café mundialmente. Entre los principales destinos de exportación de café se encuentra gran parte del oriente medio y Europa, alcanzando cifras de hasta 3.5 millones de sacos de 60 kilos de café exportados en la temporada 2016/2017 y aumentando en un 5.3 % para el 2019, representó un total de US $663.62 millones y el 7.32 % de las exportaciones totales del país en el 2019 [10] [11].

Lo que hace especial al café guatemalteco son los más de trecientos microclimas existentes en todo el territorio del país y los métodos de cultivo que se han heredado de generación en generación a lo largo de los años, provocando que el café tenga ese toque distintivo y único. Se cultiva una gran variedad de café a lo largo de los 108,889 km2 que conforman a Guatemala, esto se debe a las distintas regiones donde se cultiva el café, cómo las regiones de Acatenango, Antigua Guatemala, Atitlán, Cobán, Fraijanes, Huehuetenango, Oriente y San marcos. Cada región cuenta con métodos de cultivo distintos y cada café tiene características distintivas como: un sabor, aroma, rigidez del grano, entre otras. Las características únicas del café de región se deben gracias a las diferentes alturas sobre el nivel del mar, los volcanes que rodean a la región que hacen que las tierras sean más fértiles y cuenten con otros minerales, el clima de la región, la cantidad de humedad, la luz solar, etc [10].

Entre los principales productores de café en Guatemala se encuentra la finca El Injerto ubicada en la región de Huehuetenango. Este café se conoce como café montañoso debido a que este se siembra en la sierra de los Cuchumatanes, la sierra más alta de Centro América. Esta región se encuentra a una altura de 2000 a 2300 metros sobre el nivel del mar y por ser una región montañosa se considera una región húmeda [10].

Una de las principales características de esta productora de café es el secado. La finca El Injerto seca el café de forma "artesanal" al esparcir el café en un patio y luego con palas/rastrillo manipuladas por una persona, pasan "rayando" toda el área donde se esparció el café en un patrón de columnas y luego filas. Este procedimiento se realiza durante aproximadamente 7 horas del día, moviendo los granos para que estos se sequen al darles el sol. Este es un proceso de secado poco convencional ya que se requiere de un operario que esté manipulando el rastrillo y caminando durante las 7 horas seguidas.

Lo que se busca con el robot secador de café es automatizar este proceso cubriendo así la necesidad de mover y voltear el café de forma constante durante 7 horas seguidas. Las ventajas que presenta el robot secador de café ante el operario son que el robot podrá realizar esta tarea sin necesidad de descansar o bajar el ritmo de trabajo, también podemos mencionar la eficiencia que presenta el robot y la seguridad que cada proceso de secado será estandarizado, obteniendo así un producto de calidad tal como lo ha venido entregando la finca El Injerto a lo largo de los años.

CAPÍTULO 4

Objetivos

4.1. Objetivo general

Implementar un sistema de control automatizado para un robot secador de café.

4.2. Objetivos específicos

- Diseñar y construir un prototipo a escala con un sistema de control que utilice encoders, unidades de medición inercial y módulos de posicionamiento en tiempo real.
- Instalar y probar el sistema de control completo en la estructura real del robot secador de café.
- Realizar un listado de costos de materiales electrónicos y fabricación de circuitos impresos.

CAPÍTULO 5

Alcance

Se diseñó un prototipo a escala del robot completamente funcional (sección 7) en el cual se implementaron todos los módulos y controladores a utilizar en el robot real. En este modelo a escala se realizaron pruebas individuales de cada módulo (DWM1001-dev, Encoders, ATMega256, DC Motor Driver 2x15A, NRF24L01) y se implementaron otros módulos cómo la unidad de medición inercial (IMU).

Se realizaron pruebas de funcionamiento y control de los motores utilizando el DC Motor Driver 2x15A para alimentar y controlar los motores que se utilizaron en el robot.

Se implementó un sistema de control capaz de obtener lecturas de la velocidad angular de las ruedas a partir de encoders ópticos con los cuales se definió la pose del robot. En conjunto a esto, se realizaron diferentes sistemas de control a partir de los encoders ópticos para controlar la velocidad. Se optó por utilizar el controlador comparador para el robot móvil a escala en base a las lecturas de los encoders para controlar la velocidad de las ruedas y hacer que el robot se mueva en línea relativamente recta.

Se realizó un sistema de localización en tiempo real (*RTLS*) con ayuda de los módulos DWM1001-DEV con el cual se leyeron y manipularon los datos obtenidos con el microcontrolador para formar y observar la posición del robot en todo momento.

Se realizaron pruebas de la lectura de posición del robot con el robot estático y en movimiento (desplazamiento horizontal y vertical). Además, se hicieron dos filtros para estabilizar las lecturas de posición y obtener una medición más estable y certera.

Se realizó un sistema de control dónde se estableció una ruta la cual debe de seguir el robot a escala. El robot recopila la información de su posición utilizando el sistema de localización en tiempo real *RTLS*. Este al conocer su posición reconoce si se desvió de la ruta establecida o no. Si el robot detecta que se encuentra en una posición fuera de la ruta establecida, retoma el curso para volver a entrar a la ruta manipulando la velocidad de las ruedas. Sin embargo, debido a problemas con el hardware del sistema de localización en timepo real *RTLS*, no se pudo completar el sistema de control utilizando los módulos

DWM1001-DEV.

Debido a la escasez de chips a nivel mundial, no se pudo adquirir más módulos DWM1001-DEV para robustecer el sistema de localización en tiempo real (*RTLS*) ya que únicamente se cuentan con 4 dispositivos DWM1001 actualmente.

Debido a la deficiencia del sistema de localización en tiempo real (*RTLS*) conformado por tres *Anchors* y un *TAG*, se encontraron alternativas para la obtención de datos más certeros cómo la posición o distancia vectorial del *Anchors* hacia el *TAG*. Gracias a esto se desarrolló una guía del uso de los DWM1001-DEV para formar el sistema de localización en tiempo real (*RTLS*). En esta guia se describe la configuración del hardware, la configuración de cada uno de los DWM1001-DEV para llevar a cabo el roll de *Anchor* o *TAG*, la solicitud de datos del *TAG* hacia los *Anchors*, la lectura de datos recibidos de los *Anchors*, manipulación de datos para formar valores de posición y distancia del *TAG* y manejo de datos.

Debido a la deficiencia del sistema de localización en tiempo real (*RTLS*) actual, no se pudo completar el sistema de control realizando la fusión de sensores, sin embargo, se realizó cada uno de los controladores funcionales por separado.

Se realizó el sistema control de cuatro diferentes controles remotos, específicamente la parte de recepción y manipulación de datos para controlar al robot secador de café vía radio frecuencia utilizando módulos NRF24L01.

Se realizó un listado de costo de material (*Bill of materials*) del prototipo a escala del robot secador de café.

Debido a la pandemia mundial del Covid-19, no se pudieron completar todos los objetivos planteados tales cómo la implementación del sistema de control en el robot secador de café, en su lugar se utilizó el prototipo a escala del mismo.

Se implementó un control remoto de cada motor independiente.

Se implementó el control remoto vía radio frecuencia al robot secador de café.

Se realizaron pruebas de campo con el robot secador de café en las cuales se controló vía radio frecuencia con un control remoto.

CAPÍTULO 6

Marco teórico

6.1. Sensores y módulos

6.1.1. DWM1001-DEV

El módulo DWM1001-DEV (Figura 2), fabricado por la compañía DecaWave, puede implementarse para cumplir varias funciones, principalmente con la función de determinar la posición de un objeto en tiempo real. Al implementar un juego de módulos DWM1001-DEV (4 como mínimo) se logra formar un Sistema de ubicación en tiempo real o por sus siglas en ingles *RTLS* [12].

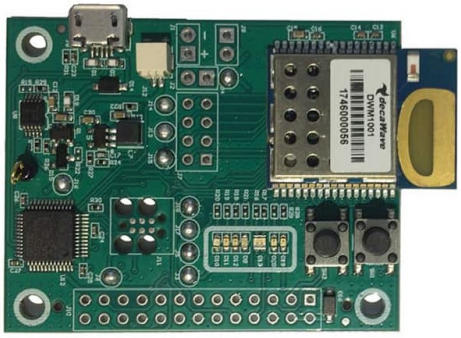


Figura 2: Módulo DWM1001.

Los *RTLS* se utilizan para poder conocer la ubicación física de un objeto. Esto se logra a través del uso de "*TAGS*" activos los cuales envían y reciben información de los módulos secundarios conocidos como "*Anchor*". Estos módulos (*TAGS* y *Anchors*) se comunican entre ellos continuamente a través de banda ultra ancha (*UWB*) a 6.5GHz en intervalos de tiempo establecidos. [13]. Cabe mencionar que esta comunicación también puede establecerse por medio de radio frecuencia, Wifi y bluetooth [14].

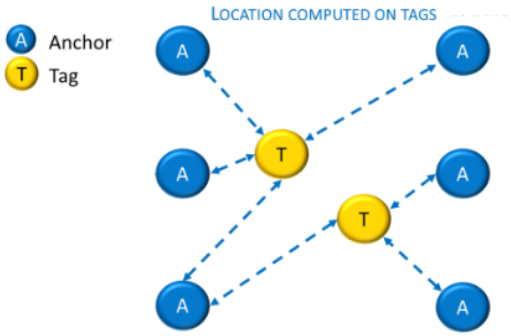


Figura 3: Ejemplo de sistema *RTLS* con módulo DWM1001

Para poder armar un sistema de ubicación en tiempo real (*RTLS*) es necesario contar con tres *Anchors* y un *TAG* cómo mínimo. La función de los Anchors es establecer un perímetro fijo dentro del cual se ubicará el TAG (Figura 3). El *TAG* es el dispositivo del cual se desea conocer la posición en tiempo real. Al colocar los *Anchor* en diferentes ubicaciones, el *TAG* funcionará como un sistema de coordenadas 3D móvil o fijo, dependiendo de las necesidades del proyecto, el cual se encargará de comunicarse con cada uno de los *Anchors* uno por uno enviando una señal y dependiendo del tiempo que demore la señal en llegar se estima la posición del *TAG*. Esta técnica se llama localización por triangulación (Figura 4) he implica la medición de la distancia de un objeto desde tres puntos diferentes. Este mismo principio es el utilizado por los dispositivos *GPS* [14] [15].

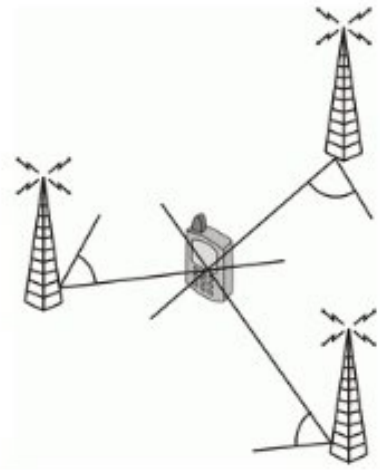


Figura 4: Triangulación

6.1.2. Unidad de medición inercial (*IMU*)

La unidad de medición inercial o por sus siglas en ingles *IMU*, es un sistema que mide la velocidad, orientación y fuerza gravitacional de un dispositivo electrónico. La *IMU* está compuesta de magnetómetros, acelerómetros y giroscopios que permiten la lectura de la inclinación, balanceo, ángulo de rotación, aceleración, velocidad y velocidad angular del dispositivo partiendo de un sistema de coordenadas inerciales (Figura 5) [16].

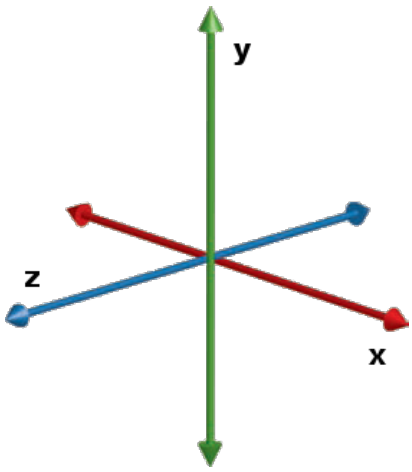


Figura 5: Eje de coordenadas

El magnetómetro es el encargado de medir el campo gravitacional del planeta para poder ubicar el norte magnético. El acelerómetro mide la aceleración en cada uno de los 3 ejes de coordenadas espaciales tomando en cuenta la aceleración generada por la fuerza gravitacional. El giroscopio se encarga de medir la velocidad angular respecto a los tres ejes del espacio. Estos tres dispositivos combinados conforman a la Unidad de medición inercial. Las *IMU* pueden ser de 3, 6 o 9 ejes o grados de libertad dependiendo de la cantidad de mediciones que puedan realizar. Para este proyecto se utilizó una *IMU* de 9 grados de libertad [17].

Para el robot secador de café se decidió implementar una *IMU* debido a que esta nos permite conocer la orientación del robot, específicamente los ángulos Roll, Pitch y Yaw (Figura 6), aunque únicamente utilizaremos la rotación en Z (ángulo Yaw) debido a que el robot se mueve en un plano 2D en los ejes X y Y por lo que su rotación en los ángulos Roll y Pitch será cero idealmente.

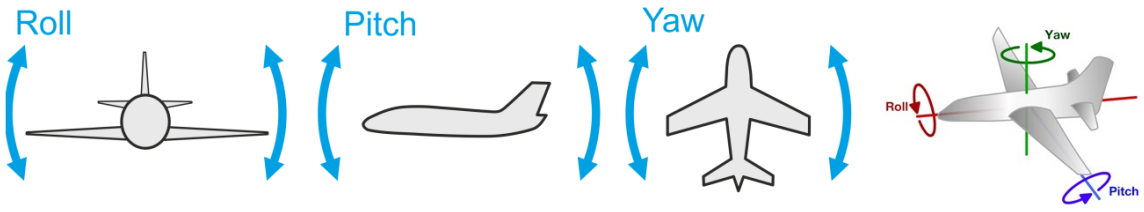


Figura 6: Roll, Pitch, Yaw

6.1.3. Módulo F249 – Encoder

El encoder es un dispositivo electromecánico que traduce la velocidad lineal, posición angular o velocidad de un motor en pulsos digitales, pulsos binarios, pulsos analógicos en función de una onda, pulsos eléctricos, entre otros, que pueden ser interpretados por algún microcontrolador, PLC o algún otro dispositivo o circuito para luego manipular esta información recopilada a conveniencia [18] [19].

Existen diferentes maneras en las que los encoders recopilan esta información para luego enviarla al controlador, entre estas podemos mencionar: mecánica, magnética, óptica y de resistencia. En el robot secador de café se utilizará un tipo de encoder óptico para la detección de la velocidad, y posición de las ruedas [19].

Figura 7: Módulo F249

El módulo F249 (Figura 7) está compuesto por un circuito LM393 FC-03. Este módulo a grandes rasgos funciona cómo un optoacoplador el cual es un interruptor que es activado mediante una luz infrarroja emitida por un diodo led (emisor) hacia un fototransistor (receptor). Cuando ocurre una interrupción entre el fototransistor y el diodo infrarrojo, el circuito se "abre", tal como lo haría un interruptor o un switch al cambiar de encendido a apagado (Figura 8) [20].

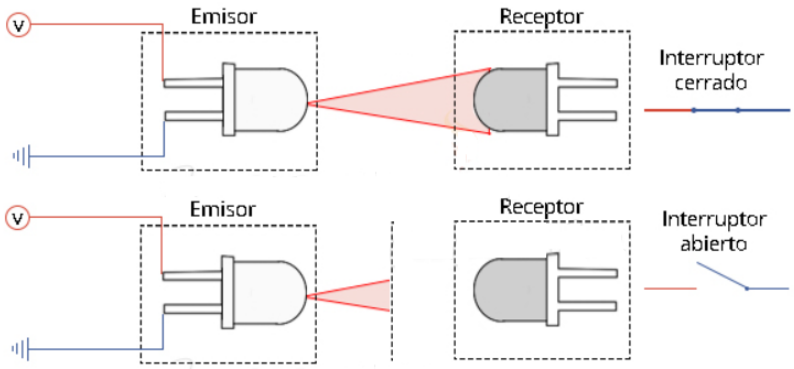


Figura 8: Funcionamiento de un optoacoplador

Para formar el encoder completo, también es necesario agregar un disco ranurado el cual va sujetado el eje giratorio del motor o rueda el cual se encargará de interrumpir la comunicación entre el emisor y receptor del módulo F249 (Figura 9). El disco ranurado, al girar en conjunto a la rueda, va interrumpiendo constantemente la comunicación emisor-receptor generando así una serie de pulsos los cuales son recopilados por el microcontrolador Arduino MEGA (en el caso del robot secador de café) e interpretados para calcular el desplazamiento lineal que ha realizado la rueda.

Cabe mencionar que se debe de colocar un encoder en cada llanta del robot para poder controlar de manera adecuada el desplazamiento y ver si una rueda no avanza más que la otra. También es importante conocer la cantidad de agujeros o ranuras con las que cuenta el disco ranurado para poder conocer la cantidad de pulsos por giro de la rueda.

Si desean conocer más al respecto del uso del módulo F249 como encoder para controlar un “4wd Smart car robot” pueden visitar el siguiente enlace [21].

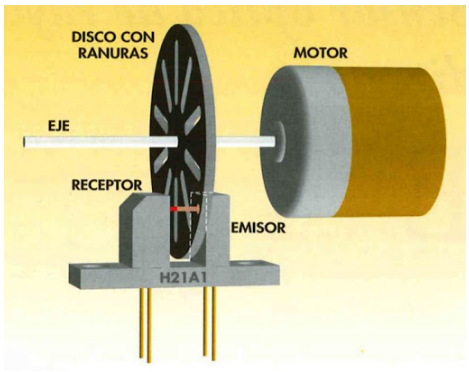


Figura 9: Encoder

6.1.4. Módulo DRI0018 – Puente H

Para poder controlar la dirección de giro, la velocidad, y la dirección (avanzar o retroceder), es necesario poder controlar de forma individual cada llanta, específicamente, el motor que controla a esta llanta. Para poder cumplir con esta tarea es muy común utilizar un circuito conocido cómo “Puente H” compuesto por transistores NPN, diodos y resistencias (Figura 10). A grandes rasgos, el puente H lo que hace es que al alimentar la entrada 1, Q1 y Q4 se cierran permitiendo el flujo de corriente y el motor girará en una dirección, y al alimentar la entrada 2, Q2 y Q3 se cierran permitiendo el flujo de corriente y provocando que el motor gire en la dirección contraria. Cabe resaltar que las entradas 1 y 2 no deben de ser alimentadas al mismo tiempo. Entre las limitaciones que presenta este circuito podemos mencionar que está sujeto a la corriente máxima que soportan los componentes electrónicos. [22]

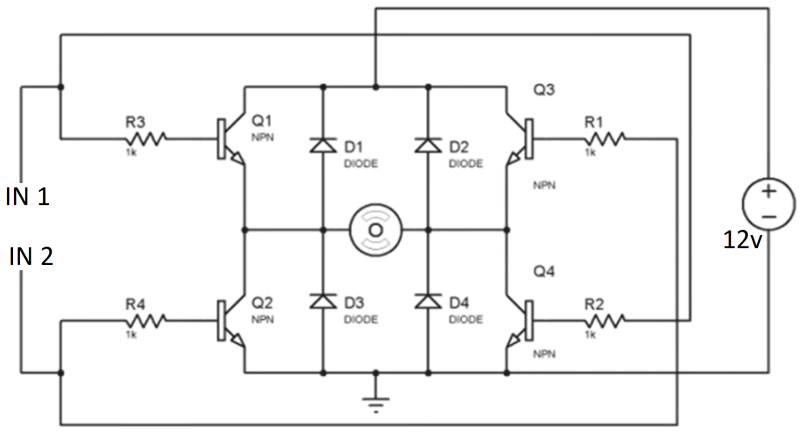


Figura 10: Puente H

Usar un puente H es una manera efectiva de alimentar y controlar la dirección de un motor, sin embargo, en el robot secador de café, se desea utilizar altos amperajes para poder activar los motores, esto se debe al tamaño de ruedas que se desea mover, el tamaño y

potencia que demandan los motores que se están utilizando y en general, el tamaño del robot completo. Por lo que se optó por utilizar el módulo DRI0018 (Figura 11) el cual cumple esta misma función que el puente H, con la ventaja que este es capaz de controlar 2 motores independientes a la vez, cuenta con 4 entradas de control (2 para un motor y 2 para otro motor), el voltaje de entrada es de 4.8 a 35v, la corriente máxima de salida es de 15A por canal, cuenta con PWM incluido, aislamiento galvánico para protección del microcontrolador u otros dispositivos electrónicos, protección contra cortocircuitos, sobre calentamiento y sobre tensión [23]. Haciéndolo el módulo ideal para el robot secador de café.

Figura 11: Módulo DRI0018

6.1.5. Módulo TXS0108E

Al tener una gran cantidad de componentes y módulos integrados, nos encontramos con la tarea de tener que manejar distintas cantidades de voltajes. Este es el caso del robot secador de café, el cual cuenta con un Arduino MEGA, módulos DWM1001, NRF24L01, Módulo de la *IMU*, Módulo F249 y un circuito para el final de carrera. Todos estos componentes se manejan con voltajes de 12, 5 y 3.3 voltios. Tenemos la ventaja que el Arduino MEGA se alimenta con 12v y entrega en su salida 3.3 voltios, sin embargo, el problema se encuentra al momento de querer obtener los 5 voltios. Para esto se utiliza el módulo TXS0108E (Figura 14) el cual nos permite tener la conversión de 3.3 v a 5 v y viceversa sin necesidad de realizar un switcheo o controlar de alguna manera el cambio de voltajes [24].

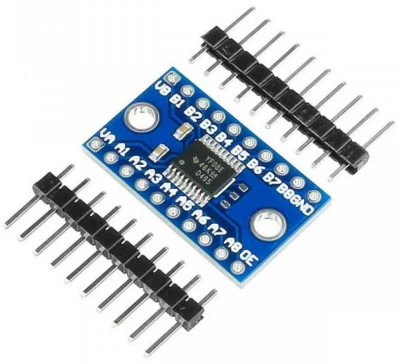


Figura 12: Módulo TXS0108E

El módulo TXS0108E cuenta con dos canales (A y B). El canal A maneja los 3.3 v tanto de entrada como salida y el puerto B se encarga de manejar los 5v tanto de entrada como salida. Ambos canales cuentan con 8 pines enumerados (A1, A2, A3.. A8 y lo mismo para el puerto B) los cuales están conectados internamente entre su equivalente del otro puerto, esto quiere decir que A1 está conectado con B1, A2 con B2 y así hasta A8 y B8. Cabe mencionar que todos los puertos, tanto los puertos de A como los de B, pueden funcionar cómo entradas o salidas independientes [24].

6.2. Microcontrolador

6.2.1. Arduino Mega

Los microcontroladores son circuitos integrados diseñados para realizar distintas tareas en sistemas embebidos. El tamaño reducido, la versatilidad y su arquitectura los hacen ser ideales para una infinita cantidad de tareas o propósitos. [25] [26]

Estos pequeños dispositivos cuentan con una unidad de procesamiento (CPU), memorias RAM y ROM y periféricos (entradas y salidas). Entre las mayores características de los microcontroladores se encuentra la posibilidad de programarlos para que realicen distintas tareas por medio de software con ayuda de hardware. Al tener entradas y salidas, los microcontroladores son capaces de recopilar y manipular información proveniente de sensores o pulsos a través de sus periféricos y realizar alguna acción específica descrita en código. Los microcontroladores cuentan con memorias para almacenar datos, convertidores de voltaje, sistemas de buses, puertos seriales, entre otras características. Existen distintos microcontroladores, algunos con más memoria, otros con más puertos periféricos, otros con mayor velocidad de procesamiento y así se encuentran otros, por lo que encontrar algún microcontrolador que cumpla con algunas necesidades no será ningún problema. [25] [26]

Para el robot secador de café, se optó por utilizar el Arduino MEGA debido a la velocidad de procesamiento. Esta es una placa que cuenta con un microcontrolador "ATmega2560" (Figura 13). Este microcontrolador cuenta con 32 bits de procesamiento, 54 puertos digitales de los cuales 15 pueden ser utilizados como PWM, 16 pines analógicos, 4 puertos seriales, un reloj de hasta 16 MHz, dos DAC's, 256 bytes de memoria Flash y 8 K-bytes de memoria SRAM. entre otras características. [27]

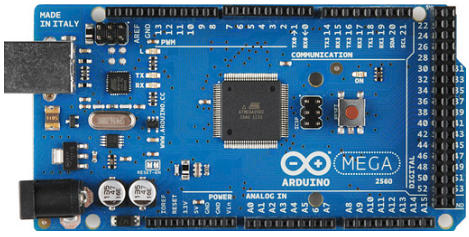


Figura 13: Arduino Mega

6.3. Control

6.3.1. Odometría

La odometría define la posición/ubicación de un vehículo a partir de diferentes sensores. Existen diferentes tipos de odometría, la odometría visual la cual utiliza cámaras y captura de imágenes para conocer la pose/posición del vehículo en el instante deseado y la odometría normal o llamada también odometría de posición la cual utiliza diferentes sensores no visuales para recaudar la información que mejor describa la pose del vehículo.

Se le llama odometría de posición a la estimación de la pose de un vehículo con ruedas a lo largo de una trayectoria o mientras navega. La odometría de posición ocupa la información de la velocidad angular de las ruedas para conocer la rotación de cada una de ellas a lo largo de un tiempo determinado y a partir de un punto de inicio, conocer la posición actual del vehículo utilizando el desplazamiento lineal.

La odometría es la integración de información incremental del movimiento a lo largo del tiempo, lo que acarrea una ineludible acumulación de fallos. Específicamente, la acumulación de fallos de orientación. Sin embargo, la implementación de la odometría en sistemas de navegación de robots móviles es esencial. [28]

De las maneras más comunes de implementar la odometría de posición en robots móviles con ruedas, es considerando el uso de encoders los cuales indicarán el movimiento angular de la rueda. Sin embargo, también nos encontramos con que la implementación de la odometría con encoders no es suficiente para definir la pose de un vehículo. Partimos desde la suposición de que las revoluciones de las ruedas pueden ser traducidas en un desplazamiento lineal relativo al suelo, del cual podemos diferir al momento que exista el efecto de deslizamiento en una de las ruedas y esta gire registrando una distancia lineal “n” pero no habiéndose movido en lo absoluto de su ubicación actual debido al deslizamiento. Por lo general podemos categorizar los fallos de la odometría en dos grupos: fallos sistemáticos, y fallos no sistemáticos. [28]

Los fallos sistemáticos se pueden representar como los fallos del sistema, esto quiere decir, los relacionados con el vehículo. Y en los fallos no sistemáticos se encuentran los fallos provocados por el exterior o factores externos.

6.4. Protocolos de comunicación

En en mundo de la electrónica actual, es muy común tener varios dispositivos o módulos que generan y reciben datos. La lectura y manipulación de los datos entre dispositivos es lo que nos permite crear sistemas de control para cumplir con una finalidad. Sin embargo, a veces se utilizan dispositivos externos para recopilar cierta información por lo que surge la necesidad de lograr comunicar a ambos módulos o dispositivos para que estos se transfieran los datos entre ellos. Para esta necesidad existen diferentes protocolos de comunicación cómo la comunicación UART, SPI, I2C, etc. Sin embargo, todos estos protocolos pertenecen a una categoría específica, Serie o Paralelo [29].

6.4.1. Comunicación en serie y en paralelo

La diferencia entre estas dos categorías se encuentra en el número de bits que transmiten simultáneamente por lo que también cuenta la cantidad de buses de datos y periféricos necesarios.

La comunicación en serie transmite sus datos un bit a la vez a través de un solo cable, por lo que generalmente es necesario no más de cuatro periféricos para su implementación.

En cambio, la comunicación paralela utiliza un cable por cada bit de transmisión.

Ambas categorías tienen sus ventajas y desventajas, dependiendo de las restricciones y necesidades de implementación. La comunicación en paralelo puede ser más rápida y directa que la comunicación serial, pero su implementación por lo general requiere más periféricos, cables y por coincidente tiene un costo más elevado. En cambio, la comunicación serial destaca por la sencillez de implementación de hardware y por qué no se limita a la cantidad de periféricos disponibles [29].

A pesar de que ambas categorías son útiles y funcionales, cada vez se apuesta más por la comunicación serial debido a que los tiempos de transmisión ahora son tan rápidos que prácticamente no hay diferencia significativa entre transmitir bit por bit que a transmitir N cantidad de bits a la vez. Además de su implementación en la cual no se ven limitad por los periféricos de los dispositivos, la comunicación serial resalta cómo una comunicación ideal.

Comunicación síncrona y asíncrona

Al hablar de comunicación, es relevante mencionar que también existen dos subcategorías, la comunicación asíncrona y síncrona.

La comunicación síncrona requiere de un agente externo que le indique cuando se va a escribir o se va a recibir un dato por lo general este agente está compuesto por una señal de reloj, en cambio en la comunicación asíncrona esta señal de reloj no es necesaria [29]. En la comunicación Serial asíncrona podemos encontrar distintos protocolos de comunicación cómo el USB, ETHERNET, TX/RX, etc.

Para establecer una comunicación asíncrona en serie es necesario establecer ciertas reglas para garantizar la transmisión de datos [29], estas reglas son:

- Bits de datos
- Bits de sincronía
- Bits de paridad
- Velocidad de baudios

Velocidad de baudios

La velocidad de baudios establece la velocidad de transmisión de datos del dispositivo a través de una línea serial. Su unidad son los bits por segundo. La velocidad de baudios es muy importante en la comunicación serial porque nos indica cuanto tiempo la línea se mantiene alto o en bajo. Además, para establecer una comunicación serial, es necesario que ambos dispositivos a comunicarse tengan la misma velocidad de baudios para que estos puedan enlazarse de manera correcta [29].

Entre los valores de baudios más utilizados podemos encontrar: 1200, 2400, 4800, 9600, 19200, 38400, 57600 y 115200. Aunque se pueden establecer otras velocidades no mencionadas.

Bits de sincronización

Para establecer la comunicación el dispositivo debe de ser capaz de saber cuándo le van a escribir algún dato y cuando ya puede dejar de leer el dato. Para esto se utilizan los bits de sincronización. Los bits de sincronización se encuentran integrados directamente en la cadena de datos. Estos bits nos indican el inicio del dato a recibir y el final del dato [29].

Bits de paridad

Este bit no es tan usado en la comunicación serial, por lo general se puede hacer caso omiso a la utilización de este bit.

El bit de paridad establece una comunicación segura entre dos dispositivos. Funciona cómo un tipo de seguro para afirmar que los datos recibidos si deben de ser leídos por el dispositivo o no. Por lo general este bit se utiliza si tenemos un bus de datos en el cual se tienen diferentes dispositivos conectados al mismo bus. Este bit no ayuda a establecer a cuál de los dispositivos de ese bus de datos es a que le queremos escribir y hacer caso omiso de los demás dispositivos. También puede funcionar cómo filtro en señales que presentan grandes cantidades de ruido.

El problema con el bit de paridad es que a veces ralentiza la comunicación, pueden existir pérdidas de datos y además es necesario implementar un manejo de errores de comunicación [29].

Hardware

La comunicación serial consta de dos cables, uno de recepción de datos y el otro de envío de datos. El pin de emisor de un dispositivo debe de conectarse al pin de recepción del otro dispositivo y viceversa.

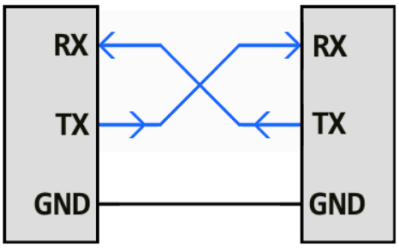


Figura 14: Módulo TXS0108E

6.4.2. Radio Frecuencia

La señal de radio frecuencia toma el concepto de una señal electromagnética inalámbrica para la transmisión de datos. Como su nombre lo indica, utiliza ondas de radio que van desde los 3kHz hasta los 300GHz. La tasa de oscilación de las ondas de radio se denominan frecuencia.

Entre las grandes ventajas de la radio frecuencia se encuentra que estas no requieren de un medio de transmisión.

Por lo general, los sistemas de radio frecuencia están compuestos por un emisor y receptor. En la mayoría de los casos la comunicación se produce en ambas vías. Como la radio frecuencia no utiliza medio para propagarse, por lo general los dispositivos cuentan con una antena o un elemento que ayuda a aumentar el alcance de la comunicación[29].

CAPÍTULO 7

Prototipo a escala - 2WD Smart Robot

7.1. Estructura

El prototipo a escala del robot secador de café se utilizó para realizar pruebas del sistema de control, lectura de sensores, y funcionamiento del mecanismo en general previo a la implementación del sistema completo en el robot secador de café. Debido a la pandemia del Covid-19 y temas estructurales, únicamente se realizaron pruebas de campo en el robot secador de café, no con el prototipo a escala

Para el prototipo a escala se utilizó el Kit de robótica *2 wheel drive motor chassis robotics Kit* o también llamado *2WD Smart robot* el cual cuenta con una base principal de acrílico para montar los componentes, los motores que moverán al vehículo, un porta baterías AA de 4 unidades, dos motores de 6v DC 100 rpm, 2 ruedas, dos discos ranurados, un interruptor y una rueda loca (Figura 15).

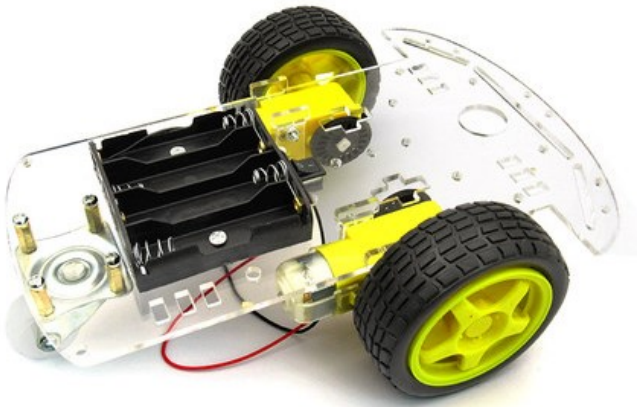


Figura 15: 2WD Smart robot

Debido a la magnitud y requerimientos del robot secador de café, se adaptó el *2WD Smart robot* a las necesidades del proyecto, para esto se adquirió una segunda unidad del *Smart Kit* y se colocaron ambas plataformas una encima de la otra totalmente alineadas. Se colocaron también separadores de 51 milímetros entre ambas plataformas acrílicas de tal forma que se aprovecharon ambas plataformas para colocar los componentes y sensores necesarios en el prototipo a escala (Figura 16).

Figura 16: Ejemplo de un Smart robot multi plataforma

7.2. Componentes

Los componentes/sensores que se colocaron el prototipo a escala del robot secador de café son:

- IMU 9GDL
- DWM-1001 DEV
- Microcontrolador Arduino Mega con chip ATmega2560
- 2 MH sensors (encoders)
- Porta baterías tamaño 18650
- Protoboard pequeño con circuito transformador de voltaje
- Protoboard pequeño con circuito bottom push de arranque
- Módulo "DC motor driver 2x15A lite"
- Switch de arranque
- Antena NRF24L01

Estos componentes se colocaron de tal manera que no interfirieron el uno con el otro al momento de cablear al smart robot completo. Se tomó en cuenta que algunos componentes como el módulo DC motor driver debía de implementarse en el robot secador de café por lo que se posicionó en un lugar dónde fue posible retirarlo de manera rápida.

7.3. Diseño 3D

Cómo se colocaron muchos componentes en el *2wd Smart robot kit*, se necesitó de un plan de conexión para que todo funcionara y estuviera en su lugar. Para esto se realizó el modelo 3D del *Smart robot* y todos los componentes que lo integran, con ayuda del diseño 3D se hicieron pruebas de ubicación de todos los componentes y se comprobó que estos no estorbarón los unos con los otro. Luego se implementó esta idea en el prototipo a escala real.

Para realizar los modelos 3D de todas las piezas, motores, sensores, chasis, y extras, se utilizó el programa de computación *AUTODESK INVENTOR* el cual permitió diseñar, modelar he integrar componentes de manera fácil. Ademas se utilizó el programa *ULTIMAKER CURA* para imprimir algunos de las piezas utilizadas.

Para realizar el modelo completo del prototipo a escala del robot secador de café, primero se buscaron los componentes básicos en internet ya que el *2wd Smart robot kit* es un producto muy comercial, fue fácil encontrar los modelos 3D de algunos componentes tales como el chasis, los motores DC, rueda loca, placa de microcontrolador Arduino, Dc motor driver 2x15A, llantas, entre otros. Para esto se utilizaron distintas paginas de modelos 3D de código abierto [30] [31].

Para realizar los modelos 3D de los sensores, módulos y componentes que no se encontraron en internet, se utilizaron las hojas de datos de cada producto y se comparó las medidas indicadas en las mismas con medidas tomadas directamente de la pieza con ayuda de un vernier con 0.02 milímetros de precisión. Se tomaron un total de 5 medidas por lado a dimensionar para tener un margen de error mínimo en la toma de mediciones.

Los componentes descargados son:

Componentes	**Link de descarga**
Rueda de motor	[32]
Motor DC 6v	[33]
Rueda ranurada	[34]
Arduino Mega	[35]
DC Motor Driver 2x10A	[36]
Regulador L7805CV	[37]
Caster Wheel	[36]
Regulador L7805CV	[38]

Cuadro 1: Componentes descargados y link de descarga.

Los componentes diseñados son:

- Chasis primer piso
- Chasis segundo piso
- Conector de baterias
- DWM1001 DEV
- Base para IMU
- Encoder
- IMU 9GDL
- Junta para ruedas
- Juntas para DWM
- Junta para plataformas
- Protoboard
- Sostenedor del motor

Una vez diseñados los componentes, se creó un ensamble 3D con todos los componentes en el cual se ubicaron los componentes uno a uno tomando en cuenta su funcionalidad, tamaño y necesidades de implementación. Para esto se partió de la ubicación de la base o chasis del primer piso y se colocaron todos los componentes necesarios tanto en la parte superior he inferior considerando los tamaños para que estos no se estorben (Figura 17)

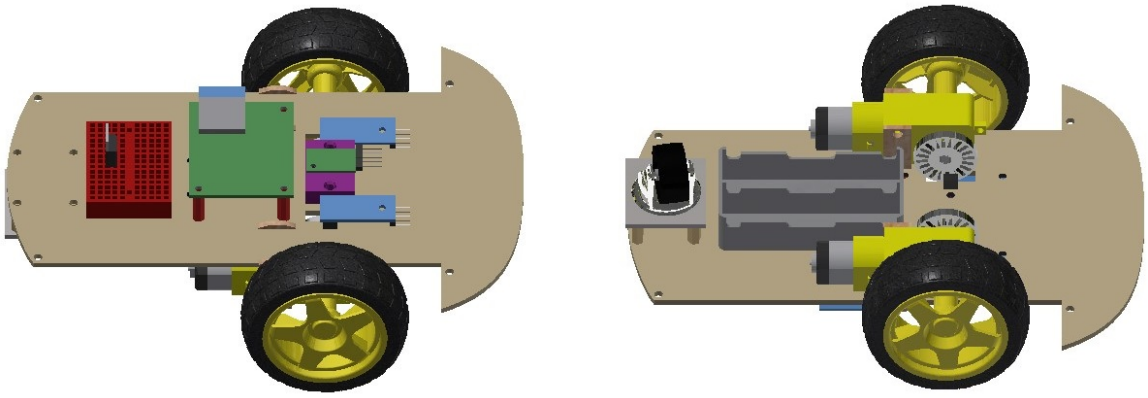


Figura 17: 2WD Smart robot primer piso

Luego de que se tenía una altura máxima del primer piso del Smart robot, se diseñaron los separadores del primer piso con el segundo piso tomando en cuenta que estos serían lo suficientemente largos para que no hubiera interferencia con algún componente al momento de armar el robot.

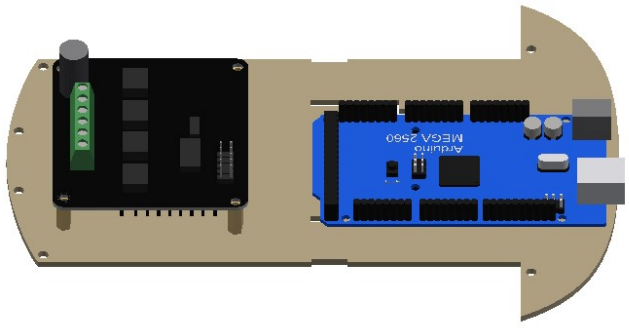


Figura 18: 2WD Smart robot segundo piso

Seguido a esto se ubicaron los componentes que irían en el segundo piso del prototipo (Figura 18) y se prosiguió a unir ambas partes, primer y segundo piso, para tener así el modelo 3D del robot completo, el cual se replicó para el prototipo del robot secador de café físicamente (Figura 19).

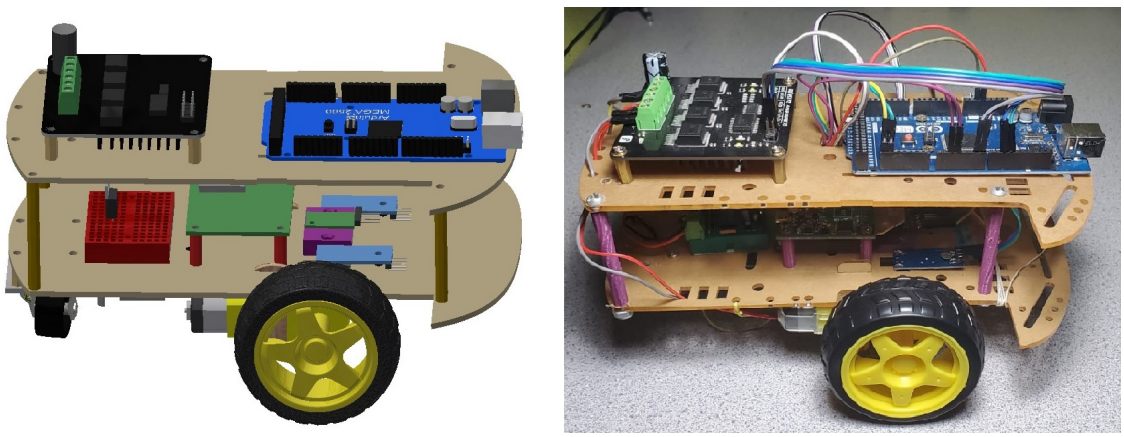


Figura 19: Modelo 3D y prototipo del 2WD Smart robot completo

CAPÍTULO 8

Modelo diferencial del robot móvil

8.1. Definición de la velocidad del robot - Modelo diferencial

Por simplicidad se representó al robot móvil con un modelo diferencial simple el cual ayudó a describir la velocidad lineal y angular del *Smart robot* y la posición en la que este se encuentra. (Figura 20). Con el modelo diferencial del robot móvil, se establecieron tres ecuaciones que ayudaron a definir los parámetros de velocidad de las ruedas.

$$\dot{x} = \frac{R}{2}(v_r + v_l)cos(\theta) \tag{1}$$

$$\dot{y} = \frac{R}{2}(v_r + v_l)sin(\theta) \tag{2}$$

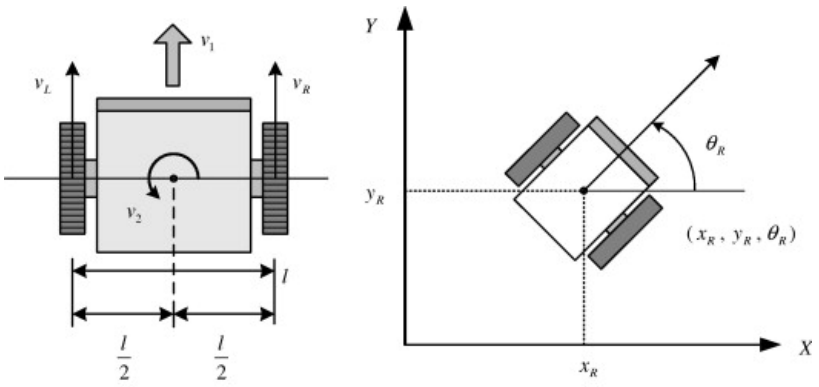
$$\dot{\theta} = \frac{R}{L}(v_r - v_l) \tag{3}$$

También se definió la velocidad lineal y angular a partir de las entradas v y ω.

$$\dot{x} = v * cos(\theta) \tag{4}$$

$$\dot{y} = v * sin(\theta) \tag{5}$$

$$\dot{\theta} = \omega \tag{6}$$



(a) Velocity of a mobile robot. (b) Representation of robot position.

Figura 20: Robot diferencial

A partir de las ecuaciones 1 y 4 y las ecuaciones 3 y 6 se obtuvo.

$$\frac{R}{2}(v_r + v_l) * cos(\theta) = v * cos(\theta) \tag{7}$$

$$\frac{R}{L}(v_r - v_l) = \omega \tag{8}$$

Al manipular 7 y 8 se obtuvo.

$$v_r + v_l = \frac{2v}{R} \tag{9}$$

$$v_r - v_l = \frac{\omega L}{R} \tag{10}$$

Despejando v_r de 9 y sustituyendo en 10 obtenemos

$$\frac{2v}{R} - v_l - v_l = \frac{\omega L}{R} \tag{11}$$

Ahora despejando para v_l de 11, se obtiene 12 y sustituyendo 12 en 9.

$$\frac{2v}{2R} - \frac{\omega L}{2R} = v_l \tag{12}$$

$$v_r + \frac{2v}{2R} - \frac{\omega L}{2R} = \frac{2v}{R} \tag{13}$$

Finalmente se obtuvo las ecuaciones que nos definen la velocidad v_r y v_l que corresponden a la velocidad de la llanta derecha y a la velocidad de la llanta izquierda respectivamente.

$$= \frac{2v + \omega L}{2R} \tag{14}$$

$$v_l = \frac{2v - \omega L}{2R} \tag{15}$$

Gracias a que se definió la velocidad en cada llanta del robot diferencial, se pudo encontrar la velocidad promedio o la velocidad del centro de masa del robot la cual, en el modelo diferencial, se ubica en el medio de los dos motores (Figura 21) la cual corresponde al promedio de ambas velocidades.

$$v = \frac{\frac{2v+\omega L}{2R} + \frac{2v-\omega L}{2R}}{2} = \frac{v_r + v_l}{2} \tag{16}$$

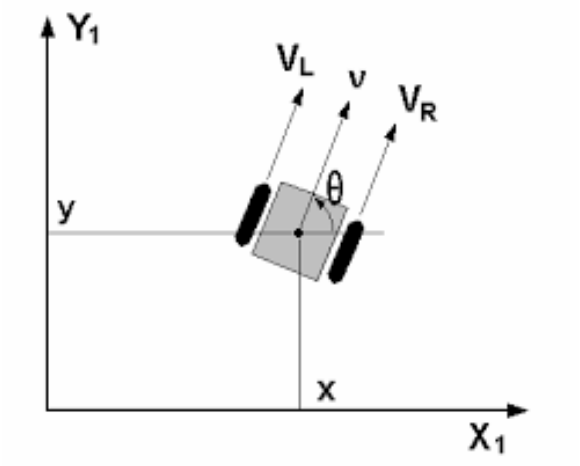


Figura 21: Velocidad del centro de masa del robot diferencial

8.2. Implementación física del modelo diferencial

Implementación física del modelo diferencial - Primera prueba

Todas las implementaciones físicas difieren de las simulaciones. Al momento de utilizar los prototipos físicos, se encontraron problemas ya que no se tenían condiciones ideales de trabajo cómo en las simulaciones. En su mayoría, los problemas se deben a agentes externos, la fabricación de los componentes utilizados o incluso a las condiciones del lugar de trabajo.

Entre los problemas que se presentaron podemos mencionar el deslizamiento de las llantas, las imperfecciones del terreno, el peso no uniformemente distribuido, la fabricación de las llantas ya que no son exactamente iguales, la eficiencia de cada motor y la velocidad que entrega cada uno con respecto al voltaje de alimentación, entre otros factores.

Implementación física del modelo diferencial - Segunda prueba

Para solucionar parte del problema de implementar físicamente el modelo diferencial del robot, se debe de contar con algún dispositivo, sensor u observador que permitan tener un feedback de lo que está pasando en el sistema, en nuestro caso el vehículo, y así poder corregirlo de ser necesario ya que el sistema asume que todo está funcionando correctamente cómo en la simulación, pero la realidad es que este presenta un comportamiento diferente al esperado.

Para esto se utilizó la odometría la cual define la posición/ubicación de un vehículo con ruedas a partir de diferentes sensores.

Se implementó la odometría a partir de encoders ópticos los cuales proporcionaron la cantidad de vueltas que ha girado la llanta a partir de ticks. Para este trabajo se utilizó el Módulo F249 y unos discos ranurados colocados en las ruedas. La serie de pulsos generados por el disco ranurado en conjunto al módulo F249 se pueden interpretar cómo ticks. Al conocer las dimensiones físicas de las ruedas del robot móvil, la cantidad de agujeros con los que cuenta el disco ranurado y el tiempo transcurrido entre cada pulso o tick, es posible determinar la velocidad angular de la llanta.

Toda esta información que se obtuvo de los encoders se utilizó como feedback del sistema y a partir de este se obtuvo el porcentaje de error de lo deseado con lo obtenido. Este porcentaje de error ayudó a poder saber que tanto se tenía que corregir la velocidad de cada motor para obtener el resultado deseado. Por último, se hizo que el sistema reaccionara ante el error y lo corrijera autónomamente.

CAPÍTULO 9

Implementación del sistema de localización en tiempo real

Para implementar el sistema de localización en tiempo real o también conocido cómo *RTLS* por sus siglas en ingles de "*Real time location sistem*"se utilizaron los módulos DWM1001-DEV fabricados por *DECAWAVE* actualmente conocido como *Qorvo*. El *RTLS* está compuesto por tres módulos que cumplen la función de ancla o *Anchor* y un cuarto módulo llamado *TAG*.

9.1. Modelo 3D - Bases para DWM1001 Anchors

Para formar el *RTLS* fue necesario perimetrar el área de trabajo con los *Anchors* para poder triangular la posición del TAG adentro de la misma. Como se necesitó colocar a los *Anchors* en diferentes lugares fue necesario alimentarlos por medio de baterías.

Debido a que se necesitó que los *Anchors* se mantuvieran funcionando durante largos periodos de tiempo, y que estos deben de ser alimentados con 5v, se utilizaron dos baterías BRC-18650, 3.7v 4200mAh para alimentar cada módulo. Sin embargo, estas baterías al colocarlas en serie suman un total de 7.4v. Debido a esto se requirió de un circuito de alimentación que entregara únicamente 5v.

El circuito y PCB diseñado para este propósito los realizó el Ing. Jonathan Saens en los prototipos anteriores [1] por lo que se utilizaron los mismos circuitos.

Como mejora para el *RTLS* se diseñó una base que sostuviera el porta baterías, el PCB de alimentación y al DWM1001-Dev para que estos módulos no se coloquen en el suelo y así tener un mejor cuidado con los mismos.

Se diseñó una base utilizando el programa AutoDesck Inventor en la cual se colocaron los tres elementos que conforman al *Anchor* (Porta baterías, PCB de alimentación, DWM1001-Dev). El diseño se realizó lo más compacto posible (Figura 22). Se utilizó madera MDF debido a la sencilles del diseño y el uso que se le daría.

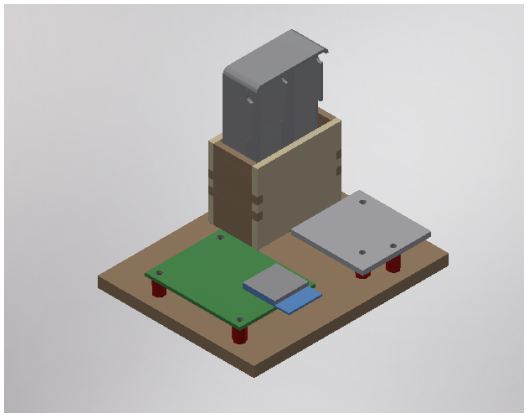


Figura 22: Modelo 3D - Bases para DWM1001 Anchors

9.2. Hardware del sistema - Configuración

Para realizar las pruebas iniciales se utilizó únicamente 3 *anchor* y un *TAG* situado en el robot móvil. Los tags se colocaron a una distancia de 60cm uno del otro formando un triángulo rectángulo (Figura 24) y se configuraron en sentido antihorario partiendo desde el *anchor* 1 tal y como lo especifica el fabricante [39].

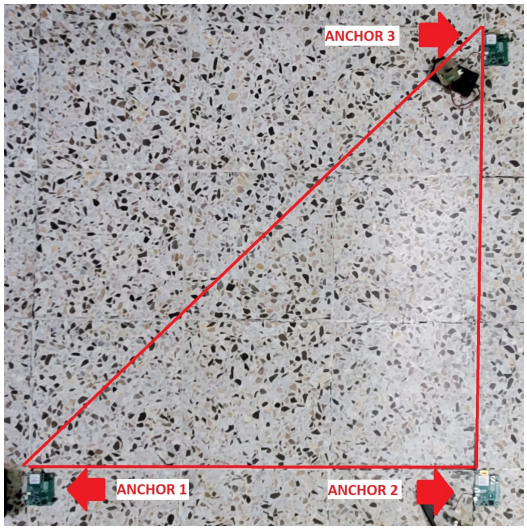


Figura 23: Configuración física de prueba de los DWM1001 como anchor

9.3. Software del sistema - Configuración de los DWM1001 usando la aplicación DRTLS

Para la configuración de los DWM1001 se utilizó la aplicación DRTLS de *DECAWAVE* [39]. En la aplicación se creó una "red" llamada "Prueba" donde se agregaron los dispositivos que se utilizaron para el *RTLS* (Figura 24).

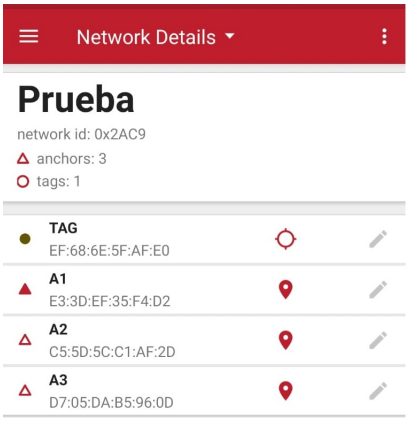


Figura 24: Red de prueba, DRTLS app

Una vez agregados los dispositivos a la red, se nombraron y configuraron los *anchors* cómo "A#" y al *TAG* se le nombró "*TAG*". Se configuró el anchor A1 como el inicializador, y los otros dos *anchors* se les debe de colocar su posición relativa a A1 en el mismo orden que se encuentran ubicados físicamente. (Figura 25)

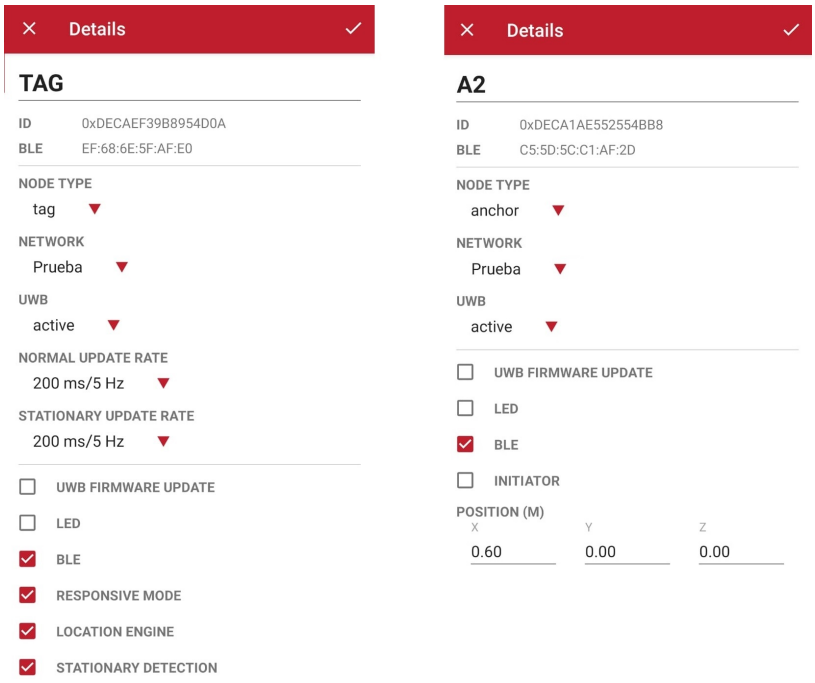


Figura 25: Red de prueba, DRTLS app

Luego se observó en la pestaña de “GRID” dentro de la aplicación, la configuración de *anchors* y *TAG* en tiempo real (Figura 26).

Figura 26: Grid de la red de prueba, DRTLS app

Esta configuración debe de realizarse cada vez que se utilizará el *RTLS* y se coloquen los *anchors* a una diferente distancia a la configurada anteriormente.

Como dato importante también se menciona que esta aplicación de DRTLS se encuentra únicamente disponible para dispositivos con un sistema operativo Android, y es imprescindible que el dispositivo que se utilice para la configuración de la red y los DWM1001 tenga conectividad bluethood.

9.4. Lectura de la posición del TAG en la terminal Tera Term

Para interactuar con el DWM1001 configurado cómo *TAG* desde el computador, se utilizó el programa “Tera Term” el cual es un emulador de terminal de código abierto. Este nos permite establecer conexiones seriales con nuestro computador. Para poder comunicarse con el DWM1001 se inició la comunicación serial con un Baud rate de 115200.

Existen múltiples comandos que se pueden utilizar para comunicase con el DWM1001. Estos se pueden encontrar de manera sencilla en el documento *"MDEK1001 System User Manual"* ubicado en la carpeta *"DWM1001, DW10001-DEV and MDEK1001 Documents, Source Code, Android Application and Firmware Image"* [39]. O de manera más detallada en el documento *"DWM1001-API-Guide"*.

Primero se debe de colocar al DWM1001 en "Shell Mode", esto se logró presionando doble enter en menos de un segundo de separación entre cada enter. Luego para observar de manera periódica la posición de los *anchor*, la magnitud del vector que une al anchor con el tag, la posición del tag (valores X, Y, Z) y un factor de calidad de la medición, se utilizó el comando ***"les"***. Este comando regresa los valores en milímetros partiendo desde la ubicación del *anchor* 1 (el origen) hacia el *TAG* (Figura 27).

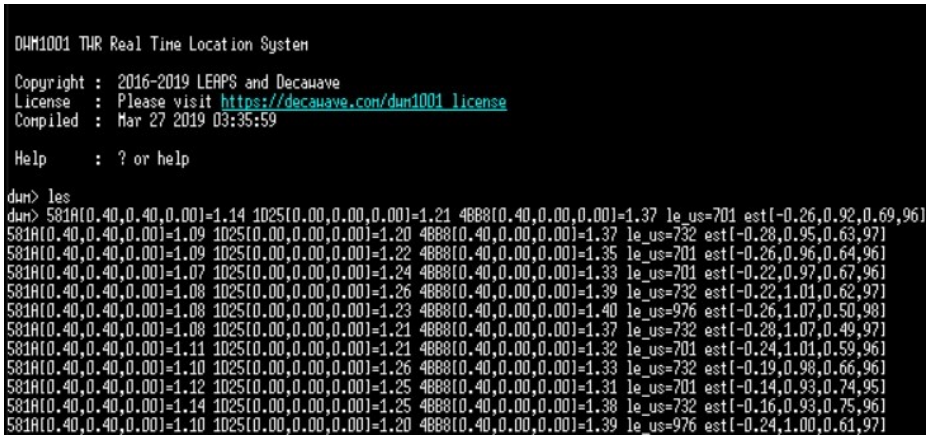


Figura 27: Datos obtenidos al utilizar comando "les" en terminal Tera Term

Si se desea observar una única medición de la posición del *TAG* (valores X, Y, Z) con respecto al *anchor* 1 (el origen), se utiliza el comando ***"apg"*** . (Figura 28).

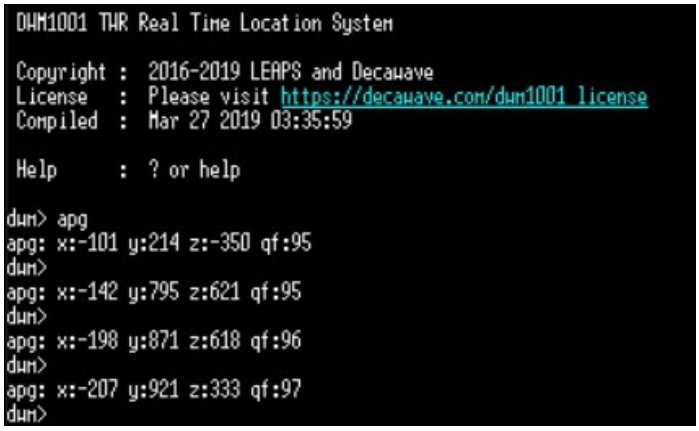


Figura 28: Datos obtenidos al utilizar comando "apg" en terminal Tera Term

9.5. Comunicación del DWM1001 con el microcontrolador ATMega2560

Otras de las maneras que existen para comunicarse con el DWM1001, es el formato de comunicación *API* (por sus siglas en ingles de *Application Programming Interface*) los cuales permiten que dos dispositivos de diferentes servicios se comuniquen con otros sin necesidad de saber cómo están implementados. Este formato establece una comunicación sencilla en la cual se le envía un dato y el dispositivo receptor envía una respuesta predefinida a ese comando recibido. La comunicación entre el TAG y el microcontrolador se realiza por medio de comunicación serial y la comunicación entre los *anchors* y el *TAG* se realiza por medio de banda ultra ancha.

Para comunicarse con el DWM1001 se utilizó el comando API "20" al cual el DWM responde con una cadena de 18 bytes (Figura 29). Los primeros 5 bytes que devuelve son

para validar el correcto envío de la posición del *TAG*. Los siguientes cuatro son la posición en X, luego cuatro para la posición en Y, seguido de 4 bytes para la posición en Z y por último un único byte para la calidad de la medición.

El primer byte indica que tipo de dato es, si este es de lectura o escritura. El segundo byte indica el largo del dato que se manejará. El tercer byte es validación o de error, este indica si existe un error en la comunicación o con los datos. El cuarto byte indica nuevamente que tipo de dato es (escritura o lectura) y por último se tiene el tamaño nuevamente antes de empezar con la lectura de la posición.

TLV response					
Type	Length	Value	Type	Length	Value
		err_code			Position
					32 bit value in little endian is x coordinate in millimeters / 32 bit value in little endian is y coordinate in millimeters / 32 bit value in little endian is z coordinate in millimeters / 8 bit value is quality factor in percents (0-100)
0x40	0x01	0x00	0x41	0x0D	0x79 0x00 0x00 0x00 0x32 0x00 0x00 0x00 0xfb 0x00 0x00 0x00 0x64

Figura 29: Cadena de 18 bytes que responde el DWM1001 al código API “20”

Al leer los datos de posición para cada dimensión (x, y, z) hay que considerar que se está recibiendo un array de 32 bits en formato *little indian*, por lo que fue necesario armar de manera correcta la posición recibida al juntar los 4 bytes que la componen. Este arreglo y la manipulación de datos se realizó por medio de software en el microcontrolador. Esto se realizó con cada una de las tres dimensiones. Cómo resultado de ordenar la cadena de datos se obtuvieron las coordenadas X, Y y Z del *TAG* expresada en milímetros.

Para verificar que datos se están recibiendo a través del puerto serial se utilizó el programa “HERCULES”, el cual permite abrir una comunicación serial con cualquier puerto COM de la computadora, escribir y recibir datos de manera sencilla y poder ver que datos son los que se están manejado a través de ese puerto.

Para las primeras pruebas se utilizaron dos placas de Arduino UNO. Uno se programó con un código que manda los valores de “2“ “0” cada 50 milisegundos y el otro se dejó en blanco. El pin de “TX“ (transferencia de datos) del primer Arduino va conectado al RX del DWM1001 (*TAG*) y el pin “TX“ del DWM1001 va conectado al “RX“ del segundo Arduino. Así con esta configuración se aseguró que no hayan colisiones en la transmisión de datos y se observó lo que el DWM1001 estaba enviando abriendo el puerto del segundo Arduino en la terminal de Hercules. (Figura 30)

Una vez identificada la cadena de bytes recibidos, se hizo la lectura de los valores en un microcontrolador Arduino MEGA. Este al recibir los valores se programó para que los desplegara en la terminal serial. Al verificar que la comunicación entre ambos dispositivos (Arduino mega y DWM1001) era correcta, se programó un apartado para verificación de la transmisión de datos, donde se compararon los cinco primeros bytes enviados por el DWM1001 los cuales indicaron la correcta comunicación y si no hubo fallas ni pérdida de datos (Figura 32).

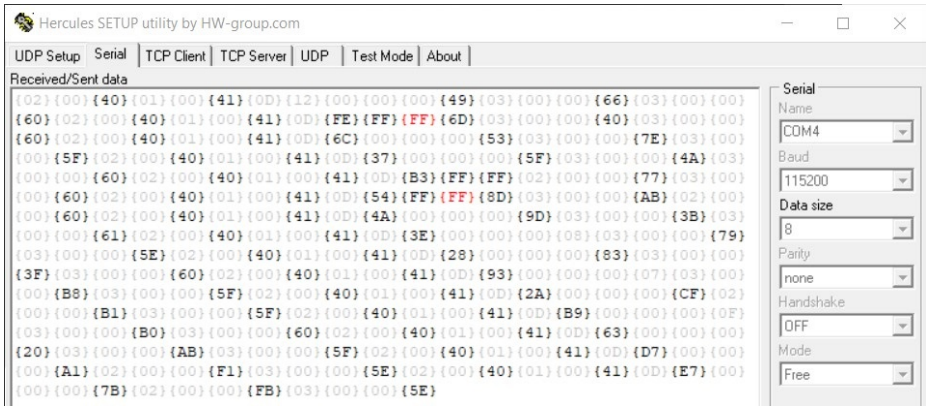


Figura 30: Respuesta del DWM1001 al "20" visto con Hercules

Una vez recibidos los bytes específicos para los primeros cinco datos, se leyeron los bytes de posición de X, Y, Z y QL (calidad). Estos datos se ordenaron y almacenaron en un array para cada dimensión, considerando el tema del *little indian* al momento de guardarlos en los arrays. Para verificar que sí se estaban armando de manera correcta las posiciones, se desplegaron los valores de X, Y, Z y QL en el monitor serial de Arduino. Una vez seguros de que se armaron de manera correcta las posiciones, ya se estaba listos para manipular esta información (Figura 31).

```
if (lectura64 == 0x40) {
  while (!Serial.available()) {}
  lectural = Serial.read();

  if (lectural == 0x01) {
    while (!Serial.available()) {}
    lectura0 = Serial.read();

    if (lectura0 == 0x00) {
      while (!Serial.available()) {}
      lectura65 = Serial.read();

      if (lectura65 == 0x41) {
        while (!Serial.available()) {}
        lectural3 = Serial.read();

        if (lectural3 == 0x0D) {
          Serial.readBytes(bufer, 13);
        }
      }
    }
  }
}

X = (bufer[3] << 24) + (bufer[2] << 16) + (bufer[1] << 8) + (bufer[0]);
Y = (bufer[7] << 24) + (bufer[6] << 16) + (bufer[5] << 8) + (bufer[4]);
Z = (bufer[11] << 24) + (bufer[10] << 16) + (bufer[9] << 8) + (bufer[8]);
Q = bufer[12];
```

Figura 31: Código de verificación y formación de coordenadas en arduino

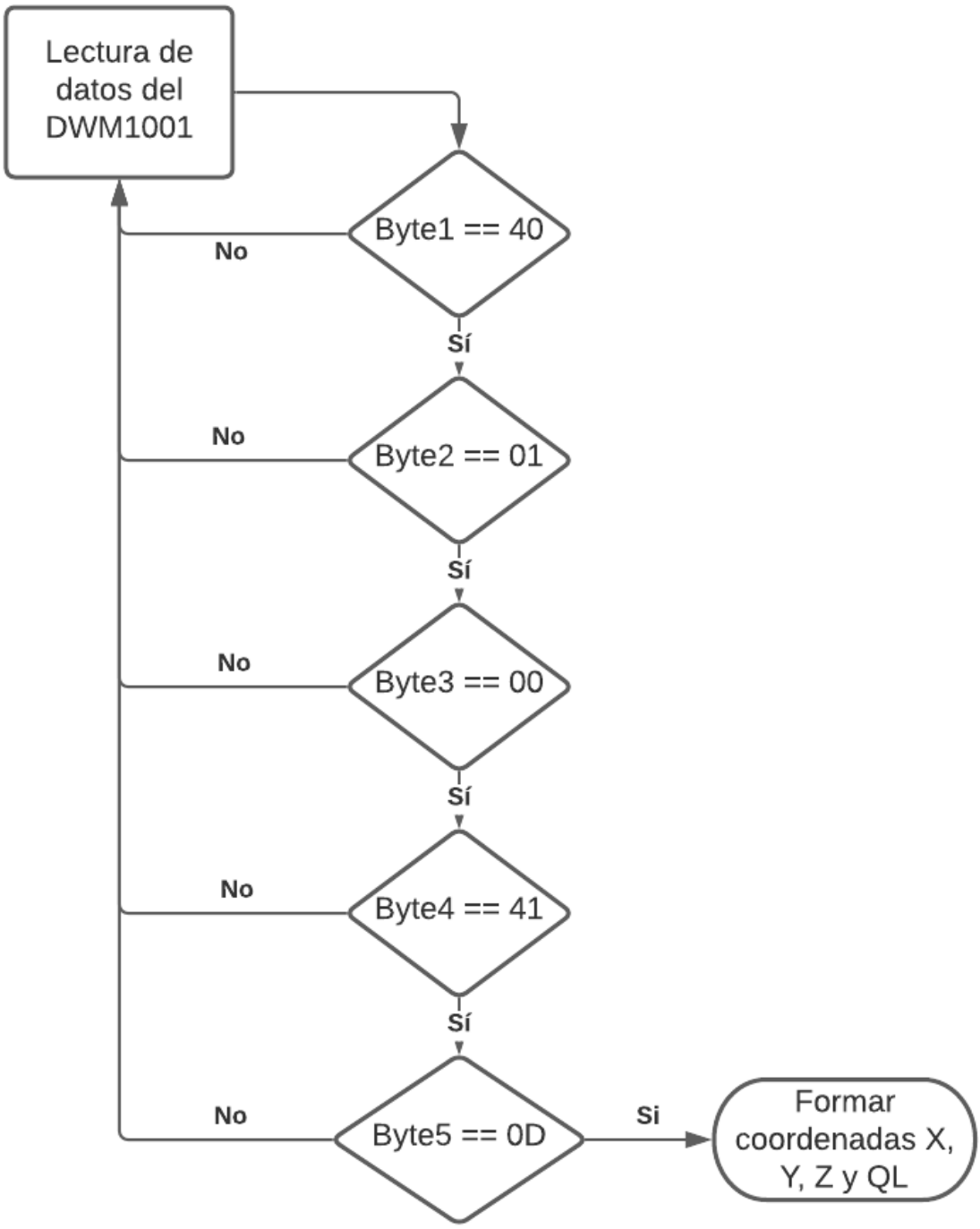


Figura 32: Diagrama de flujo función de verificación y lectura de datos

CAPÍTULO 10

Recopilación y manipulación de posición obtenidos del sistema de localización en tiempo real

10.1. Recopilación de datos de posición recibidos

Así cómo los sistemas GPS tiene un rango de exactitud y precisión de la lectura de posición, el *RTLS* también tiene un rango de exactitud y precisión de lecturas de posición. Este a pesar de no ser milimétricamente exacto, tiene un menor rango de medición de posición que los sistemas GPS normales actuales. El fabricante de los módulos DWM1001, actualmente conocido como Qorvo, asegura que un sistema de localización en tiempo real realizado con sus dispositivos tiene un rango de error en la lectura de la posición de hasta 10 centímetros cómo máximo [39].

Considerando el rango de 10 centímetros, se tomaron mediciones de posición del *TAG* dejándolo estático en un lugar mientras se observaba la información en el monitor serial de Arduino. Con esto se observó la imprecisión de las lecturas de posición. A pesar de que el *TAG* se encontraba estático, este presentaba saltos de posición en los ejes X, Y y Z. Estos saltos incluso llegaron a superar los 10 centímetros entre la posición actual y la registrada por el *TAG*.

El fabricante menciona que estas mediciones pueden ser más precisas si el área perimetrada por los *anchors* es extensa, por lo que se decidió volver a tomar las medidas de posición separando más los *anchors* los unos de los otros ampliando el área perimetrada. Aun así, se obtuvieron bastantes saltos en la posición del *TAG* en los tres ejes coordenados.

10.2. Solución 1 - Promedio de las mediciones de posición recibidas

Como solución al problema de la variación de datos de posición obtenidos del *RTLS* se realizó un promedio de los mismos. Con el promdio se disminuyó el rango de dispersión en los saltos de posición. Se obtuvo una representación significativa de los valores recibidos. Gracias a esto, a pesar de recibir un valor muy por afuera de la media aritmética, el valor o variable de control, en este caso la posición, no cambiaba de manera abrupta, y se mantenía entre un rango menor al verdadero (Figura 33).

Tiempo	X	average X	Y	average Y	Z	averageZ
1	556	545	765	792	-226	-217
2	556	546	765	791	-226	-217
3	556	546	765	791	-226	-218
4	556	546	765	790	-226	-218
5	556	547	765	790	-226	-218
6	556	547	765	789	-226	-218
7	556	547	765	789	-226	-219
8	549	547	781	788	-226	-219
9	549	547	781	788	-226	-219
10	549	548	781	788	-226	-220
11	549	548	781	788	-226	-220
12	549	548	781	787	-226	-220
13	549	548	781	787	-226	-221
14	549	548	781	787	-226	-221
15	549	548	781	786	-226	-221
16	549	548	781	786	-226	-222
17	549	549	781	786	-226	-222

Figura 33: Ejemplo de la posición en mm del TAG obtenida al encontrarse estático

Para esto se creó una función llamada “promedio“ (Figura 34) la cual como primer argumento recibe el valor de la posición que se quiere agregar al promedio y como segundo argumento se especifica a cuál vector de promedio se desea agregar. Esto quiere decir, a cuál de los tres ejes coordenados pertenece el valor del primer argumento.

Al especificar en cuál vector (X, Y, Z o Q) queremos almacenar el valor, la función se encarga de leer los valores actuales dentro del vector, luego hace un desplazamiento de todos los valores de la función en una posición desechando así el valor más antiguo del vector y colocando el nuevo valor de posición en la primera posición del vector. Luego de haber agregado el nuevo valor de la posición, vuelve a guardar el vector entero. Una vez guardado el vector, se obtiene el promedio sumando todos los valores del vector y se divide el resultado entre la cantidad de valores sumados. Este resultado es el valor que devuelve la función “promedio".

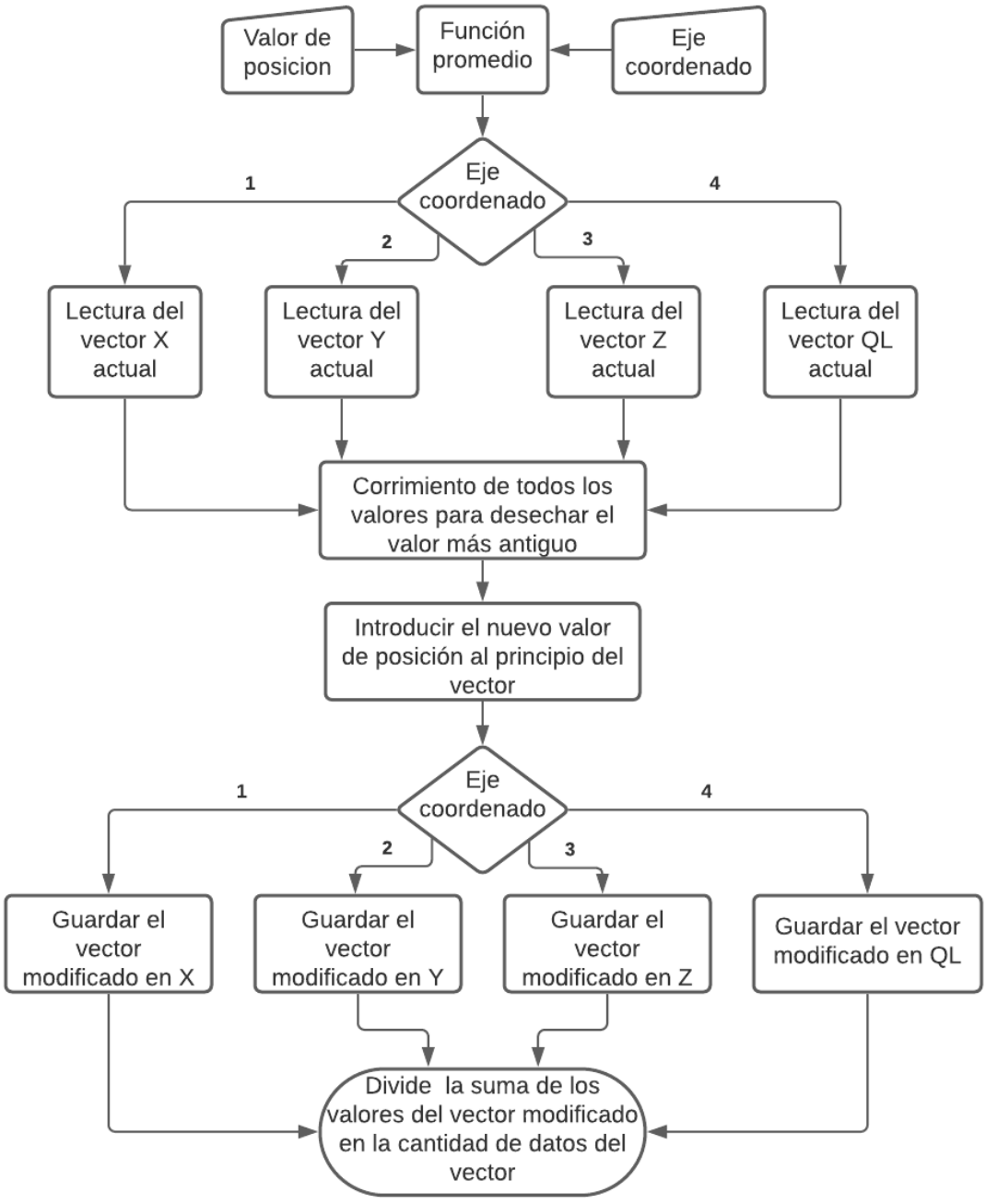


Figura 34: Diagrama de la función promedio

10.2.1. Selección del tamaño del vector para formar el promedio

Cómo la recepción de datos proveniente del DWM1001 - *TAG* es muy rápida, se pudo tomar un número alto para formar el vector de promedio para cada eje coordenado y el factor de calidad. De esta manera se pudo disminuir el rango de los valores de la posición.

Para esto se realizó una prueba de medición de posición del robot mientras este se encontraba estático. La prueba consistió en tomar diez mil (10,000) valores de la posición del robot además del resultado de la función promedio al introducir el valor respectivo de la posición para cada eje (X, Y, Z). Con esto se observó y comparó la posición del robot con y sin promedio.

Para observar de mejor manera estos resultados, se graficó el valor de X, Y y Z en conjunto al promedio para cada eje (X, Y, Z) en función del número de muestra. Con esto se observó gráficamente cómo la función promedio entrega un valor más lineal y preciso de la posición del robot a comparación de la obtenida directamente del *TAG* (Figura 35).

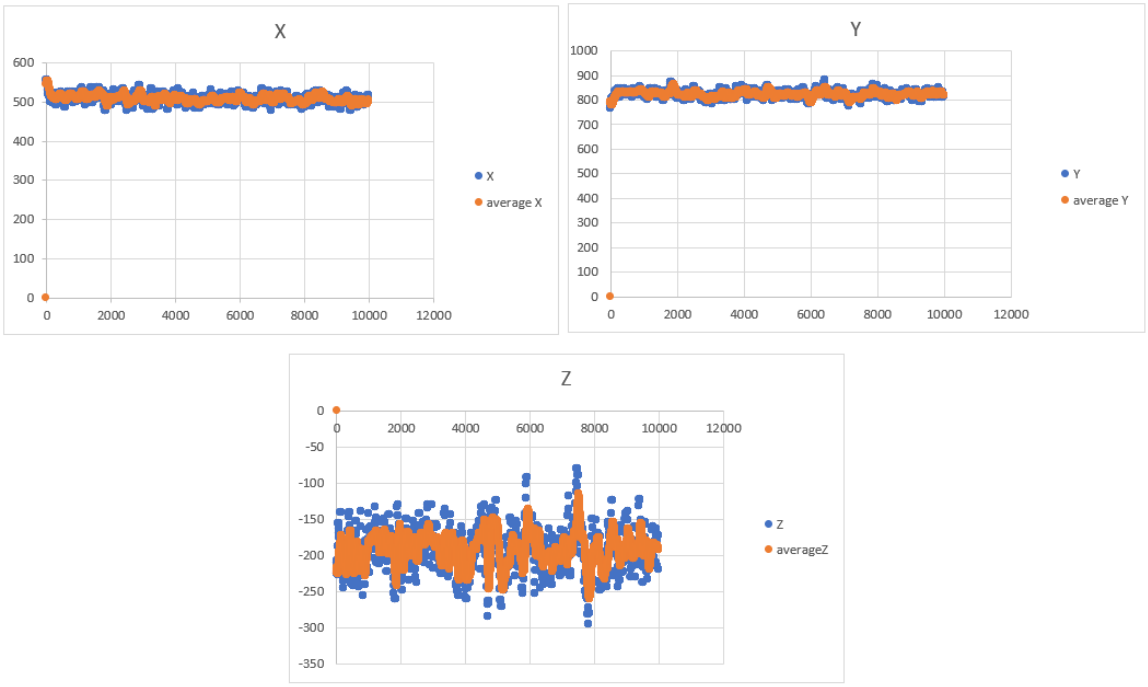


Figura 35: Resultado gráfico de posición del robot (estado estático) valor real y valor promedio

Este primer resultado gráfico se realizó con un tamaño del vector promedio de 150 unidades y se observó que aún se presentaban algunos picos indeseables. Al observar esto, se realizaron la misma prueba con tamaños del vector promedio de 100, 200 y 300 unidades para ver si variaba el resultado del promedio de manera significativa al tener más datos que lo conformarán.

Se realizaron 10 corridas con el robot estático para cada tamaño del vector (100, 200 y 300 unidades), cada corrida se realizó bajo las mismas condiciones que la primera prueba descrita. Se tomaron diez mil (10,000) muestras de la posición del robot y se graficó el valor de la posición obtenida por el *TAG* y el promedio ante cada valor de la posición obtenido para los tres ejes (X, Y, Z).

Cómo resultado se observaron gráficas parecidas para los tres tamaños del vector promedio (100, 200, 300 unidades). Se realizó un cuadro comparativo para cada corrida (Figura 36) con los tres ejes de coordenadas (X, Y y Z) para los 3 tamaños del vector promedio (100, 200, 300) dónde se observa:

- Tamaño máximo de la posición en el eje
- Tamaño mínimo de la posición en el eje
- Tamaño máximo de la posición en el eje utilizando la función promedio
- Tamaño mínimo de la posición en el eje utilizando la función promedio
- La diferencia entre el máximo y el mínimo de la posición (rango de posición)
- La diferencia entre el máximo y el mínimo de la posición (rango de la posición) utilizando la función promedio

Max X	Min X	Max Av. X	Min Av. X
541	471	534	475
Max Y	Min Y	Max Av. Y	Min Av. Y
878	770	873	773
Max Z	Min Z	Max Av. Z	Min Av. Z
-114	-290	-126	-280

Dif. X	Dif. Av
70	59
Dif. Y	Dif. Av
108	100
Dif. Z	Dif. Av
176	154

Figura 36: Ejemplo cuadro comparativo de valores de posición con y sin promedio

Con estas tablas pudimos observar que efectivamente la función promedio disminuía la variación de las mediciones, ósea, el rango de la posición. Adicional a esto, se realizó un último cuadro donde comparamos los resultados generales para cada tamaño de vector promedio para cada eje de coordenadas. Para esto se utilizaron los 10 cuadros obtenidos por cada corrida para cada tamaño de vector promedio (100, 200 y 300 unidades) y se sacó el promedio de la "diferencia entre el máximo y el mínimo de la posición (rango)" con y sin promedio, para saber un aproximado de entre cuales rangos nos encontrábamos con cada tamaño del vector promedio para los tres ejes. (Figura 37).

Al ver la tabla (Figura 37) se concluyó que si existe una gran diferencia en la posición utilizando la función promedio en comparación de cuando no se utiliza. También se observó que si hay un cambio significativo al momento de aumentar el tamaño del vector promedio. A pesar de que el tamaño del vector promedio de 300 dio los rangos más pequeños, mientras se realizaron las corridas para recopilar las mediciones de la posición, se observó una gran diferencia en el tiempo de recolección de datos. Se encontraron diferencias de hasta 3 o 4 minutos en sacar las diez mil muestras (10,000) utilizando 200 y 300 unidades en el tamaño del vector. Debido a esa diferencia de tiempo se optó por utilizar el tamaño del vector

Tamaño de vector	Promedio Diferencia X	Promedio Diferencia Av. X
100	76.500	85.300
200	87.100	59.500
300	92.600	52.800

Tamaño de vector	Promedio Diferencia Y	Promedio Diferencia Av. Y
100	95.400	116.889
200	114.900	83.100
300	117.700	64.300

Tamaño de vector	Promedio Diferencia Z	Promedio Diferencia Av. Z
100	167.600	146.400
200	183.200	128.200
300	193.200	96.100

Figura 37: Promedio de rangos obtenidos para la posición por eje coordenado utilizando tamaño de vector de 100, 200 y 300 unidades

promedio de 200 unidades ya que se consideró que este valor cumplió con ser un buen punto intermedio entre disminución del rango de los valores de posición y la velocidad de recolección y manejo de datos.

10.2.2. Pruebas de recolección de datos de posición con el robot en movimiento

Para observar y validar la recolección de datos de la posición del robot, se realizaron pruebas con tres diferentes escenarios:

1. Mientras el robot se encontraba estático.
2. Mientras el robot se desplaza en el eje X.
3. Mientras el robot se desplaza en el eje Y.

Las pruebas consistieron en recolectar diez mil (10,000) muestras de la posición del robot además del promedio para dos de los ejes coordenados (X, Y) tal y como se realizó anteriormente para definir el tamaño del vector promedio. Para estas pruebas se utilizó el vector promedio con una dimensión de 200 unidades que fue el valor establecido en las pruebas anteriores.

Esta prueba se llevó a cabo para los tres escenarios. En el caso del escenario 2 y 3 en los cuales el robot debía moverse, este se colocó sobre una bolsa plástica y se desplazó de manera uniforme lo más recto posible, siguiendo una línea guía, a lo largo del eje evaluado (Figura 38). Se utilizó una bolsa plástica ya que esta presenta menor fricción que el hule de las ruedas del robot lo que facilitó el desplazamiento. Debido a las agarraderas de la bolsa, con esta también fue más fácil manipular al robot en una dirección específica.

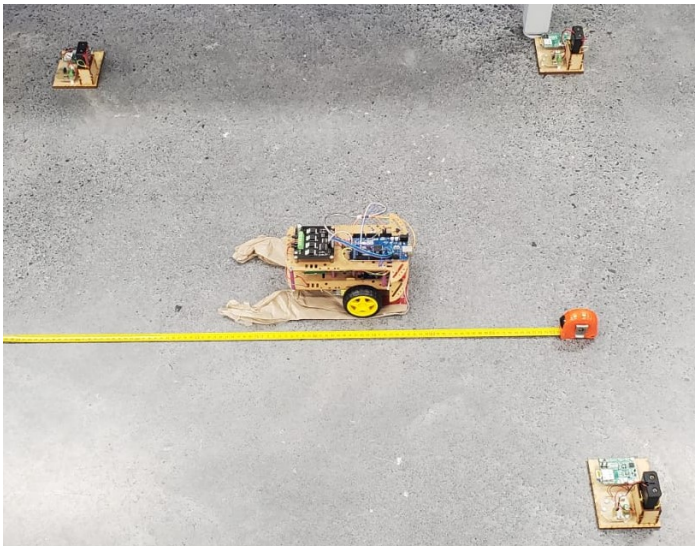


Figura 38: Manipulación del robot para escenarios en pruebas en movimiento

Para tener una muestra significativa de las mediciones, se realizaron 10 corridas por escenario. De igual manera que en las pruebas anteriores, se graficaron y promediaron los resultados.

Muestra de resultados de posición del robot estático

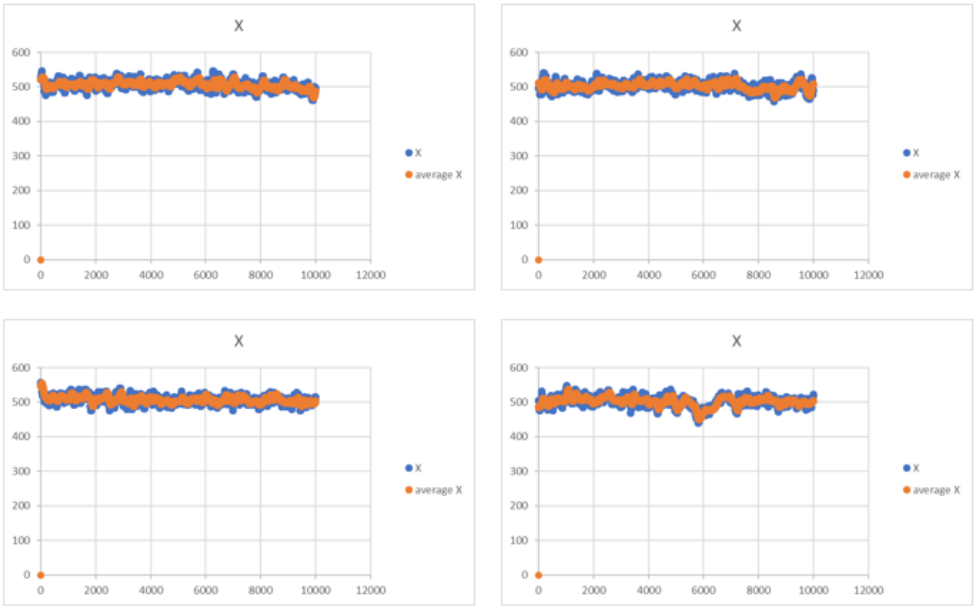


Figura 39: Gráficas de posición del robot en eje X con el robot estático

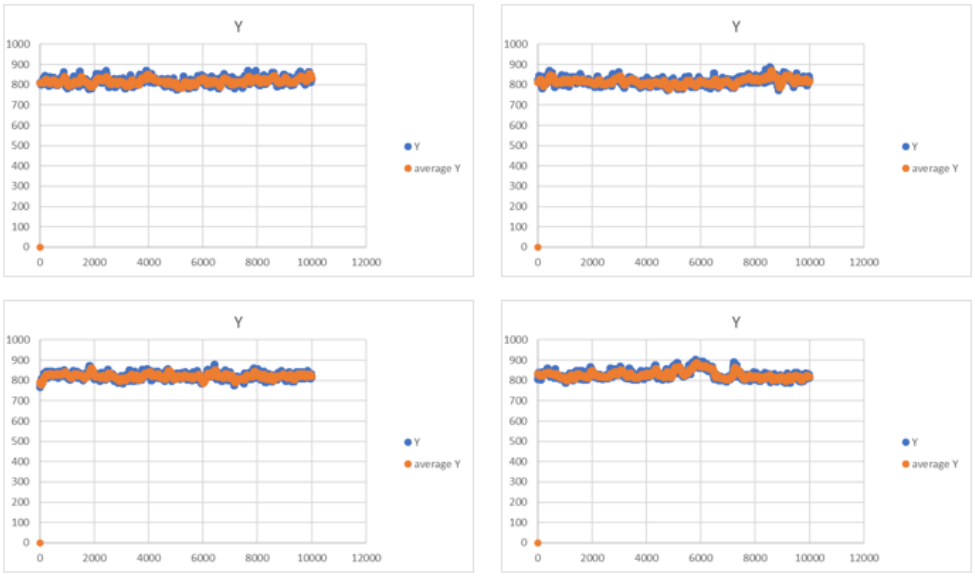


Figura 40: Gráficas de posición del robot en eje Y con el robot estático

VALORES EN CENTÍMETROS

Tamaño de vector promedio	Promedio Diferencia X	Promedio Diferencia Av. X
200	8.71	5.95

Tamaño de vector promedio	Promedio Diferencia Y	Promedio Diferencia Av. Y
200	11.49	8.31

Tamaño de vector promedio	Promedio Diferencia Z	Promedio Diferencia Av. Z
200	18.32	12.82

Figura 41: Resultado promedio de posiciones con robot estático

Muestra de resultados de posición con movimiento del robot en el eje X

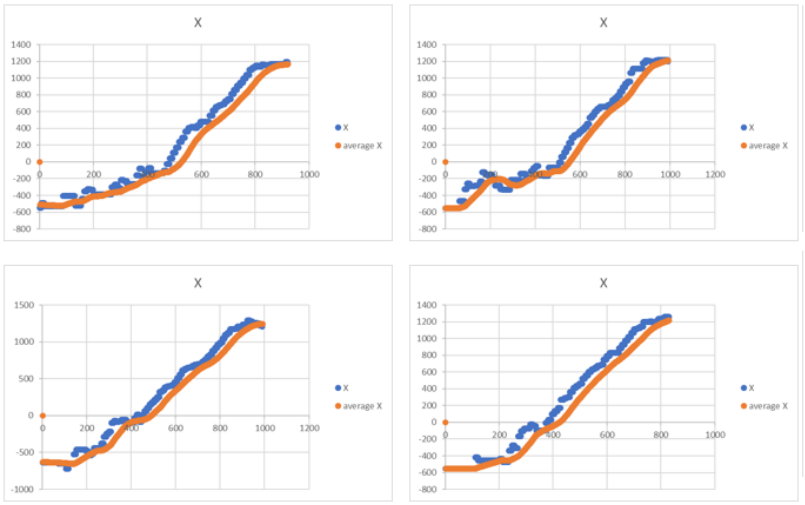


Figura 42: Gráficas de posición del robot en eje X con el robot en movimiento en el eje X

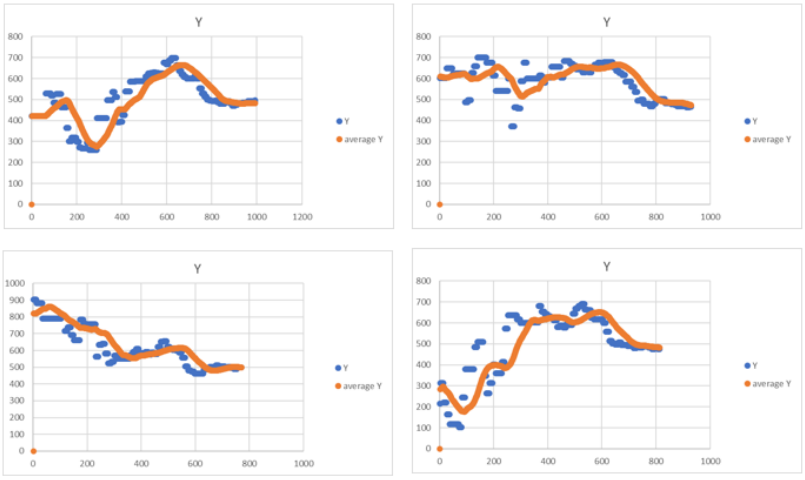


Figura 43: Gráficas de posición del robot en eje Y con el robot en movimiento en el eje X

VALORES EN CENTÍMETROS

Tamaño vector promedio	Promedio Diferencia X	Promedio Diferencia Av. X
200	166.04	159.33

Tamaño vector promedio	Promedio Diferencia Y	Promedio Diferencia Av. Y
200	49.64	38.31

Tamaño vector promedio	Promedio Diferencia Z	Promedio Diferencia Av. Z
200	56.1	45.67

Figura 44: Resultado promedio de posiciones con el robot en movimiento en el eje X

Muestra de resultados de posición con movimiento del robot en el eje Y

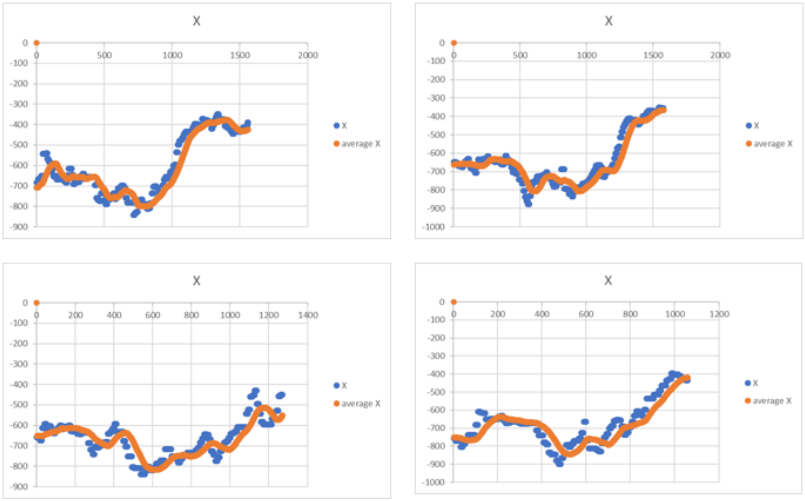


Figura 45: Gráficas de posición del robot en eje X con el robot en movimiento en el eje Y

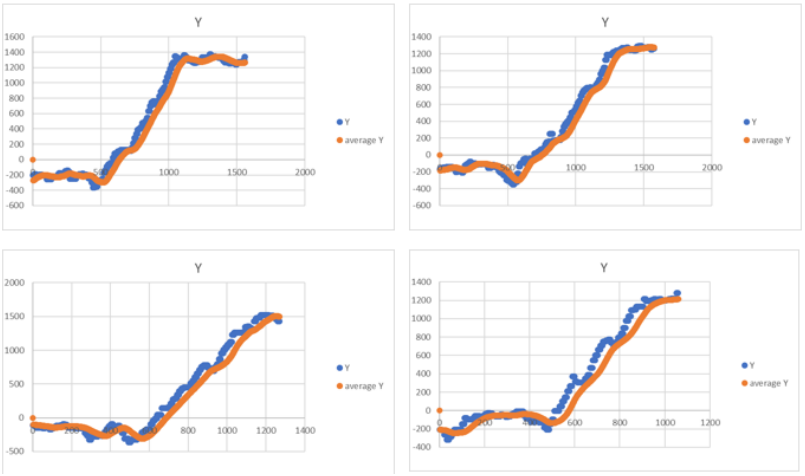


Figura 46: Gráficas de posición del robot en eje Y con el robot en movimiento en el eje Y

VALORES EN CENTÍMETROS

Tamaño vector promedio	Promedio Diferencia X	Promedio Diferencia Av. X
200	47.17	39.67

Tamaño vector promedio	Promedio Diferencia Y	Promedio Diferencia Av. Y
200	157.05	149.82

Tamaño vector promedio	Promedio Diferencia Z	Promedio Diferencia Av. Z
200	32.9	25.62

Figura 47: Resultado promedio de posiciones con el robot en movimiento en el eje Y

Al gráficar los resultados del movimiento del robot, se obtuvieron tres conclusiones importantes.

1. La función promedio sí reduce la variación y dispersión de las mediciones de posición, entregando un resultado más limpio con el cual poder trabajar.

2. Al manipular al robot únicamente en un eje coordenado, el otro eje en el que no se manipuló el robot debería de permanecer constante o por lo menos mantener una línea lo suficientemente recta. Se observó que este no es el caso, por lo que parece que el robot no solo se desplaza en el eje coordenado que nosotros movemos si no que este va dando saltos hacia los lados de forma muy aleatoria, lo cual es un gran problema.

3. Debido a la gran cantidad de saltos que presenta el robot, se debe implementar otro filtro para los datos de posición recibida.

10.3. Solución 2 - Descarte de posiciones mayores al rango establecido

Como el problema que presentaba el robot eran los saltos de valores de la posición entre un rango muy amplio, se hizo un filtro previo a introducir el valor de la posición medida al vector promedio. Esto se realizó con el fin de evitar los saltos amplios de posición consiguiendo una menor dispersión de los valores medidos y obtener una medición más limpia. Este filtro se consideró un filtro pasa bandas, que únicamente permitirá pasar a los valores que se encuentren entre un rango establecido.

Primero se estableció el rango de valores del filtro pasa bandas. Como indica el fabricante Qorvo, las mediciones del *RTLS* pueden variar en un rango máximo de 10 centímetros de la posición real [39]. Basándose en esto, se decidió por colocar un rango de más menos 10 la posición actual, esto quiere decir, un rango de 20 centímetros.

Para realizar el código, se estableció una posición inicial del robot el eje X con respecto al *anchor* iniciador. Al conocer la posición inicial teórica del robot, se pudieron crear límites permitidos. En este caso, se colocó la posición inicial teórica del robot en 40 centímetros, el límite inferior 30 centímetros y el límite superior de 50 centímetros en el eje X (Figura 48). Lo que deja un rango de valores de posición en X permitido de 30 centímetros a 50 centímetros para que el valor sea aceptado.

```
int LineaX = 400;                    // Valor en mm de la linea que queremos seguir
int Limite_InfX = LineaX - 100;      // Limite inferior en mm de linea
int Limite_SupX = LineaX + 100;      // Limite superior en mm de linea
```

Figura 48: Posición inicial teórica del robot, y límites superior e inferior

Cómo lo que se buscaba era filtrar los valores de posición antes de entrar al vector promedio, este filtro lee directamente el valor de la posición recibida del *RTLS* después de haber sido creada por el código de verificación y formación de coordenadas (Figura 32). Si se recibe un valor de posición X mayor al límite superior o menor al límite inferior, este valor se descarta. Al contrario, si se recibe un valor de X entre los límites superior e inferior establecidos, esa coordenada de posición del robot se aceptaba y se agrega cada valor (posición X, Y, Z y la calidad) en su respectivo vector promedio (Figura 49).

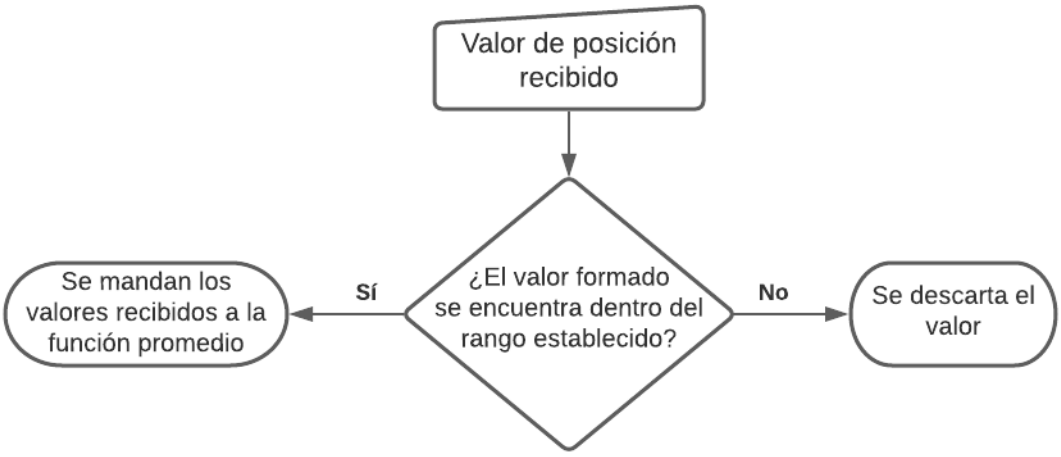


Figura 49: Diagrama del filtro pasa bandas

Para la parte del hardware, el robot se colocó a una distancia X igual o lo más exacta posible al valor de la posición inicial teórica establecida. Esto quiere decir que el robot se colocó a 40cm a la derecha de nuestro *anchor* iniciador. Esto aseguró que el robot arranca dentro de nuestro rango permitido.

10.3.1. Pruebas del nuevo filtro

Para verificar que el filtro funciona, se utilizó la terminal de arduino para desplegar el valor de la posición X actual del robot y si esta se rechaza o se agrega al promedio. Esto se realizó colocando al robot a 40centímetros de nuestro *anchor* iniciador y se dejó ahí en posición estática.

Lo que se recibió fue una serie de posiciones en X oscilantes entre 30 a 50 centímetros los cuales se agregarón al vector promedio, sin embargo, esto no fue el caso. En la terminal se observaron valores oscilantes entre rangos que superan los 10 centímetros de la posición inicial (40 centímetros), con picos de hasta 70 centímetros.

El fin del filtro fue controlar esos picos y descartar los valores que no cumplieran con el rango establecido. El robot a pesar de estar en posición estática en la posición inicial establecida, las lecturas de posiciones descartadas superaban las 30 lecturas seguidas. Se tenía previsto que se descartarían mediciones de posición; sin embargo, se esperaba que este valor no superara las 10 mediciones de posición seguidas. Sin embargo, las posiciones descartadas eran más frecuentes de lo esperado.

La cantidad de mediciones de posición del robot que se descartaban era tan grande y con tanta frecuencia que las mediciones que si eran aceptadas representan una perspectiva errónea de lo que en verdad está entregando el *RTLS* para la posición del robot. Lo que llevó a la conclusión que a pesar de los dos filtros presentados en la solución 1 y la solución 2, la posición del robot recibida por el *RTLS* no es suficiente para el sistema de control.

CAPÍTULO 11

Propuesta 2 - Recopilación y manipulación de datos de distancia para posición del robot utilizando el sistema de localización en tiempo real

11.1. Comando dwm_loc_get

Como no se obtuvieron resultados satisfactorios al realizar la lectura de posición del robot debido a distintos factores, se optó por encontrar la posición en base a distancias.

En el documento *"DWM1001-API-Guide"* ubicado en la carpeta *"DWM1001, DW10001-DEV and MDEK1001 Documents, Source Code, Android Application and Firmware Image"* [39] se presentan códigos API para manipular al DWM1001. En este documento se presenta el comando *"dwm_loc_get"* en la página 53 del PDF.

La descripción del comando indica que se obtiene la última distancia del *anchor* (hacia el *TAG*) y la posición asociada al mismo.

El código API de este comando es "0x0C" "0x00". Para utilizar este comando es necesario que el *TAG* se encuentre asociado a una red *RTLS*. Al mandar el código "0x0C" "0x00" al *TAG* se reciben una serie de bytes cómo respuesta. El primer byte indica que tipo de dato es, si este es de lectura o escritura. El segundo byte indica el largo del dato que se manejará. El tercer byte es validación o de error, este indica si existe un error en la comunicación o con los datos. El cuarto byte indica nuevamente que tipo de dato es (escritura o lectura) . El quinto byte indica nuevamente el largo de la siguiente estructura a recibir. Los siguientes 13 bytes indican la posición del anchor, al igual que en el capítulo 20. A diferencia de el comando "0x00" "0x02", aqui tenemos más byte de información. Luego de recibir la posición, en el décimo noveno byte se recibe el tipo de dato acompañado del vigésimo byte el cual especifica el largo del próximo byte. Ahora el siguiente byte (el vigésimo primero) indica el número de *Anchors* conectados en la red. Seguido a esto se reciben 2 bytes que especifica la dirección de banda Ultra ancha o dicho de otra manera, el nombre serial del *DWM1001*

al que se le está comunicando. Luego se reciben 4 bytes que indican la distancia (magnitud del vector) del anchor hacia el *TAG*. Cabe resaltar que la distancia se recibe en formato little indian, por lo que es necesario ordenarla para obtener la lectura de distancia correcta. También se recibe un byte de factor de calidad de la lectura y nuevamente 13 bytes con la lectura de posición. Dependiendo del tamaño de la red y la cantidad de *Anchors* conectada a la misma el byte de número de *Anchors* conectados varía. Al tener más de un *Anchors* conectado, se especificará el nombre del *Anchors* seguido de los 4 bytes de distancia, el de factor de calidad y por último los 13 bytes de posición y luego de esto se recibe el nombre del siguiente *Anchors* y todos los datos que lo acompañan (Figura 50).

TLV response								
Type	Length	Value	Type	Length	Value	Type	Length	Value
		err_code			position, 13 bytes			1 byte, number of distances encoded in the value
					4-byte x / 4-byte y / 4-byte z / 1-byte quality factor			
0x40	0x01	0x00	0x41	0x0D	0x4b 0x0a 0x00 0x00 0x1f 0x04 0x00 0x00 0x9c 0x0e 0x00 0x00 0x64	0x49	0x51	0x04

TLV response (residue of the frame from previous table)								
Value								
2 bytes UWB address	4-byte distance	1-byte distance quality factor	position in standard 13 byte format	...	2 bytes UWB address	4-byte distance	1-byte distance quality factor	position in standard 13 byte format
position and distance AN1				AN2, AN3	position and distance AN4			

Figura 50: Cadena de bytes que responde el DWM1001 al código API “0x0C” “0x00”

Se utilizó este comando para obtener el nombre del *Anchors* y la distancia (magnitud del vector) que hay entre el *Anchors* y el *TAG* para encontrar matemáticamente la posición del *TAG*.

11.2. Definiendo matemáticamente la posición del TAG

Para definir la posición del *TAG* utilizando el sistema de localización en tiempo real compuesto por 3 *Anchors* a partir de la recta entre el *Anchors* con el *TAG*, fue necesario definir las posiciones de los *Anchors*.

La manera más conveniente de definir el perímetro formado por los tres *Anchors* es formando un triángulo rectángulo. Se ubicó al *TAG* adentro del perímetro triangular (Figura 51). Donde A1 equivale al *Anchor1*, A2 a *Anchor2* y A3 a *Anchor3*.

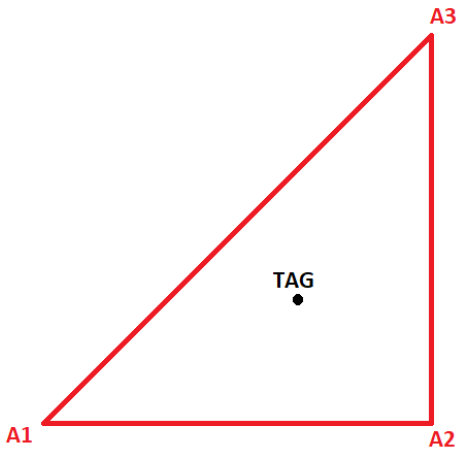


Figura 51: Sistema de localización en tiempo real ubicación

A partir de aquí se observó que el *TAG* tiene una recta correspondiente hacia cada *Anchor* (Figura 52).

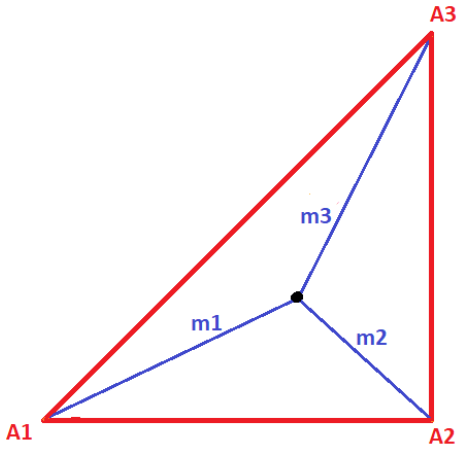


Figura 52: Rectas entre TAG y Anchors

Se observó que entre las rectas m# y las distancias A1A2, A2A3 y A1A3 se forman ángulos los cuales se dividieron en dos (Figura 53).

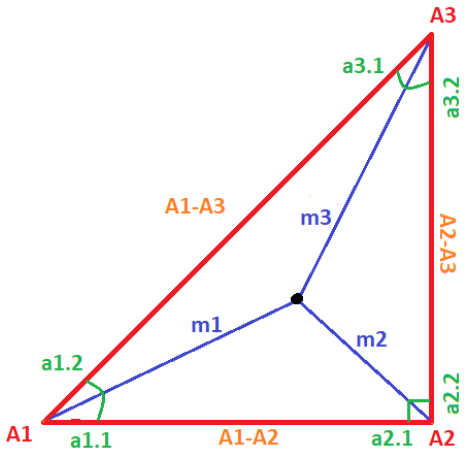


Figura 53: Ángulos formados entre las rectas y el perímetro triangular

Por la posición física de los *Anchors* se conocen las distancias A1A2, A2A3 y A1A3. Las rectas m1, m2 y m3 se obtienen a partir del comando *dwm_loc_get*. Para encontrar la posición del *TAG* a partir de las rectas m1, m2 y m3 es necesario encontrar tres ángulos distintos (a1.1, a1.2, a2.1, a2.2 y a3.1, a3.2).

Para esto se utilizó la ley de cosenos la cual se define como:

$$a^2 = b^2 + c^2 - 2bc * cos(A) \tag{17}$$

Despejando para el ángulo A se obtiene

$$cos(A) = \frac{-a^2 + b^2 + c^2}{2bc} \tag{18}$$

$$A = cos^{-1}(\frac{-a^2 + b^2 + c^2}{2bc}) \tag{19}$$

la definición de los ángulos a1.1, a1.2, a2.1, a2.2, a3.1 y a3.2 radica de la ecuación (19) previamente definida. Al ver la Figura 53 se observan tres distintos triángulos (Figura 54).

Triangulo 1 definido por los segmentos A1A3, m3, m1 y los ángulos a1.2 y a3.1.

Triangulo 2 definido por los segmentos A2A3, m3, m2 y los ángulos a2.2 y a3.2.

Triangulo 3 definido por los segmentos A1A2, m2, m1 y los ángulos a1.1 y a2.1.

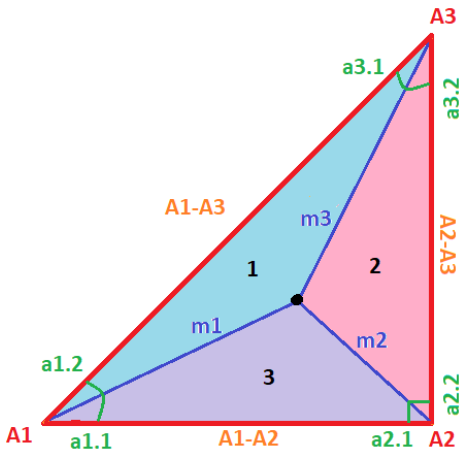


Figura 54: Triángulos formados entre las rectas de los anchors y el perímetro triangular

Al definir todos los ángulos obtenemos:

$$a1.1 = cos^{-1}(\frac{-m_2^2 + (A1A2)^2 + m_1^2}{2(A1A2)m_1}) \tag{20}$$

$$a1.2 = cos^{-1}(\frac{-m_3^2 + (A1A3)^2 + m_1^2}{2(A1A3)m_1}) \tag{21}$$

$$a2.1 = cos^{-1}(\frac{-m_1^2 + (A1A2)^2 + m_2^2}{2(A1A2)m_2}) \tag{22}$$

$$a2.2 = cos^{-1}(\frac{-m_3^2 + (A2A3)^2 + m_2^2}{2(A2A3)m_2}) \tag{23}$$

$$a3.1 = cos^{-1}(\frac{-m_1^2 + (A1A3)^2 + m_3^2}{2(A1A3)m_3}) \tag{24}$$

$$a3.2 = cos^{-1}(\frac{-m_2^2 + (A2A3)^2 + m_3^2}{2(A2A3)m_3}) \tag{25}$$

A partir del ángulo y la recta es posible definir las coordenadas X y Y de la posición actual del *TAG* partiendo desde cualquier *Anchor* utilizando las funciones de seno y coseno para un triángulo rectángulo.

Definiendo la posición del *TAG* a partir del *Anchor 1*

$$X = m_1 * \cos(\frac{a1.1 * pi}{180}) \tag{26}$$

$$Y = m_1 * sen(\frac{a1.1 * pi}{180}) \tag{27}$$

Definiendo la posición del *TAG* a partir del *Anchor 2*

$$X = A1A2 - (m_2 * cos(\frac{a2.1 * pi}{180})) \tag{28}$$

$$Y = m_2 * sen(\frac{a2.1 * pi}{180}) \tag{29}$$

Definiendo la posición del *TAG* a partir del *Anchor 3*

$$X = A1A2 - (m_3 * sin(\frac{a3.2 * pi}{180})) \tag{30}$$

$$Y = A2A3 - (m_3 * cos(\frac{a3.2 * pi}{180})) \tag{31}$$

Estas tres coordenadas para X y para Y son equivalentes una con la otra. Ya que indican la posición del *TAG* a partir del *Anchor 1* o dicho de otra manera el *Anchor inicial*

11.3. Recopilación y manipulación de datos utilizando código "0x0C" "0x00" para encontrar la posición del TAG a partir de distancias

Como se describió en la sección anterior, fue necesario primero definir el perímetro triangular que formarán los tres *Anchor* especificando la posición de cada uno y la distancia a la que se encuentran.

Para esto se crearon las variables *PosA1, PosA2 y PosA3* las cuales definen la posición de cada uno de los *Anchor* en forma de vector. En estas variables se definió la posición X y Y de cada *Anchor* utilizando al *Anchor 1* cómo el origen.

Se crearon las variables *ANC1, ANC2 y ANC3* en las cuales se guardó el nombre serial de cada *Anchor* para saber a cuál de los tres *Anchor* se refería la información recibida del sistema de localización en tiempo real.

Se crearon las variables *Dist1, Dist2 y Dist3* para guardar las magnitudes de las tres rectas recibidas del sistema de localización en tiempo real.

Para manipular el tiempo que debe de pasar sin recibir valores de los *Anchors* en el *TAG* para volver a solicitar la información. Se crearon las variables *MillisActual*, *MillisAnterior*

y *ValorMillis*. Con estas variables se creó un cronómetro con ayuda de la función millis(). Para esto primero se asignó una cantidad de millisegundos a la variable *ValorMillis* la cual indica el tiempo máximo permitido sin recibir datos en el *TAG*.

Por último, se crearon las variables *a, b, c* que representan a las rectas *A1A2, A2A3 y A1A3* respectivamente.

En el robot se colocó un boton de arranque para dar inicio a las acciones del programa. En el *LOOP* del programa se revisa cuando se presione el botón. Al presionar el boton se le envió el comando *dwm_loc_get* al *TAG* con el código "12" "0" una sola vez. También se le asignó el valor de *millis()* a las variables *MillisActual*, *MillisAnterior*. Por último, se le indicó al programa que entre al estado1 (Figura 55).

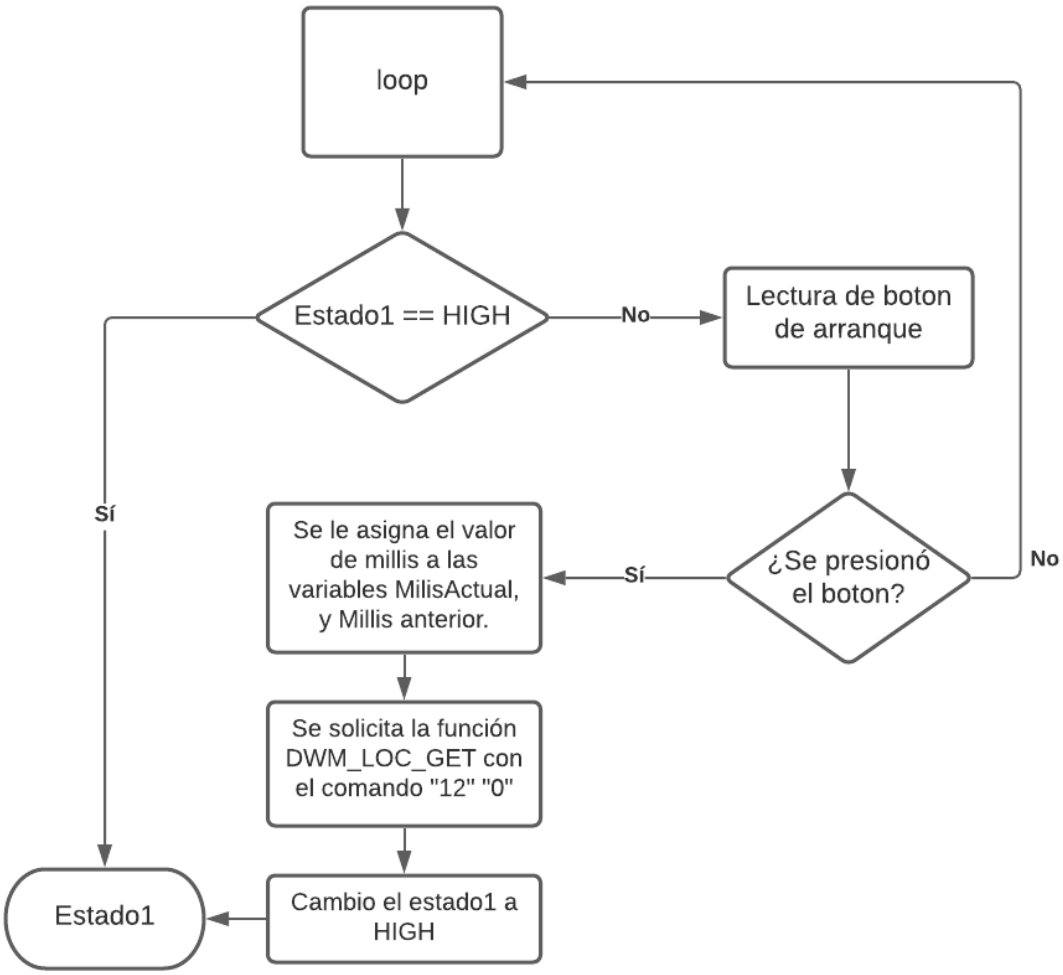


Figura 55: Diagrama del loop del programa

El estado1 está compuesto por tres secciones: Verificación de tiempo sin recepción de datos, lectura de datos y actualización del tiempo transcurrido (Figura 56).

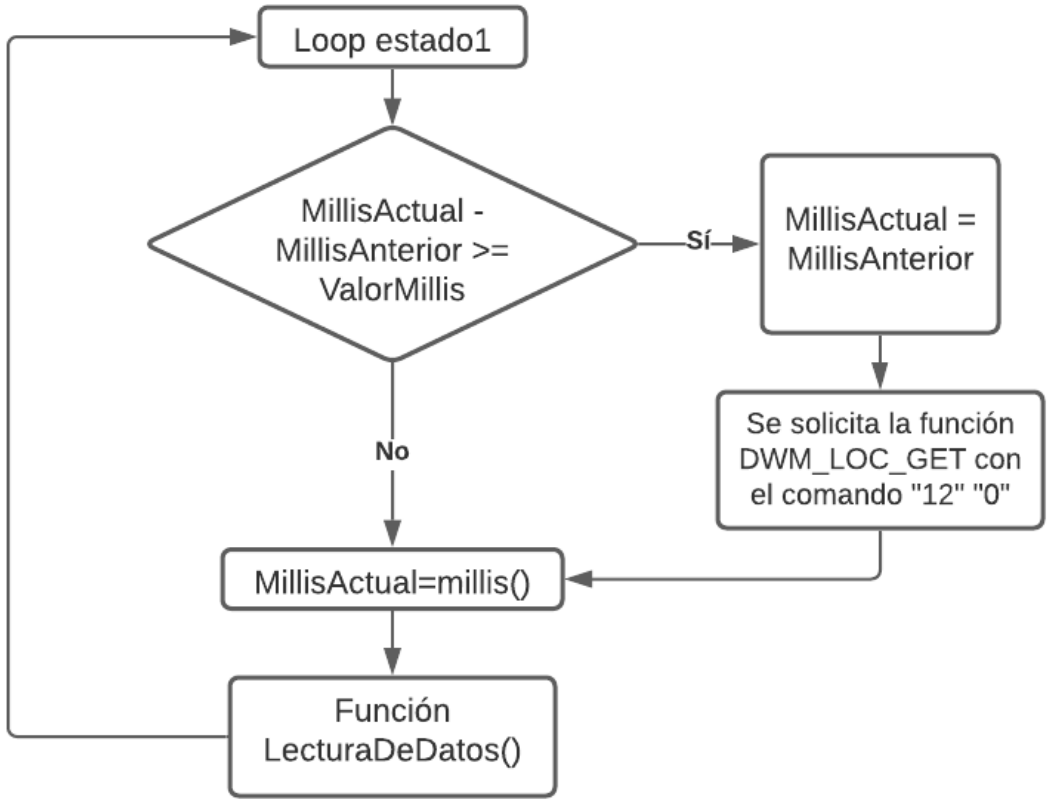


Figura 56: Diagrama del Loop del estado1

Estructura condicional para solicitar datos al RTLS periódicamente

Esta estructura condicional verifica si el valor de *MillisActual* es mayor al valor de *MillisAnterior*. Si esta diferencia entre las dos variables supera el valor de *ValorMillis*, se envía nuevamente el código "12" "0" al *TAG* y se actualiza el valor de las variables *MillisAnterior* para reiniciar el cronómetro. De esta manera se aseguró que siempre se le solicite a los *Anchors* la información de su posición y la distancia entre ellos y el *TAG* cada cierto tiempo (Figura 56).

Cada vez que se realiza la lectura de datos se actualiza el valor de la variable *MillisActual* y para verificar cuanto tiempo ha transcurrido desde la última lectura de datos realizada.

Función Lectura de datos

La función *LecturaDeDatos()* (Figura 57) realiza la lectura de los bytes de verificación y se comparan los valores recibidos para confirmar que efectivamente se recibió la información correcta de los *Anchors*. Los bytes verificados son el 19, el 20 y el 21 que corresponden al tipo de dato (lectura o escritura), tamaño del siguiente byte y al número de *Anchors* en la red respectivamente.

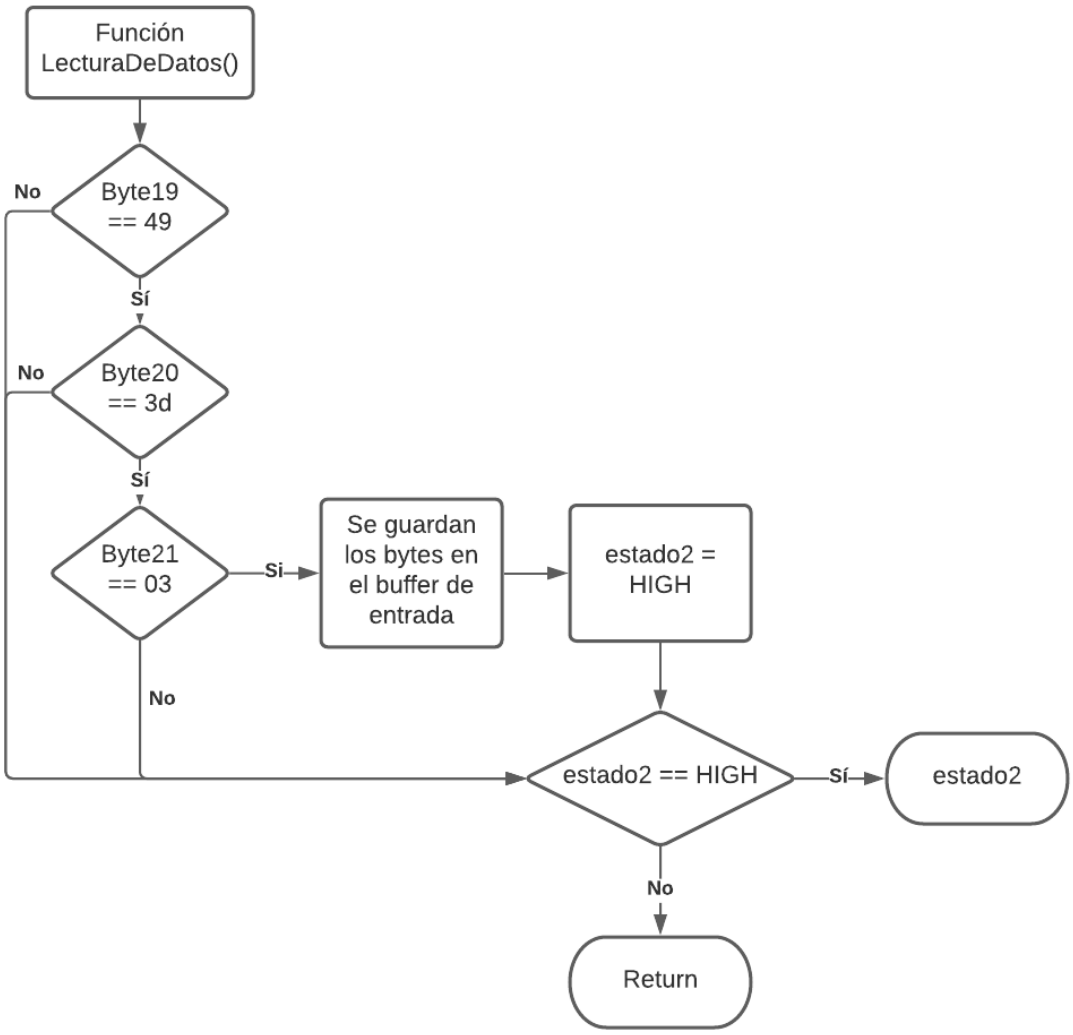


Figura 57: Diagrama de la función Lectura de datos

Si los valores recibidos son correctos, se guardan los siguientes valores recibidos en un buffer de tamaño 60 y luego se entra a la estructura condicional *estado2* donde se reciben los datos, se forman los nombres y las distancias, se ordenan, se manipulan para obtener la posición del *TAG* y por último se actualiza el valor de la variable *MillisAnterior* y se solicita nuevamente las distancias a los *Anchors* (Figura 58).

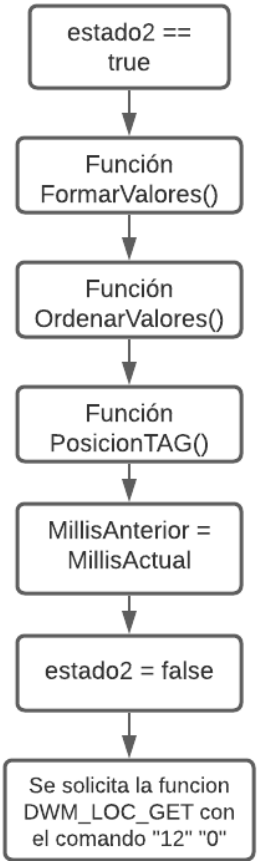


Figura 58: Diagrama del estado2

En el caso donde no coinciden los bytes de verificación, no se entra a la estructura condicional *estado2* y se regresa a Loop principal del código donde se actualiza y se verifica la condicional de *MillisActual y MillisAnterior*

Adentro de la estructura condicional *estado2* se encuentran las funciones: *FormarValores()*, *OrdenarValores()* y *PosiciónTAG* (Figura 58).

Función formar valores

La función *FormarValores()* lee los valores del buffer y arma de manera correcta tomando en cuenta el formato *Little indian* el nombre del *Anchor* y su distancia hacia el *TAG* al asignarle a las variables *ANC1, ANC2 y ANC3* los nombres de los *Anchors* recibidos y en las variables *Dist1, Dist2 y Dist3* las distancias (Figura 59).

Para comprobar que se estaban recibiendo y formando bien los nombres de los *Anchors* y las distancias, se dejó al robot en una posición estática y se realizaron las mediciones. Gracias a la aplicación *DRTLS* se supo que el *Anchor1* tiene el número serial 1D25, el *Anchor 2* tiene el número serial 4BB8 y el *Anchor 3* tiene el número serial 581A. Al imprimir los valores formados se observó que efectivamente se formaron los nombres y las distancias correctamente. Sin embargo, se observó que los valores no entraban en el mismo orden siem-

```
void FormarValores() {
  // ----- Formando anchor1 y distancia 1 -----
  ANC1 = (bufer[1] << 8) + (bufer[0]);
  Dist1 = (bufer[5] << 24) + (bufer[4] << 16) + (bufer[3] << 8) + (bufer[2]);

  // ----- Formando anchor1 y distancia 2 -----
  ANC2 = (bufer[21] << 8) + (bufer[20]);
  Dist2 = (bufer[25] << 24) + (bufer[24] << 16) + (bufer[23] << 8) + (bufer[22]);

  // ----- Formando anchor1 y distancia 3 -----
  ANC3 = (bufer[41] << 8) + (bufer[40]);
  Dist3 = (bufer[45] << 24) + (bufer[44] << 16) + (bufer[43] << 8) + (bufer[42]);
}
```

Figura 59: Función Formar valores

pre. Esto quiere decir que al enviar el código "12" "0" al *TAG*, este recopilaba la información de los *Anchors* de forma aleatoria. A veces la información recopilada al principio pertenecía al *Anchor2*, y otras veces al *Anchor1* y así aleatoriamente (Figura 60).

```
Anchor1: 581A
Distancia1: 135
Anchor2: 1D25
Distancia2: 99
Anchor3: 4BB8
Distancia3: 1644

Anchor1: 4BB8
Distancia1: 195
Anchor2: 581A
Distancia2: 134
Anchor3: 1D25
Distancia3: 1644
```

Figura 60: Verificación de orden en las posiciones

Función ordenar valores

Debido a esto se realizó la función *OrdenarValores()* (Figura 61) en la cual organizaba a los *Anchors* y a sus distancias para que siempre el primer valor de distancia formado perteneciera al *Anchor1*, el segundo valor de distancia al *Anchors2* y el tercero al *Anchor3*.

Para esto se realizó una serie de estructuras condicionales donde se comparó el primer nombre de *Anchor* recibido y luego se evaluaron las siguientes dos alternativas para el segundo *Anchor*. Dependiendo de cual estructura cumplió completamente, se ordenaron los *Anchors* y sus distancias respectivamente.

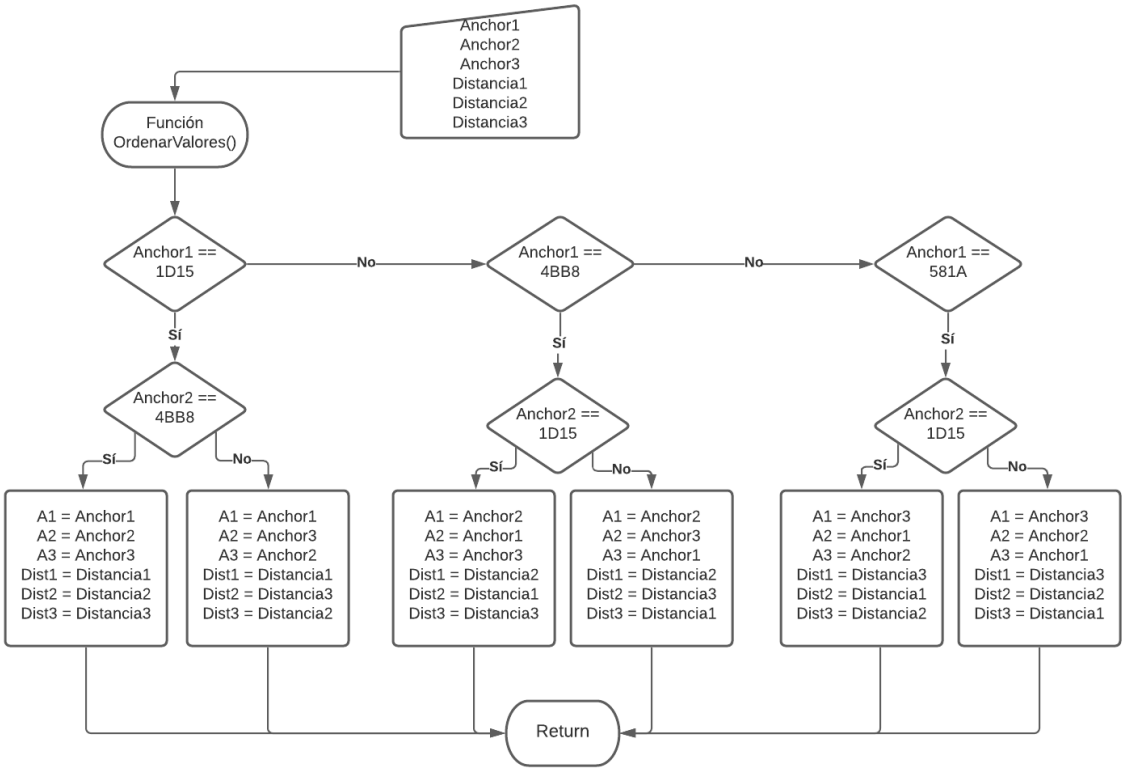


Figura 61: Función para ordenar los valores recibidos

Función posición TAG

Por último, está la función *PosicionTAG* (Figura 64). En esta función se refleja todo lo visto en el capítulo 11.2 (Figura 62).

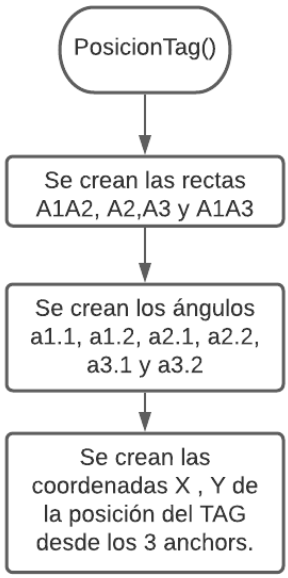


Figura 62: Diagrama de la función posiciónTAG

Para esto se inició definiendo las rectas *a, b, c* que representan a las rectas *A1A2, A2A3 y A1A3* respectivamente. Como las distancias entre *Anchors* puede cambiar dependiendo del terreno a perimetrar, estás rectas de definieron a partir de las coordenadas de los *Anchors*.

Los ángulos a1.1, a1.2, a2.1, a2.2, a3.1 y a3.2 se representan como An1, An2, Bn1, Bn2, Cn1 y Cn2 respectivamente en la Figura 64. Tal como se describió en el capítulo 11.2, estos ángulos se componen de las distancias recopiladas del sistema de localización en tiempo real (Dist#) y las que conforman el perímetro (a, b ,c).

En la Figura 64 se observa que se utilizó la función trigonométrica *acos()*. Esta función equivale a la función trigonométrica *arc cos()*. El problema con la función *arc cos()* es que matemáticamente no está definida para valores fuera del intervalo entre *(-1 < n < 1)*. Al evaluar el argumento de la función trigonométrica *acos()* se obtuvieron valores por encima del intervalo permitido. Esto provoca que la función *acos()* no esté definida matemáticamente (Figura 63)por lo que no se pudo obtener la posición a partir de la distancia.

```
Anchor 1 - 1D25: 1D25
Distancia1: 212.00
An1: nan
Posición: nan , nan
Anchor 2 - 4BB8: 4BB8
Distancia2: 427.00
Bn1: nan
Posición: nan , nan
Anchor 3 - 581A: 581A
Distancia3: 404.00
Cn1: nan
Posición: nan , nan

Anchor 1 - 1D25: 1D25
Distancia1: 226.00
An1: nan
Posición: nan , nan
Anchor 2 - 4BB8: 4BB8
Distancia2: 465.00
Bn1: nan
Posición: nan , nan
Anchor 3 - 581A: 581A
Distancia3: 432.00
Cn1: nan
Posición: nan , nan
```

Figura 63: Resultado del código

A pesar de eso se definió también la posición del *TAG* a partir de los ángulos y las distancias leidas desde el sistema de localización en tiempo real.

```
// ------------------------------------------------------
// FUNCIÓN PARA FORMAR POSICIÓN DEL TAG
void PosicionTAG() {
  a = sqrt(pow((PosA3[0] - PosA2[0]), 2) + pow(PosA3[1] - PosA2[1], 2));
  b = sqrt(pow((PosA3[0] - PosA1[0]), 2) + pow(PosA3[1] - PosA1[1], 2));
  c = sqrt(pow((PosA2[0] - PosA1[0]), 2) + pow(PosA2[1] - PosA1[1], 2));

  An1 = acos((-pow(Dist2, 2) + pow(c, 2) + pow(Dist1, 2)) / (2 * c * Dist1)) * (180 / pi);
  float An2 = acos((-pow(Dist3, 2) + pow(b, 2) + pow(Dist1, 2)) / (2 * b * Dist1)) * (180 / pi);

  Bn1 = acos((-pow(Dist1, 2) + pow(c, 2) + pow(Dist2, 2)) / (2 * c * Dist2)) * (180 / pi);
  float Bn2 = acos((-pow(Dist3, 2) + pow(a, 2) + pow(Dist2, 2)) / (2 * a * Dist2)) * (180 / pi);

  Cn1 = acos((-pow(Dist1, 2) + pow(b, 2) + pow(Dist3, 2)) / (2 * b * Dist3)) * (180 / pi);
  float Cn2 = acos((-pow(Dist2, 2) + pow(a, 2) + pow(Dist3, 2)) / (2 * a * Dist3)) * (180 / pi);

  X1 = Dist1 * cos(An1 * pi / 180);
  Y1 = Dist1 * sin(An1 * pi / 180);

  X2 = (PosA2[0] - PosA1[0]) - (Dist2 * cos(Bn1 * pi / 180));
  Y2 = Dist2 * sin(Bn1 * pi / 180);

  X3 = (PosA3[0] - PosA1[0]) - (Dist3 * sin(Cn2 * pi / 180));
  Y3 = (PosA3[1] - PosA2[1]) - (Dist3 * cos(Cn2 * pi / 180));
}
```

Figura 64: Función para encontrar la posición del TAG

Programa para comprobar función Posición TAG

Para comprobar que el código realizado con las operaciones matemáticas fuera correcto, se realizó un programa con la misma estructura y funciones matemáticas utilizadas. En este programa en lugar de obtener las distancias de los *Anchors* al *TAG*, se colocó las coordenadas donde supuestamente se encuentra el robot (Figura 65).

```
float Punto [2] = { 0.75, 0.43};

float PosA1 [2] = {0, 0};
float PosA2 [2] = {1, 0};
float PosA3 [2] = {1, 1};

float a = sqrt(pow((PosA3[0] - PosA2[0]), 2) + pow(PosA3[1] - PosA2[1], 2));
float b = sqrt(pow(PosA3[0] - PosA1[0], 2) + pow(PosA3[1] - PosA1[1], 2));
float c = sqrt(pow(PosA2[0] - PosA1[0], 2) + pow(PosA2[1] - PosA1[1], 2));

float Dist1 = sqrt(pow(PosA1[0] + Punto[0], 2) + pow(PosA1[1] + Punto[1], 2));
float Dist2 = sqrt(pow(PosA2[0] - Punto[0], 2) + pow(PosA2[1] + Punto[1], 2));
float Dist3 = sqrt(pow(PosA3[0] - Punto[0], 2) + pow(PosA3[1] - Punto[1], 2));
```

Figura 65: Simulación del sistema RTLS

Simulando así la posición actual del *TAG*, el programa realizó nuevamente todos los cálculos de igual manera que como si se estuvieran recopilando la información del RTLS (Figura 66).

```
float An1 = acos((-pow(Dist2, 2) + pow(c, 2) + pow(Dist1, 2)) / (2 * c * Dist1)) * (180 / pi);
float An2 = acos((-pow(Dist3, 2) + pow(b, 2) + pow(Dist1, 2)) / (2 * b * Dist1)) * (180 / pi);

float Bn1 = acos((-pow(Dist1, 2) + pow(c, 2) + pow(Dist2, 2)) / (2 * c * Dist2)) * (180 / pi);
float Bn2 = acos((-pow(Dist3, 2) + pow(a, 2) + pow(Dist2, 2)) / (2 * a * Dist2)) * (180 / pi);

float Cn1 = acos((-pow(Dist1, 2) + pow(b, 2) + pow(Dist3, 2)) / (2 * b * Dist3)) * (180 / pi);
float Cn2 = acos((-pow(Dist2, 2) + pow(a, 2) + pow(Dist3, 2)) / (2 * a * Dist3)) * (180 / pi);

float X1 = Dist1 * cos(An1 * pi / 180);
float Y1 = Dist1 * sin(An1 * pi / 180);

float X2 = (PosA2[0] - PosA1[0]) - (Dist2 * cos(Bn1 * pi / 180));
float Y2 = Dist2 * sin(Bn1 * pi / 180);

float X3 = (PosA3[0] - PosA1[0]) - (Dist3 * sin(Cn2 * pi / 180));
float Y3 = (PosA3[1] - PosA2[1]) - (Dist3 * cos(Cn2 * pi / 180));
```

Figura 66: Simulación del sistema RTLS

En la Figura 66 se observa que se calcularon tres valores de X y tres valores de Y, estos valores corresponden a la posición del *TAG* calculada desde cada *Anchor*. Como en este programa se introdujo la supuesta posición del *TAG*, estos valores de X y Y calculados son iguales a las coordenadas de posición indicadas.

Al tener los valores de *X1 Y1*, *X2 Y2* y *X3 Y3* iguales a las coordenadas del punto especificado para la posición del *TAG* se comprobó que matemáticamente el código si calcula de manera correcta la posición del *TAG*. Por lo que el problema al utilizar el RTLS es cuestión del tamaño de los datos.

CAPÍTULO 12

Modo automático

12.1. Control con encoders ópticos

Para hacer que el robot se condujera de modo autónomo fue necesario tener información de la velocidad de giro cada rueda ya que estas están impulsadas por motores DC. Los motores DC, debido a su fabricación, nunca podrán hacer girar las ruedas 100 % en sincronía, siempre habrá un motor que hará girar una rueda más rápido que la otra. En ocasiones, estas diferencias de velocidad de giro pueden llegar a ser despreciables, sin embargo, en el robot secador de café no es el caso. Debido a esto, la implementación de un modo de leer la velocidad de cada rueda y la implementación de un controlador para la velocidad de cada motor fue necesario.

La lectura de la velocidad de giro de cada rueda se logró por medio de encoders ópticos. Para esto se implementó una rueda ranurada en cada una de las llantas impulsada por el motor DC. La rueda ranurada, al estar sujetado al eje rotativo del motor, gira a la misma velocidad que lo hace la llanta, y con ayuda del módulo F249 (capítulo 6.1.3) se contabiliza cada una de las ranuras que pasan por él, a esto se le llama tick. A este sistema de rueda ranurada en conjunto del módulo F249 es llamado "encoder óptico".

12.1.1. Primer propuesta de control - Odometría para controlar la posición del robot

La herramienta por excelencia que se utiliza al momento de utilizar encoders ópticos para implementación de un sistema de control para un vehículo con ruedas es la odometría, ya que esta entrega la posición actual del robot partiendo de un punto de referencia, al igual que es posible tener control de la velocidad del centro de masa del robot.

Como primer controlador, se implementó la odometría (capítulo 6.3.1) acompañado de un controlador PID para controlar la posición final del robot. Sin embargo, el propósito final del robot era seguir una ruta constantemente, no únicamente llegar a una posición final exacta, por lo que controlar la posición final del robot no cumplía con nuestras necesidades. Además, aun se presentaban problemas que acompañan a la odometría como el deslizamiento de las ruedas por lo que era necesario implementar otro controlador que compense la posición al presentarse el deslizamiento.

12.1.2. Segunda propuesta de control - Control de la velocidad del robot a partir de encoders

Control de velocidad utilizando odometría y control PID

Para la propuesta de control de velocidad del robot utilizando encoders se utilizó nuevamente la odometría solo que ahora se agregó la dimensión de tiempo.

La base de este controlador constaba de medir la cantidad de tiempo que se tardaba el módulo F249 en reportar "n" cantidad de ticks, en este caso, 20 ticks. Al conocer el diámetro de la llanta y la cantidad de agujeros que tiene el disco ranurado, se sabe cuántos ticks deben reportarse para que la llanta haya dado una revolución y así con estos datos poder calcular la velocidad angular y luego lineal. El problema fue al momento de implementar el tiempo ya se presentaba una incertidumbre en las lecturas. Como se usaron encoders ópticos con un disco ranurado, existió el caso en que no se mueve la rueda, pero una de las ranuras activa y desactiva un pulso, al igual que el deslizamiento. Otro de los mayores problemas que se encontró es cuando el disco ranurado se encuentra tapando la comunicación infrarroja del módulo F249 de manera continua, ósea, el sensor infrarrojo se ve interrumpido por la parte del disco ranurado que se encuentra entre dos agujeros del disco de manera estática (Figura 67).

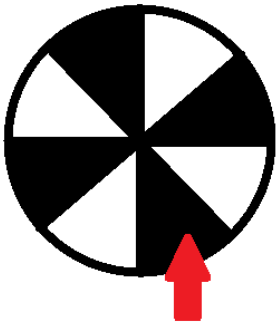


Figura 67: Ejemplo de la interferencia de la comunicación infrarroja del módulo F249

En este caso, a pesar de que la llanta A se encuentre de manera estática, el controlador seguía haciendo conteo del tiempo que se tardó esta llanta en completar una revolución por lo que el controlador PID aceleraba de manera abrupta la llanta A y detenía a la llanta B lo que provocaba que el carro cambiara de dirección. Este cambio de velocidades se repite una y otra vez por lo que cada vez se acumulaba más el error en las mediciones.

Para solucionar este problema se colocó un tiempo de interrupción mínimo entre tick y tick, si este tiempo de interrupción entre tick y tick era superado, se asumía que la rueda se encontraba parada o estática. El problema que se obtuvo con esta solución fue que ese tiempo era aleatorio o dicho de otra manera, era un tiempo que no se podía calcular matemáticamente, además debido al tiempo de interrupción se llegaba a asumir que la llanta se encontraba estática pero no era el caso, por lo que al hacer esto estábamos introduciendo un tipo de deslizamiento al sistema que a la larga se puede representar hasta cómo si la llanta haya realizado revoluciones completas y el controlador no lo tomará en cuenta.

Debido a estos problemas se optó por implementar otra manera de controlar la velocidad de las llantas a partir de los encoders ópticos.

Control de velocidad utilizando conteo de ticks y control proporcional

Como existieron problemas al implementar el tiempo para el control de velocidad con odometría y el controlador PID, se realizó un control más sencillo pero útil. Para este control se asumió la existencia del deslizamiento cómo algo inevitable. Se controló únicamente la velocidad dejando el problema del deslizamiento para corregirlo mediante controladores a base de otros sensores.

Para este caso, se trabajó con los ticks del encoder para cada rueda y se buscó mantener la misma cantidad de conteo de ticks en ambas ruedas. Al mantener el mismo conteo de ticks entre las ruedas, se aseguró que ambas ruedas estaban girando a la misma velocidad.

El controlador funciona de la siguiente manera: Ambos motores se les colocó una velocidad inicial, esta velocidad es proporcionada por un valor PWM a cada motor a través del módulo DRI0018 (capítulo 6.1.4). La velocidad inicial representa la velocidad constante con la que el robot se desplaza. Mientras el robot avanza, los encoders van realizando conteos de ticks, cada vez que se presente un tick en un encoder, se incrementa en una unidad la variable individual creada que representa a cada rueda. Gracias a eso se pudo saber con certeza cuantos ticks reportaba cada motor.

Como la velocidad de cada motor está ligada al valor PWM que se le manda a cada uno, se utilizó la velocidad PWM de cada motor para controlar al robot. Se realizó una función comparativa en la cual se leyó el número de ticks de cada motor (motor A y B) y se determinó si un motor presentaba un número de ticks mayor al otro. Si el motor A presenta una cantidad de 67 ticks y el motor B presenta una cantidad de 53 ticks, se disminuye el valor PWM del motor A para que este baje la velocidad y así el motor B que continua a la misma velocidad inicial alcance al motor A en ticks. Si ambos motores presentan el mismo valor de ticks, se les colocó el mismo valor de PWM a ambos motores. De esta manera se realizó el controlador proporcional en el cual se toma en cuenta el valor presente de ticks para manipular la velocidad.

Los problemas que presentó este controlador fue que no siempre se reporta al mismo tiempo el tick en cada uno de los sensores, por lo que se obtenía una diferencia de ticks de por lo menos una unidad en un instante de tiempo. Para solucionar esto, se realizó la variación en la velocidad de un motor únicamente si la diferencia entre el número de ticks entre ambos motores era superior a dos unidades. De esta manera se aseguró que ese intervalo

de tiempo entre lectura de cada encoder no perjudicó al control. Además, se agregaron límites inferiores de velocidad con dos fines.

1. Si el conteo de ticks del motor A va por debajo del conteo de ticks del motor B y el motor A no logra alcanzar al motor B, el motor B seguirá disminuyendo su valor PWM hasta llegar al punto de estar en valores negativos lo cual no es posible y ocasionaría un problema en el sistema.

2. El robot al estar compuesto por varios módulos y sensores, tiene un peso razonable, por lo que es necesario un torque mínimo por parte de cada motor para poder superar el estado de reposo y empezar a mover al robot, si el torque de los motores se encuentra por debajo del torque que supera el reposo, los motores no lograrán mover al robot y estos podrían arruinarse. Para prevenir esto, se colocó el límite inferior con el cual el motor aún se encuentra por encima del torque mínimo necesario para superar el estado de reposo y así se asegura que los motores no se arruinen.

El controlador proporcional de velocidad resultante se presenta en (Figura 68).

```
void loop() {
  avanzar(Velocidad_D, Velocidad_I);    // Mandar velocidad a los motores

  if (Tick_D > Tick_I + 1) {            // Si el número de ticks es mayo en el motor derecho
    Velocidad_D = Velocidad_D - 1;      // bajarle la velocidad al motor derecho
  } else if (Tick_I  > Tick_D + 1) {    // Si el número de ticks es mayo en el motor izquierdo
    Velocidad_I = Velocidad_I - 1;      // bajarle la velocidad al motor izquierdo
  } else {                              // si ambos tienen la misma cantidad de ticks
    Velocidad_D = PWM;                  // ambos se les carga la misma velocidad
    Velocidad_I = PWM;
  }

  if (Velocidad_D < PWM_Min) {          // Limite mínimo de velocidad aceptada
    Velocidad_D = PWM_Min;
  }
  if (Velocidad_I < PWM_Min) {          // Limite mínimo de velocidad aceptada
    Velocidad_I = PWM_Min;
  }
}

void avanzar(char a, char b) {        // Funcion para mover motores
  analogWrite (E1, a);
  analogWrite (E2, b);
}

void EncoderD () {                  // Lectura de tick motor derecho
  Tick_D = Tick_D + 1;
}

void EncoderI () {                  // Lectura de tick motor izquierdo
  Tick_I = Tick_I + 1;
}
```

Figura 68: Control proporcional a base de comparación de ticks

A pesar del buen funcionamiento del controlador de velocidad proporcional, aun se encontraba presente el deslizamiento. Como solución se utilizó el *RTLS* para darle solución al problema de deslizamiento y cumplir con la ruta descrita para el robot para secar el café.

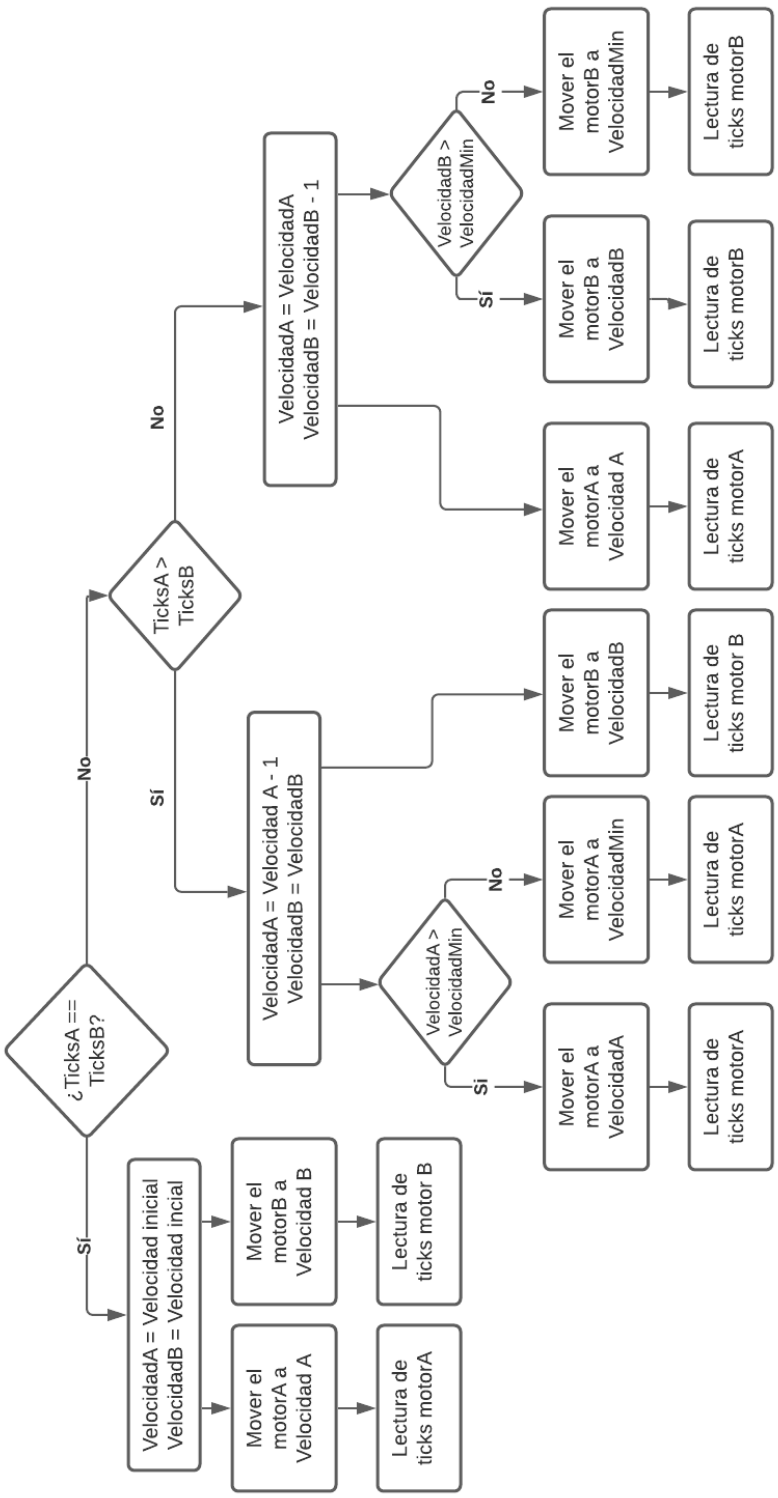


Figura 69: Diagrama del funcionamiento del controlador

12.2. Control de posición a base del RTLS

En este caso, lo que se buscó es tener la posición actual del robot en todo momento (coordenadas X y Y) y a raíz de estas hacer un camino o una línea con un ancho mayor al del carrito para procurar que el robot no se salga de esta línea. Este es el mismo concepto del robot seguidor de una línea negra, sin embargo, en nuestro caso lo que el robot hace es tratar de permanecer adentro del camino, cómo si el camino fuera la línea a seguir.

Para esto primero se formó el camino o la línea a seguir. Esta línea parte de una coordenada X en la cual permanecería. El robot debía de avanzar en dirección de la coordenada Y siempre permaneciendo en la misma coordenada X o por lo menos lo más cercano posible. Sin embargo, al existir un error en la lectura de la posición del *TAG* de hasta 10cm según el fabricante [39], se hizo que la línea tenga un ancho de 20 centímetros para que el robot tenga un margen de error de más-menos 10 centímetros de movimiento.

Se definió la posición X a seguir y los límites superior e inferior. Se colocó al robot físicamente en la coordenada X establecida viendo en dirección Y positivo. Luego el robot empezó a leer su posición actual para formar el vector promedio y así tener una referencia de donde se encontraba ubicado.

Una vez formado el vector posición, se alimentaron los motores con un mismo valor PWM para que ambas ruedas se muevan a la misma velocidad y el robot se mueva en línea recta. Sin embargo, este no fue el caso.

Cómo aún existe el deslizamiento, y la diferencia de eficiencia de los motores DC, el robot tendía moverse para uno de los lados. Para esto se implementó el sistema de localización en tiempo real. Cómo se recopiló la información de la posición actual del robot en todo momento, se verificó si este permanecía dentro de los límites establecidos. Cuando el vector promedio indicaba que el robot se había salido de los límites, el controlador disminuía la velocidad del motor opuesto a dónde se superó el límite para que así el robot volviera a encarrilarse (Figura 70). En este controlador, al igual que en el capítulo (12.1.2), se establecieron límites de velocidad mínimos para la posible velocidad de cada motor, argumentado de la misma manera que se hizo anteriormente.

De esta manera se aseguró que el robot permaneciera dentro de la línea indicada y en conjunto al control por medio de encoders ópticos, se lograría un control óptimo y así cumplir con barrer el terreno de café siguiendo el patrón establecido.

Sin embargo, debido los saltos de posición y la inexactitud de los datos de posición recibidos del *RTLS* vistos en el capítulo (10.3.1), este controlador a pesar de funcionar correctamente y mantener al robot dentro de la línea establecida, no se puede utilizar porque las mediciones de los datos de posición son erróneas. El controlador hace que el robot siga una línea incorrecta ya que detecta que el robot se encuentra en otra posición que no es la actual provocando que en realidad el robot se descarrile al tratar de buscar nuevamente la línea cuando esto no es necesario.

```
// ----- Mandando velocidad a los motores -----
avanzar(Velocidad_D, Velocidad_I);

// ----- Ambos motores a la misma velocidad -----
if (Velocidad_D == Velocidad_Min && Velocidad_I == Velocidad_Min) {
  Velocidad_D = Velocidad;
  Velocidad_I = Velocidad;
}

// ----- Control de velocidad -----
if (promedio(X, 1) > Limite_SupX) {          // Si se detecta que el robot superó el límite superior
  Velocidad_I = Velocidad_I - 1;             // Se disminuye la velocidad del motor opuesto
} else if (promedio(X, 1) < Limite_InfX) {   // Si se detecta que el robot superó el límite inferior
  Velocidad_D = Velocidad_D - 1;             // Se disminuye la velocidad del motor opuesto
} else {                                     // Si el robot se encuentra encarrilado
  Velocidad_D = Velocidad;                   // Ambos motores van a la misma velocidad.
  Velocidad_I = Velocidad;
}

// ----- Límite de velocidad mínima -----
if (Velocidad_I < Velocidad_Min) {
  Velocidad_I = Velocidad_Min;
}
if (Velocidad_D < Velocidad_Min) {
  Velocidad_D = Velocidad_Min;
}
```

Figura 70: Sistema de control según la posición recopilada por el RTLS

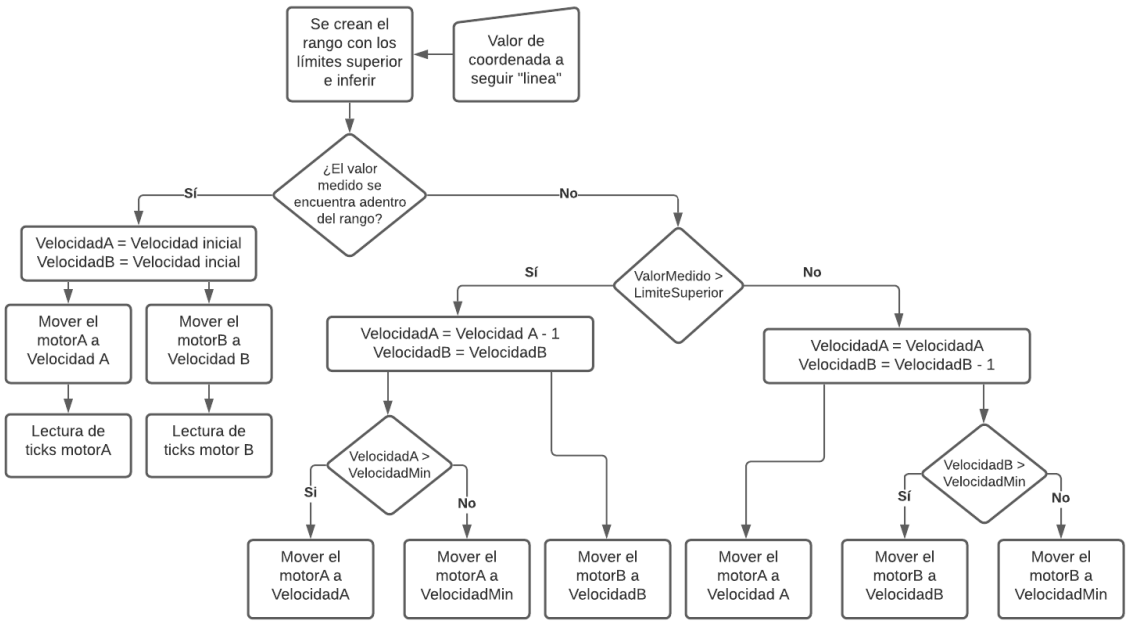


Figura 71: Diagrama del control de posición a base del RTLS

CAPÍTULO 13

Modo manual

Como se solicitó la opción de manipular al robot manualmente, se implementó un control remoto para que un operario pueda manipularlo. La comunicación entre el control remoto y el robot se realizó por medio de radio frecuencia con ayuda de módulos NRF24L01 (Figura 72).

Figura 72: Módulo NRF24L01

Este módulo trabaja en la banda de 2.4GHz. Se utilizó módulos NRF24L01 debido a su bajo consumo, bajo costo y la potencia de transmisión la cual gracias a su antena logra un rango de hasta 100m sin interrupciones en su campo de visión. [40]

Se realizó la conexión de radio frecuencia entre el control remoto y el robot estableciendo primero el protocolo de comunicación que se utilizaría entre ambos dispositivos. Para esto se inicializó el módulo receptor, se establecieron los canales de escritura y de lectura cada uno con sus respectivas direcciones a utilizar. A ambos dispositivos se les colocó la misma dirección de enlace de comunicación. Por último, se se estableció el nivel de amplificador de poder (Power amplifier level) (Figura 73) [41]. Esta configuración se establece una única vez

al iniciar el sistema de comunicación vía RF. Como dato importante, los módulos NRF24L01 manejan una comunicación síncrona por lo que fue necesario establecer una señal de reloj en el sistema.

Luego de inicializada la comunicación vía RF, se leyeron datos de manera similar a la lectura de datos via comunicación serial por medio de cables.

En el caso del robot secador de café se implementaron cuatro diferentes controles remotos, dos de ellos son propuestas para el control principal del proyecto y los otros dos control se diseñaron para realizar pruebas de movimiento y manejo del robot manualmente.

```
// NRF24L01
radio.begin();                          //inicializamos el NRF24L01
radio.openWritingPipe(addresses[0]);    //Abrimos un canal de escritura (00001)
radio.openReadingPipe(1, addresses[1]); //Abrimos el canal de Lectura (00002)
radio.setPALevel(RF24_PA_MIN);
```

Figura 73: Inicialización del protocolo de comunicación de RF

13.1. Primera propuesta de control remoto

La primera propuesta de control remoto la presentó Debora Raquel López. Deboda realizó la comunicación de radio frecuencia a base de módulos NRF24L01.

Los parametros del robot que Debora controla a través del control remoto son:

1. Avance
2. Cruce a la izquierda
3. Retroceso
4. Cruce a la derecha
5. Selección de modo manual/automático
6. Selección de velocidad (lenta, media, alta)

Debora eligió trabajar la manipulación del robot utilizando Joysticks. El primer joystick manipula el avance y retroceso del robot, y el segundo controla el movimiento lateral (movimiento de izquierda o derecha). Para el cambio de modo manual/automático utilizó un slide switch que define en qué modo se encuentra operando el robot. Y por último utilizó un push bottom para define la velocidad de robot (lenta, media, rápida).

Si el robot se encuentra en modo automático, este debe de funcionar autónomamente por lo que se inhabilita las lecturas de comandos enviados por el control remoto hacia el robot hasta que este se coloque nuevamente en modo manual.

Para la transmisión de datos, Débora trabajó con estructuras de datos las cuales se controlan y modifican a través del control remoto. Esta estructura está compuesta por los parámetros del robot a controlar. Luego que se manipulan los datos, se realiza la transmisión y por último se envía a través de la comunicación vía RF hacia el robot. Del lado del robot, se recibe la estructura completa y está dependiendo de que valor contenga el parámetro de la estructura, se define que acción realizará el robot.

Los parámetros recibidos son almacenados en variables booleanas. Estos se encuentran en alto o en bajo, o visto de otra manera, en 1 o 0. De esta manera al recibir la estructura proveniente del control remoto se logró identificar fácilmente que acción se solicitó realizar.

Para controlar al robot se realizó una serie de comparaciones donde se verificó cual es el parámetro que se encuentra en alto y dependiendo de eso, se realiza la acción solicitada. Cómo por ejemplo se puede recibir que el control se encuentra en modo manual y el parámetro de avanzar se encuentra en alto, esto quiere decir que el resto de parámetros se encuentran bajos y la acción que realizará el robot es avanzar a la velocidad definida, la cual por defecto es "lenta".

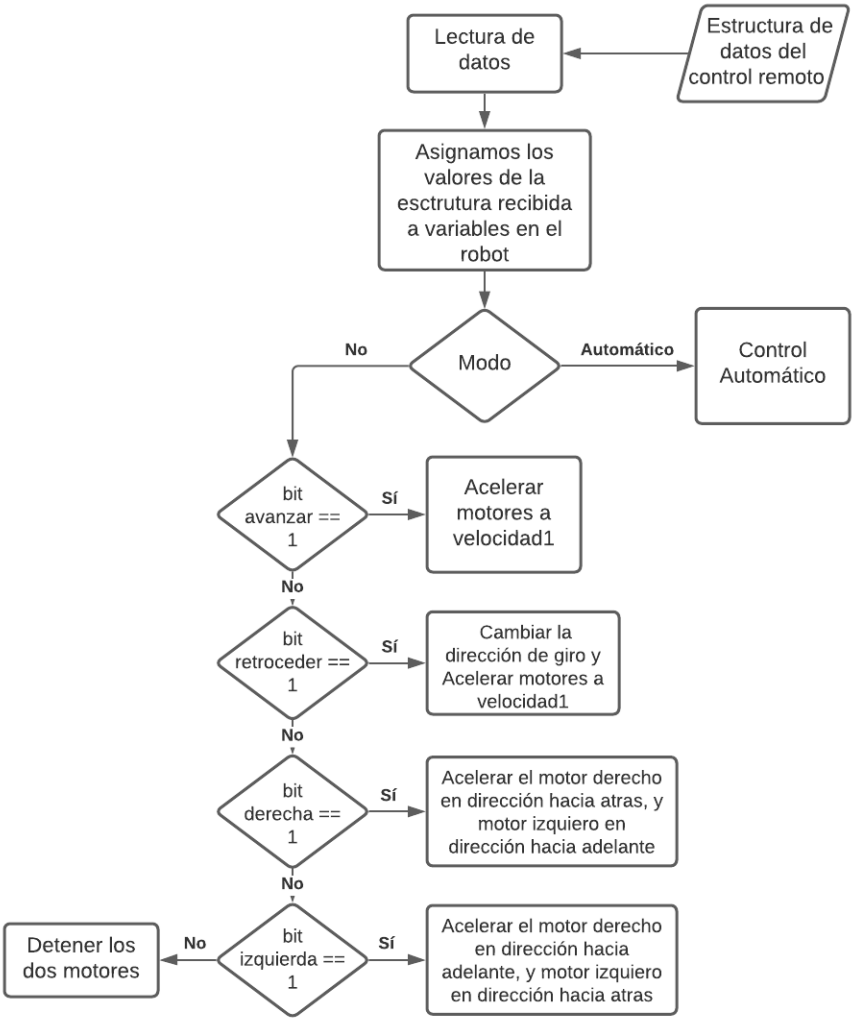


Figura 74: Diagrama del funcionamiento de la primera propuesta de control remoto

13.2. Segunda propuesta de control remoto

La primera propuesta de control remoto la presentó Jeffrey Muñoz. Jeffrey realizó la comunicación de radio frecuencia a base de módulos NRF24L01.

Los parámetros del robot que Jeffrey controla a través del control remoto son:

1. Cruce a la derecha
2. Cruce a la izquierda
3. Retroceso
4. Avance
5. Selección de modo manual/automático
6. Selección de velocidad (lenta, media, alta)

Jeffrey eligió trabajar la manipulación del robot utilizando botones. Para el cambio de modo manual/automático utilizó un push bottom el cual realiza la función de invertir el valor que definirá en qué modo se encuentra operando el robot (automático o manual). Para mover al robot se utilizaron 4 push bottom los cuales indican la función de avanzar, retroceder, cruzar a la izquierda o cruzar a la derecha respectivamente. Por último, utilizó tres push bottom para definir la velocidad de robot, lenta, media o rápida respectivamente.

Si el robot se encuentra en modo automático, el robot debe de funcionar autónomamente por lo que se inhabilita las lecturas de comandos enviados por el control remoto hacia el robot hasta que este se coloque nuevamente en modo manual.

Para la transmisión de datos, Jeffrey trabajó con un array de datos el cual se controlan y modifican a través del control remoto. Este array es de dimensión 1x8. Cada uno de los bits del array pertenece a un parámetro a manipular en el robot por lo que al presionar un botón se está refiriendo a un único bit del array. Estos bits se manipulan y luego se envía a través de la comunicación vía RF hacia el robot para así controlar los parámetros necesarios en el robot. Del lado del robot, se recibe el array enviado y este dependiendo del valor del array, se define que acción realizar.

Para simplificar el control del robot, los parámetros recibidos son representados cómo valores booleanos. Estos se encuentran en alto o en bajo, o visto de otra manera, en 1 o 0. De esta manera al recibir el array proveniente del control remoto se logró identificar fácilmente que acción se solicitó realizar. Se le asignó una posición del array a cada uno de los parámetros del robot para realizar la manipulación de datos.

Para controlar al robot se realizó una serie de comparaciones donde se verificó cual es el parámetro que se encuentra en alto y dependiendo de eso, se realiza la acción solicitada. Cómo por ejemplo se puede recibir que el control se encuentra en modo manual y el parámetro de avanzar se encuentra en alto, esto quiere decir que el resto de parámetros se encuentran bajos y la acción que realizará el robot es avanzar a la velocidad definida, la cual por defecto es "lenta".

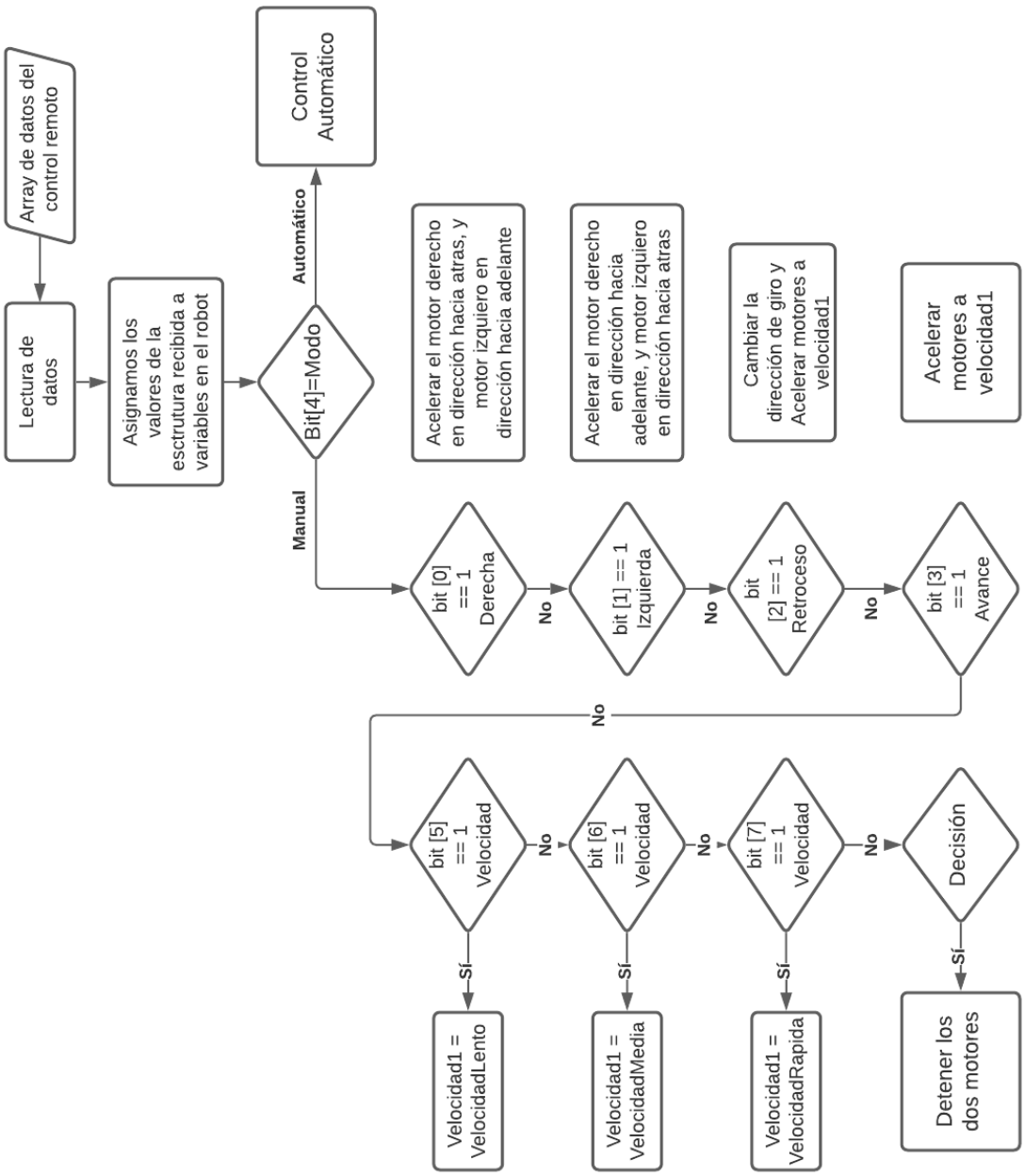


Figura 75: Diagrama del funcionamiento de la segunda propuesta de control remoto

CAPÍTULO 14

Robot secador de cafe

14.1. Envio de datos - Control remoto

Para el control remoto se utilizó la comunicación de radio frecuencia a base de módulos NRF24L01. Se controló el avance y retroceso de las ruedas individualmente. también se controló la manipulación de dos motores DC que se encargan de elevar las aspas que mueven el café.

Se utilizaron dos joysticks, uno para cada una de las ruedas, al igual que dos push bottoms para manipular el movimiento de las aspas. En este caso la velocidad del robot depende de la intensidad con la que se manipulan los joysticks. Este control tiene 10 posibles velocidades para el robot secador de café y 3 posibles estados para las aspas (levantar, bajar y parar).

Se creó también un segundo control el cual sustituye los joysticks por push bottons. Este control maneja únicamente 2 velocidades para cada motor (adelante o atrás) y para las aspas controla los tres posibles estados.

Se manipulan 3 parámetros del robot: motor izquierdo, motor derecho y aspas. Se enviaron 3 bytes desde el control al robot. Para asegurarse que la comunicación se estableciera correctamente se implementaron 2 bytes más al envió de datos desde el control remoto.

El primer byte de la cadena es el byte de *inicio*. Este byte funciona como un seguro, si se recibe este byte, se le indica al microcontrolador que se empezará la lectura de datos provenientes del control remoto. Los siguientes 3 byte son los especificados anteriormente. Por último, el quinto byte indica cuando ya se terminaron de recibir los valores para la manipulación del robot, este byte funciona como un byte de cierre. De esta manera se aseguró que no se filtren señales parásitas o hayan perdida de datos.

Se decidió realizar la manipulación de los motores individualmente con el fin de poder corregir la dirección del robot ya que este debido a diferentes factores, no anda en línea recta por sí solo al alimentar a ambos motores con un pulso PWM.

14.2. Recepción de datos y manipulación del robot secador de café

Para asegurar la lectura de datos, si no se recibe el byte de inicio ni de cierre, no se realizará la lectura completa. Dicho de otra forma, se descartan los siguientes valores entrantes hasta recibir el byte de inicio y cierre. Lo que hace que nuestra lectura sea más segura. Los signos de inicio y cierre de recepción de datos son: "<" y ">" respectivamente.

El robot secador de café se mantiene leyendo constantemente información a través del módulo NRF24L01. Si se recibe el byte de inicio y de cierre en las respectivas posiciones, se toma la lectura de datos cómo correcta.

Para manipular los motore que controlan las aspas del robot, se tienen 3 posibles valores a recibir. El valor "0" indica elevar las aspas. El valor "1" el estado estático de las aspas. Y el valor "2" indica el retraer aspas. Para esto se creó una función llamada "F_aspas". Como argumento de esta función se recibe el valor de "0", "1", o "2" (Figura 76).

```
// FUNCIÓN ASPAS
void F_Aspas (int a) {
  switch (a) {

    case 0:
      digitalWrite(As1, LOW);
      digitalWrite(As2, HIGH);
      digitalWrite(As4, LOW);
      digitalWrite(As3, HIGH);
      Serial.println("Aspas adelante");
      break;

    case 1:
      digitalWrite(As1, LOW);
      digitalWrite(As2, LOW);
      digitalWrite(As4, LOW);
      digitalWrite(As3, LOW);
      Serial.println("Aspas parado");
      break;

    case 2:
      digitalWrite(As1, HIGH);
      digitalWrite(As2, LOW);
      digitalWrite(As4, HIGH);
      digitalWrite(As3, LOW);
      Serial.println("Aspas atras");
      break;

  }
}
```

Figura 76: Función para controlar aspas del robot

El módulo PWM está compuesto por 32 bytes, por lo que tiene un rango de 0 a 255 bits. El control remoto controlado por joysticks envía un valor comprendido entre 0 a 20. Con el valor de 0 se indica la posición del joystick en un extremo y con 20 se indica la posición del extremo contrario del joystick. De esta manera se logró mapear todo el movimiento del joystick de extremo a extremo en un rango de 21 valores (20 de movimiento y 1 en su posición natural).

Se utilizan pulsos PWM para manipular la velocidad de los motores. Debido al módulo DRI0018 (capítulo 6.1.4) para el cambio de dirección de los motores se debe de colocar en alto o en bajo los pines de control de dirección.

Para controlar a los motores de las ruedas, se creó la función *Movimiento*. Esta función recibe cómo primer argumento el valor de la velocidad proveniente del control remoto y cómo segundo argumento se le indica a cuál de los dos motores (derecha o izquierda) se desea mover.

En el robot secador de café recibe el valor entre el rango de 0 a 20 para las velocidades de los motores de las ruedas. Este rango de valores representa el valor de movimiento hacia adelante o hacia atrás del robot. Para controlar la dirección de movimiento del robot se separó en dos este rango. Los valores de 0 al 9 indican la dirección de giro en una dirección y los valores entre el 11 al 20 indica la dirección de giro en sentido contrario. El valor "10" representa cuando el joystick está en su posición natural, ósea, no se está manipulando. Por lo que este valor indica que ese motor está en reposo. De esta manera se logró asignar la dirección de movimiento del robot (Figura 77).

```
// Selección de dirección del motor
if (a < 10) {              // Movimiento hacia atras
  Direccion = LOW;
} else if (a > 10) {       // Movimiento hacia adelante
  Direccion = HIGH;
} else {                   // Si no se recibe valor
  Direccion = LOW;
}

// Selección de motor a mover
if (b == 1) {              // Motor derecho
  E_motor = E1;
  M_motor = M1;
} else if (b == 2) {       // Motor Izquierdo
  E_motor = E2;
  M_motor = M2;
}
```

Figura 77: Rangos para la dirección de giro del robot

Para cambiar la velocidad de cada motor dependiendo del número recibido entre el rango de 0 a 20, se realizó una estructura switch case. Dependiendo del valor recibido se le asigna un valor PWM a la variable "Velocidad" la cual se le envía al motor indicado (Figura 78). La variable velocidad únicamente recibe argumentos positivos ya que se indica la dirección de giro de la rueda gracias a las estructuras antes mencionadas (Figura 77).

```
// Selección de velocidad de la rueda
switch (a) {
  case 0: Velocidad = 255; break;
  case 1: Velocidad = 230; break;
  case 2: Velocidad = 205; break;
  case 3: Velocidad = 180; break;
  case 4: Velocidad = 155; break;
  case 5: Velocidad = 130; break;
  case 6: Velocidad = 105; break;
  case 7: Velocidad = 80; break;
  case 8: Velocidad = 55; break;
  case 9: Velocidad = 30; break;
  case 10: Velocidad = 0; break;
  case 11: Velocidad = 30; break;
  case 12: Velocidad = 55; break;
  case 13: Velocidad = 80; break;
  case 14: Velocidad = 105; break;
  case 15: Velocidad = 130; break;
  case 16: Velocidad = 155; break;
  case 17: Velocidad = 180; break;
  case 18: Velocidad = 205; break;
  case 19: Velocidad = 230; break;
  case 20: Velocidad = 255; break;

  default: Velocidad = 0; break;
    return;
}
```

Figura 78: Cambio de velocidad de giro de la rueda

Por último se envía la dirección de giro, la velocidad y se especifica cuál de los dos motores se manipulará gracias a los valores formados por las estructuras anteriores. De esta manera con una única función (*Movimiento*) se logró manipular por completo al robot (Figura 79).

```
//-----------------------------------------------
// Función para que se muevan las ruedas
void Movimiento(byte a, byte b) {
  bool Direccion;
  byte E_motor;
  byte M_motor;
  byte Velocidad;

  // Selección de velocidad de la rueda
  switch (a) {

  // Selección de dirección del motor
  if (a < 10) {              // Movimiento hacia atras

  // Selección de motor a mover
  if (b == 1) {              // Motor derecho

  digitalWrite(M_motor, Direccion);
  analogWrite (E_motor, Velocidad);

}
```

Figura 79: Función para controlar el movimiento del robot

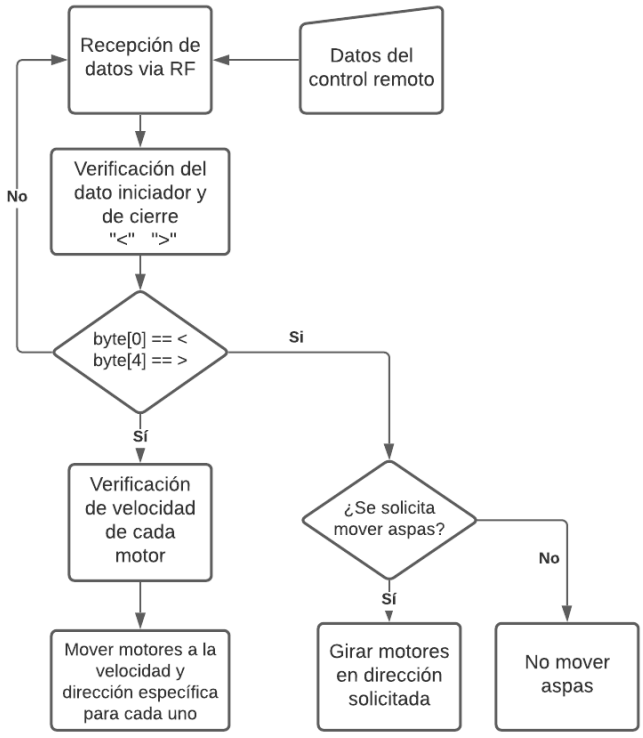


Figura 80: Diagrama de flujo del control remoto

14.3. Implementación física

El apartado estructural del robot lo realizó Juan Pablo Zea y el apartado eléctrico lo realizó Alvaro Torres.

Para la manipulación de la parte estructural del robot, se implementaron los siguientes componentes electrónicos:

- Arduino Mega
- Módulo DRI0018
- Módulo DWM1001
- Módulo NRF24L01

Para esto se montaron los componentes en una tabla de madera MDF (Figura 81). Esta tabla se sujetó a la parte estructural de enfrente del robot por medio de pernos M4.

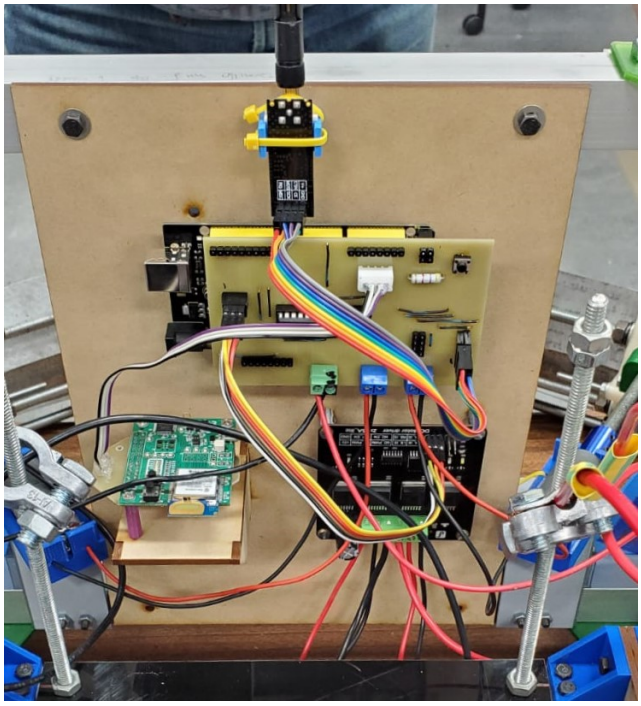


Figura 81: Sujeción de los componentes electrónicos

Cada módulo tiene distinto voltaje de alimentación. El arduino mega es alimentado con 12 voltios. El módulo DRI0018 y el módulo DWM1001 tienen un voltaje de alimentación de 5 voltios. Y por último el módulo NRF24L01 tiene un voltaje de alimentación de 3.3 voltios. También se tomó en cuenta el voltaje de alimentación de los motores de las ruedas y los motores de las aspas. Cada uno de estos tiene un voltaje de alimentación de 12 voltios.

Para alimentar a la estructura completa se utilizó un a batería de carro (Figura 82). Para proteger al circuito se colocó un switch de palanca de dos entradas.

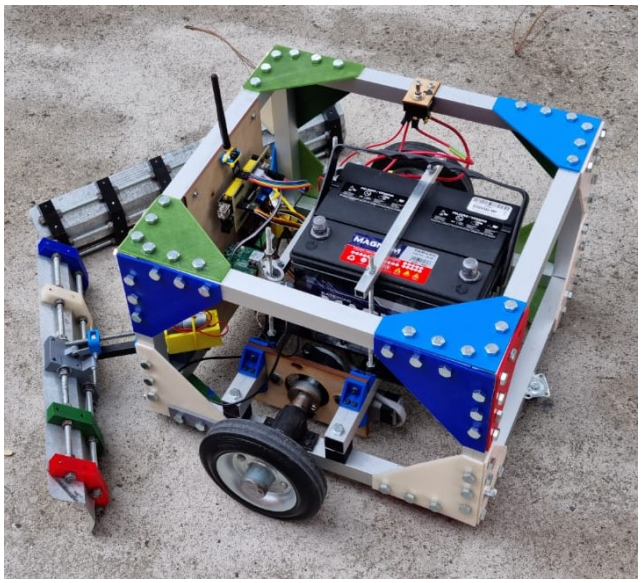


Figura 82: Alimentación del robot secador de cafe

Se obtuvieron los distintos voltajes menores a 12 voltios del arduino Mega. La distribución de voltajes se realizó de la siguiente manera (Figura 83).

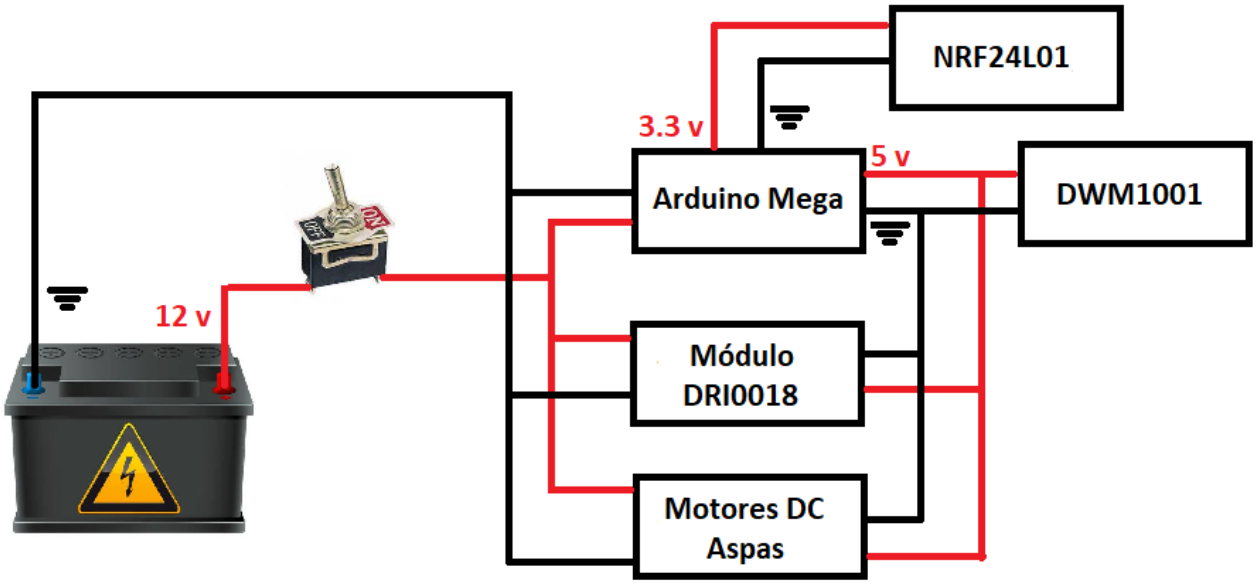


Figura 83: Distribución del voltaje de alimentación

14.4. Robot secador de café resultado final

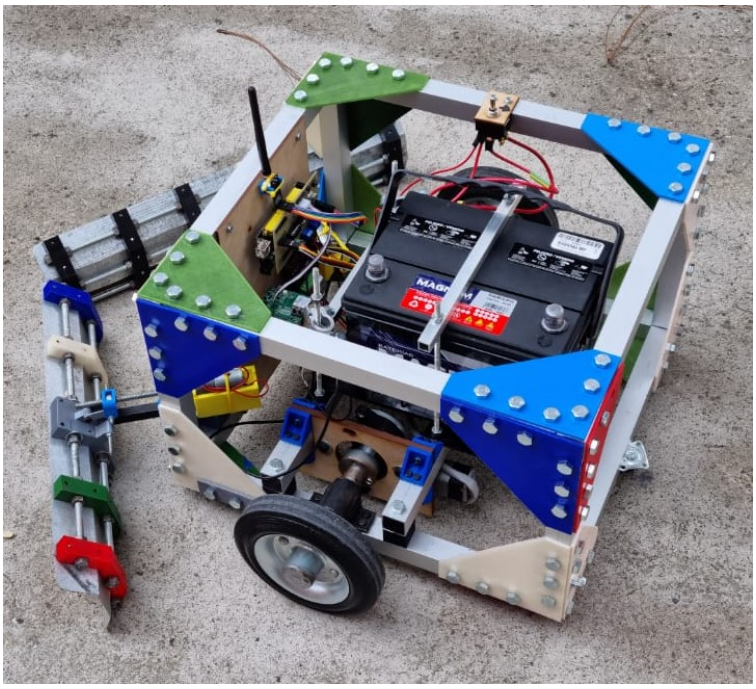


Figura 84: Robot secador de cafe parte trasera

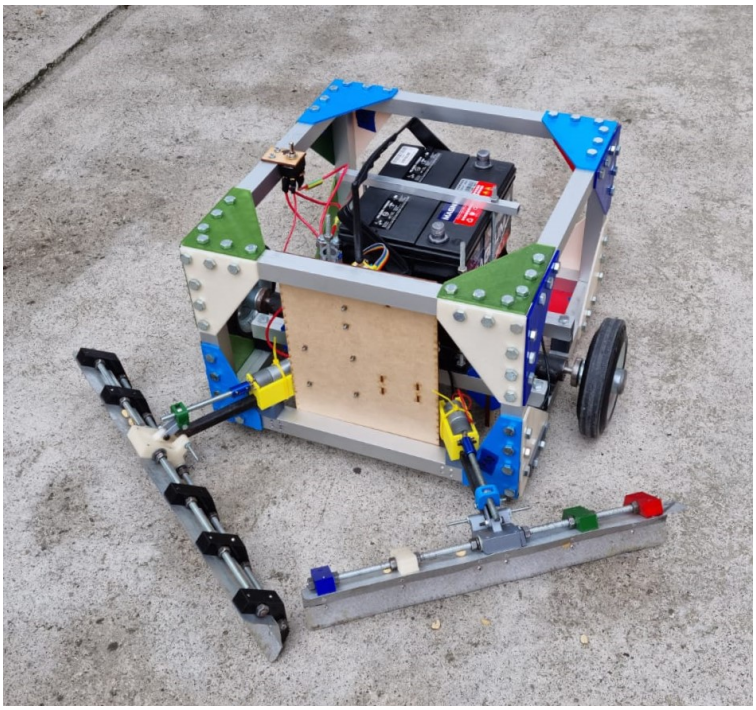


Figura 85: Robot secador de café parte delantera

CAPÍTULO 15

Listado de materiales y costos

Cantidad	Componente	Lugar de compra	Precio unitario	Total
1	DC Motor Driver 2x15A, 4.8-35V	Robotshop	$44.95	Q247.49
1	Mini IMU-9 V5	Pololu/Digikey	$15.95	Q123.30
4	DWM1001-DEV	DigiKey	$39.00	Q1,205.98
1	NRF24L01 & Modulo	Amazon	Q69.23	Q69.23
4	Porta Batería	La Electrónica	Q13.00	Q52.00
1	Arduino Mega	Electrónica DIY	Q170.00	Q170.00
2	Módulo F249	Megacef	Q25.00	Q50.00
8	Batería 18650	ABC Electrónica	Q50.00	Q400.00
2	Two wheel drive motor chassis robotics Kit	Electrónica SMD	Q125.00	Q250.00
2	Protoboards 170 puntos	Electrónica SMD	Q11.00	Q22.00
1	Conector de balance LIPO JST-XH 4S	Electrónica SMD	Q9.00	Q9.00
4	LM7805	Electrónica SMD	Q4.00	Q16.00
3	Interruptor deslizable 3 pines	Electrónica SMD	Q0.75	Q2.25
2	Metro de cable para protobard	Electrónica total	Q10.00	Q20.00
1	Push Bottom 12X12mm	Electrónica total	Q11.00	Q11.00
1	Switch de Palanca SMTS-102 SPDT	Electrónica total	Q12.00	Q12.00
4	Terminal de Bornera de 2 pines	Electrónica total	Q2.00	Q8.00
1	Capacitores cerámicos	Electrónica total	Q0.50	Q0.50
1	Resistencia de 1000ohm	Electrónica total	Q0.50	Q0.50
1	Set de 20 jumpers hembra-macho	Electrónica total	Q15.00	Q15.00
8	Tornillos cabeza plana phillips 3x12 mm	La casa del tornillo	Q1.40	Q11.20
3	Tornillos cabeza plana phillips 3x22 mm	La casa del tornillo	Q1.55	Q4.65
18	Tornillos cabeza plana, plano 3x18 mm	La casa del tornillo	Q1.68	Q30.24
4	Tuercas de seguridad zinc 6/32"	Novex	0.25	Q1.00
4	Tornillos máquina cabeza redonda 6 32x2	Novex	0.6	Q2.40
2	Tablas de MDF 22x24 cm	D-HIVE UVG	Q5.00	Q10.00
	TOTAL			**Q2,743.74**

CAPÍTULO 16

Conclusiones

1. Se disñó e implementó un robot a escala utilizando el kit *2WD Smart Robot*, en conjunto a los módulos DWM1001-DEV, Encoders, módulos NRF24L01 y el módulo DRI0018.
2. Un sistema de localización en tiempo real con módulos DWM1001 compuesto por 3 *anchors* y 1 *TAG* presenta errores de medición de posición de hasta 70 centímetros.
3. El vector promedio con 300 muestras es aproximadamente 175 % más tardado que el vector promedio con 200 muestras.
4. La función promedio redujo la dispersión de datos del eje Y medida por le sistema de localización en tiempo real en un 31.69 % cuando el robot se encuentra estático.
5. La función promedio redujo la dispersión de datos del eje X medida por le sistema de localización en tiempo real en un 27.68 % cuando el robot se encuentra estático.
6. Al desplazar el robot únicamente sobre eje X se presentó una variación de valores medidos de la posición del robot en el eje Y de hasta 496.4 centímetros.
7. Al desplazar el robot únicamente sobre eje X, se presentó una variación de valores medidos de la posición del robot en el eje Z de hasta 561 centímetros.
8. Al desplazar el robot únicamente sobre eje Y, se presentó una variación de valores medidos de la posición del robot en el eje X de hasta 471.7 centímetros.
9. Al desplazar el robot únicamente sobre eje Y, se presentó una variación de valores medidos de la posición del robot en el eje Z de hasta 329 centímetros.
10. Los motores implementados en el robot secador de café demostraron ser adecuado para los requisitos del proyecto.
11. El costo de implementar un prototipo a escala del robot secador de café con todos los módulos necesarios es de Q2743.74.

CAPÍTULO 17

Recomendaciones

1. Utilizar el promedio con n $>$ 100 mediciones de la posición al momento de querer utilizar las mediciones del *RTLS* para el sistema de control ya que este presenta una variación de 10 centímetros cómo máximo según el fabricante. Sin embargo, se demostró que este valor es mayor.
2. Adquirir por lo menos 2 DWM1001-DEV más para implementar el *RTLS* compuesto por 4 *anchor* y un *TAG* como mínimo.
3. Implementar un filtro de Kalman extendido al realizar la fusión de sensores utilizando la odometría obtenida con encoders, la posición del *TAG* medida del *RTLS* y la detección de la dirección de movimiento al utilizar la unidad de medición inercial (*IMU*).
4. Realizar pruebas de funcionamiento del *RTLS* colocando a los *anchors* a diferentes alturas sobre el nivel del suelo para comprobar si afecta el campo de visión entre ellos.
5. Diseñar un perímetro estándar de MxN centímetros en donde se colocarán siempre los *anchors* en el mismo orden y configurar a los *anchors* a esas distancias para no tener que estar utilizando la aplicación DRTLS cada vez que se quieran hacer pruebas.
6. Al implementar un sistema de localización en tiempo real con más *anchors*, verificar si es necesario mantener la velocidad de recopilación de datos del *RTLS* o si se puede incrementar esta velocidad.
7. Diseñar otra placa de alimentación para los *anchors* en la cual se implemente otro regulador de voltaje que tenga una menor perdida de potencia para alimentar a los módulos DWM1001.
8. Triangular y encontrar matemáticamente la posición del robot con base a las distancias de cada *anchor* hacia el *TAG* utilizando únicamente el primer cuadrante del plano cartesiano. Tomando como punto de origen al *Anchor1* y al *Anchor2* alineado en el eje X del *Anchor1*
9. Realizar más pruebas de localización de la posición del robot a partir de las distancias utilizando el comando DWM_POS_GET con el código APi “0x0C” “0x00”

CAPÍTULO 18

Bibliografía

[1] J. Sáenz, "*Diseño de Sistema Electrónico y de Control de un Robot Móvil de Secado de Café*," Universidad del valle de Guatemala, 2020.

[2] Ripipsa. (2019). "*Robots autónomos: que son, cómo funcionan y qué ventajas ofrecen*," dirección: https://ripipsacobots.com/robots-autonomos/.

[3] TESLA. (2021). "*Modelo S*," dirección: https://www.tesla.com/models.

[4] FRE. (2021). "*International Field Robot Event - Event*," dirección: https://www.fieldrobot.com/event/index.php/info/.

[5] ——, (2021). "*International Field Robot Event*," dirección: https://www.fieldrobot.com/event/.

[6] DLG-Feldtage. (2020). "*Field Robots - Fiction or Reality?*" Dirección: https://www.youtube.com/watch?v=bHzV4xi0JQk.

[7] ——, (2018). "*Field Robots Event proceedings 2018*," dirección: https://www.fieldrobot.com/event/wp-content/uploads/2019/06/Proceedings_FRE2018.pdf.

[8] CIG. (2019). "*Exportaciones: principales productos de exportación*," dirección: https://cig.industriaguate.com/inversionistas/estadisticas-macroeconomicas/exportaciones-principales-productos-de-exportacion/.

[9] datosmacro. (2020). "*Guatemala - Exportaciónes de Mercancías*," dirección: https://datosmacro.expansion.com/comercio/exportaciones/guatemala.

[10] FórumCafé. (2018). "*EL CAFÉ DE GUATEMALA*," dirección: http://www.forumdelcafe.com/noticias/cafe-guatemala.

[11] Mineco. (2019). "*Café de Guatemala*," dirección: https://www.mineco.gob.gt/sites/default/files/cafe_en_guatemala.pdf.

[12] Telectrónica. (2014). "*¿Qué es RTLS (Real Time Location System)?*" Dirección: https://telectronica.com/que-es-rtls-real-time-location-system/#:~:text=RTLS%20es%20el%20uso%20de,un%20enfoque%20com%C3%BAn%2C%20la%20triangulaci%C3%B3n..

[13] BIZDATA.IO. (2020). "*Real Time Location System (RTLS)*," dirección: https://www.bizdata.io/base-de-conocimiento/vision-general/rtls/.

[14] Anónimo. (2011). "*Tecnologías de acceso celular*," dirección: https://sites.google.com/site/informaticaycomunicacion2011/home/telefonia-movil/tecnologias-de-acceso-celular.

[15] UBITEC. (2019). "*¿Cómo funciona el GPS?*" Dirección: https://ubitec.mx/como-funciona-el-gps/#:~:text=El%20receptor%20GPS%20recibe%20una,lejos%20est%C3%A1%20de%20cada%20sat%C3%A9lite.

[16] Moto1Pro. (2017). "*Unidad de medición inercial: IMU para motos de calle*," dirección: https://www.moto1pro.com/reportajes-motos/unidad-de-medicion-inercial-imu-para-motos-de-calle#:~:text=La%20Unidad%20de%20Medici%C3%B3n%20Inercial,velocidad%2C%20orientaci%C3%B3n%20y%20fuerzas%20gravitacionales.&text=Su%20base%20es%20un%20sistema,un%20veh%C3%ADculo%20en%20tres%20ejes.

[17] Blascarr. (2017). "*Introducción al IMU - Sistema de navegación inercial*," dirección: https://www.blascarr.com/lessons/introduccion-al-imu-sistemas-de-navegacion-inercial/.

[18] DYNAPAR. (2019). "*Encoder funcionamiento: el mejor resumen de cómo funciona un encoder*," dirección: https://www.dynaparencoders.com.br/blog/es/encoder-funcionamiento/.

[19] Demaquinasyherramientas. (2017). "*¿Qué es un encoder, cuáles son sus tipos y para qué sirven?*" Dirección: https://www.demaquinasyherramientas.com/mecanizado/encoder-tipos.

[20] mECAFENIX. (2018). "*¿Que es un optoacoplador y como funciona?*" Dirección: https://www.ingmecafenix.com/electronica/optoacoplador/.

[21] Andromina. (2017). "*Encoder and Arduino.Tutorial about the IR speed sensor module with the comparator LM393 (Encoder FC-03)*," dirección: http://androminarobot-english.blogspot.com/2017/03/encoder-and-arduinotutorial-about-ir.html.

[22] L. union. (2015). "*Explicación PUENTE H - Funcionamiento - Inversión de giro con transistores*," dirección: https://www.youtube.com/watch?v=CdIQL2LQyWA.

[23] S. electrónica. (2021). "*DRI0018, DRIVER DE 2 MOTORES DC A 15A*," dirección: https://www.sigmaelectronica.net/producto/dri0018/.

[24] T. Instrument. (2020). "*TXS0108E 8-Bit Bi-directional, Level-Shifting, Voltage Translator for Open-Drain and Push-Pull Applications*," dirección: https://www.ti.com/lit/ds/symlink/txs0108e.pdf?ts=1619584489503&ref_url=https%253A%252F%252Fwww.google.com%252F.

[25] B. Lutkevich. (2021). "*Microcontroller (MCU)*," dirección: https://internetofthingsagenda.techtarget.com/definition/microcontroller.

[26] Sherlin.Xbot.es. (2014). "*¿QUÉ ES UN MICROCONTROLADOR?*" Dirección: http://sherlin.xbot.es/microcontroladores/introduccion-a-los-microcontroladores/que-es-un-microcontrolador.

[27] Arduino. (2021). "*Arduino Mega*," dirección: https://store.arduino.cc/products/arduino-mega-2560-rev3.

[28] B. internet Tec. (2018). "*[Enciclopedia Online Gratuita] Diccionario de Internet y Tecnologías de la Información y la Comunicación (TIC):*" dirección: https://www.paraisodigital.org/internet/ll-odometria-que-es-definicion-y-significado-descargar-videos-y-fotos.html.

[29] wordpress. (2020). "*¿Qué es la comunicación serie?*" Dirección: http://robots-argentina.com.ar/didactica/que-es-la-comunicacion-serie/#comments.

[30] Grabcad. (2021). "*3D Printing CAD Collaboration Software*," dirección: https://grabcad.com/.

[31] thingiverse. (2021). "*MakerBot Thingiverse*," dirección: https://www.thingiverse.com/.

[32] KlimukVI. (2018). "*Wheel D65x25*," dirección: https://grabcad.com/library/wheel-d65x25-1.

[33] R. Teixeira. (2014). "*GEAR MOTOR DG01D-A130*," dirección: https://grabcad.com/library/gear-motor-dg01d-a130-1.

[34] D. Belarus. (2018). "*Speed Encoder for TT motor 20 lattice*," dirección: https://www.thingiverse.com/thing:3197326/files.

[35] M. A. R. Castaño. (2020). "*Arduino MEGA 2560*," dirección: https://grabcad.com/library/arduino-mega-2560-6.

[36] J. Antonio. (2014). "*Dc motor driver shield*," dirección: https://grabcad.com/library/dc-motor-driver-shield-1.

[37] E. Chugay. (2019). "*TO-220 transistor package*," dirección: https://grabcad.com/library/to-220-transistor-package-1.

[38] A. C. Ccollcca. (2016). "*Ball Caster Metal - 3/8"(Rueda loca)*," dirección: https://grabcad.com/library/ball-caster-metal-3-8-rueda-loca-1.

[39] DECAWAVE. (2016). "*DWM1001 DEVELOPMENT BOARD*," dirección: https://www.decawave.com/product/dwm1001-development-board/.

[40] naylampmechatronics. (2016). "*TUTORIAL BÁSICO NRF24L01 CON ARDUINO*," dirección: https://naylampmechatronics.com/blog/16_tutorial-basico-nrf24l01-con-arduino.html.

[41] doxygen. (2012). "*RF24 Class Reference*," dirección: https://maniacbug.github.io/RF24/classRF24.html.

CAPÍTULO 19

Anexos

En el siguiente enlace se presentan los códigos desarrollados, modelos CAD, imágenes, documentos, hojas de datos, códigos API, videos y demas cosas utilizadas/realizadas en el proyecto.

https://gitlab.com/kekellner/robot-de-cafe-alvaro-torres