INSTITUTO TECNOLÓGICO SUPERIOR DE ATLIXCO

Organismo Público Descentralizado del Gobierno del Estado de Puebla

INGENIERÍA MECATRÓNICA

Diseño e Implementación de un Robot Seguidor de Línea Autónomo Profesional Basado en un Microcontrolador ARM Cortex M4.

Reporte de actividades realizadas en la empresa Instituto Tecnológico Superior De Atlixco durante el periodo comprendido de 09 del 2020 hasta 01 del 2021 para su acreditación.

Presenta:

Jesús Balbuena Palma

Asesores:

Ing. Raúl Eusebio Grande
Dra. Mariana Natalia Ibarra Bonilla

Atlixco, Pue. Febrero 2021.

Agradecimientos

Le agradezco a Dios por acompañarme y guiarme a lo largo de toda mi vida, por ser mi apoyo y fortaleza en los momentos de debilidad, pero sobre todo por brindarme una vida llena de aprendizajes, experiencias y felicidad.

Le doy gracias a mi madre Patricia por ser esa persona que siempre ha estado en todo momento conmigo apoyándome y guiándome hasta convertirme en el hombre que soy. A mi padre José que ha hecho lo que ha podido para apoyarme. A mi hermana Monserrat Guadalupe por darme sus sabios consejos y por cuidarme. A mi hermana María de los Ángeles por ser mi cómplice y compañera de muchas aventuras. Pero principalmente, a mi hermano José Ángel por haberme apoyado a lo largo de toda la carrera y por ser parte fundamental del desarrollo de este proyecto.

Gracias a la Fundación BBVA por haberme apoyado a lo largo de todos mis estudios, por haberme hecho crecer profesionalmente, pero sobre todo por haber cambiado significativamente el rumbo de mi vida. También quiero agradecer a la organización YoDOC, quien sin su apoyo no hubiera sido posible la creación de este proyecto, el cual impactará positivamente en los estudiantes del Tecnológico de Atlixco.

Me gustaría agradecer a la Dra. Mariana N. Ibarra Bonilla por todo su apoyo incondicional que me brindó a lo largo de la carrera y quien confió en mi para la realización de este proyecto. También al Ing. Raúl Eusebio Grande por su apoyo y confianza que me brindó para el desarrollo de este proyecto. Así como a todos mis docentes de la ingeniería que me brindaron su tiempo, conocimientos y vivencias para formar mi educación universitaria.

Resumen

El presente proyecto tiene como finalidad el diseñar y construir un Robot Seguidor de Línea utilizando un microcontrolador ARM Cortex M4. Con lo cual se busca que los estudiantes del Tecnológico de Atlixco puedan participar de forma competitiva a nivel profesional en eventos nacionales e internacionales, tales como: Talent Land, Robomatrix, Roboduca, Robot Challenge, Intelibots Tournament, Robotic People Fest, Guerra de Robots del IPN, entre otros más; y así fortalecer su formación profesional.

Esta plataforma surge de la necesidad de contar con un hardware que reúna los requerimientos necesarios de un Robot Seguidor de Línea Velocista Pro con Turbina, el cual pueda competir de forma profesional contra participantes de otros países. Siendo uno de los objetivos planteados la reducción de los costos de fabricación de los Robots Seguidores de Línea, cuyo valor en el mercado suele rondar entre los $5,000 a $15,000 por unidad.

Para el desarrollo de esta plataforma robótica se eligió el software KiCad, ya que es un programa bastante completo utilizado en la automatización del diseño electrónico. KiCad facilita el diseño de esquemáticos para circuitos electrónicos y su conversión a placa de circuito impreso (del inglés: Printed Circuit Board, PCB). Así mismo, se eligió como entorno de programación el IDE de STM32CubeIDE. Con esto, se busca optimizar al máximo el control del sistema embebido con el que cuenta el Robot producto de este proyecto.

Por otro lado, se muestran adicionalmente los resultados que se obtuvieron de la participación en el ROBOTIC PEOPLE FEST Feria Universitaria Colombia 2020 y en el Intelibots Tournament Virtual 2020, los cuales se realizaron de manera virtual debido a la contingencia sanitaria ocasionada por el COVID-19.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Introducción

La robótica móvil es una de las áreas de la robótica que más ha presentado avances en los últimos años. Es una nueva generación de avanzada tecnología que puede usarse en un amplio rango de sectores. Configuraciones disruptivas en la mecánica, estructuras modernas de control y navegación, desarrollo de vehículos no tripulados y nuevas aplicaciones han transformado lo que conocíamos acerca de los sistemas robóticos. El pensar que en algunos años las máquinas realizarán el trabajo operativo de muchas personas no es ciencia ficción, sino una realidad, consecuencia de la denominada cuarta revolución industrial. Un conjunto de tecnologías tales como la robótica, la inteligencia artificial y el internet de las cosas; las cuales cambiarán radicalmente la forma en como estamos acostumbrados a vivir.

En el sector educativo, se han desarrollado plataformas robóticas como un medio para incentivar la investigación y la innovación tecnológica. A su vez, han surgido una gran variedad de encuentros estudiantiles donde los participantes comparten experiencias que les ayudan a fortalecer sus habilidades en áreas tales como: la programación de sistemas embebidos, electrónica, mecánica y control digital; las cuales son materias de suma importancia para el desarrollo de un ingeniero mecatrónico.

Dentro del ámbito académico-deportivo, los robots móviles se emplean en una gran variedad de competencias y categorías, como, por ejemplo: Robot Laberinto, Drone Racing, Robot Futbol, Robot Seguidor de Línea y Robot Sumo, entre otro más. Siendo una de las más populares, la categoría de Robots Seguidores de Línea Velocistas, en inglés Line Follower Robot (LFR).

La gran demanda por parte de las escuelas en participar en los múltiples torneos de robótica, ha dado como resultado una mayor exigencia en el desempeño de las diferentes plataformas robóticas y sus esquemas de competencia. Este proyecto plantea el diseño, implementación y control de un Robot Seguidor de Línea Autónomo Profesional, basándose en un microcontrolador ARM Cortex M4. Ante lo cual, se busca optimizar el funcionamiento de estos robots móviles y disminuir sus costes de fabricación.

El presente documento se divide en cuatro capítulos. El primer capítulo consta de las generalidades del proyecto, planteamiento del problema, objetivos (generales y específicos) y justificación. El segundo capítulo aborda el marco teórico en el cual estará basado el diseño y control del robot. En el tercer capítulo se encuentra el desarrollo de nuestra plataforma robótica, el cual va desde el diseño de la PCB hasta la programación del sistema embebido del robot. Seguidamente, en el capítulo cuatro se pueden observar los resultados obtenidos de nuestro sistema robótico. Y finalmente, se tienen las conclusiones y recomendaciones que servirán como aporte para futuras investigaciones y el continuo mejoramiento de los Robots Seguidores de Línea Velocistas.

Capítulo 1: Generalidades del Proyecto

En este capítulo se dará a conocer una breve descripción del Instituto Tecnológico Superior de Atlixco, así como también de la División de Ingeniería Mecatrónica. Además, se mostrará el planteamiento del problema, los objetivos, la justificación y los alcances que tendrá este proyecto.

1.1 Historia del Instituto Tecnológico Superior de Atlixco (ITSA)

El Instituto Tecnológico Superior de Atlixco (ver figura 1.1), es un organismo público descentralizado del estado de Puebla, que surge como un instrumento de desarrollo para la región del valle Atlixco-Matamoros; iniciando sus labores en septiembre de 1998, impartiendo dos carreras a nivel licenciatura en las áreas de ingeniería Electromecánica y Bioquímica. En septiembre del 2000 se incorpora a la oferta académica la carrera de Ingeniería en Sistemas Computacionales, para agosto de 2002 la carrera de Ingeniería Industrial y en agosto de 2004 la carrera de Ingeniería Mecatrónica. En agosto de 2015 se autoriza y se incorpora la carrera de Licenciatura en Gastronomía, teniendo así 6 carreras hasta el día de hoy [1].

Figura 1.1. Logo del Instituto Tecnológico Superior de Atlixco (ITSA).

- **Misión**: Somos una Institución de Educación Superior Tecnológica que forma profesionistas en ciencia y tecnología, capaces de transformar su entorno, contribuyendo al bienestar social del medio ambiente [2].
- **Visión**: Ser una institución líder en el ámbito educativo, certificada y acreditada en los mayores estándares de calidad, reconocida por la formación humana y profesional de las nuevas generaciones que impactan en el desarrollo social sustentable [2].

- **Valores**: Liderazgo, Responsabilidad Social, Compromiso, Perseverancia, Honestidad y Ética [2].

El desarrollo de este proyecto se llevó a cabo en el área de la División de Ingeniería Mecatrónica del ITSA, cuya dirección se encuentra a cargo del Ing. Alejandro Lozano Montiel. La ubicación de las instalaciones del Tecnológico de Atlixco (ver figura 1.2), es Prolongación Heliotropo, No. 1201, Colonia Vista Hermosa, Atlixco, Puebla.

Figura 1.2. Instalaciones del Instituto Tecnológico Superior de Atlixco (ITSA).

1.2 División de la Ingeniería Mecatrónica

- **Objetivo de la Carrera:**

  Formar profesionales con la capacidad de crear, diseñar, modelar y construir tanto herramientas, equipo, así como administrar procesos que involucren dispositivos electrónicos y mecánicos, seleccionando los materiales más adecuados y realizando la programación de software necesaria, para hacer frente a la creciente demanda de sistemas automatizados e inteligentes (ver figuras 1.3 y 1.4) [3].

- **Objetivos Educacionales:**

  Identificar, analizar, formular, desarrollar y resolver problemas de ingeniería aplicando ciencias básicas y materias de especialidad. Ejecutar, analizar y minimizar los procesos de diseño programación e ingeniería que resulten en proyectos que cumplen las necesidades de la región. Desarrollar y conducir la investigación adecuada; analizar e interpretar datos y utilizar las herramientas

aprendidas en la ingeniería mecatrónica para establecer conclusiones y recomendaciones. Comunicarse efectivamente con los diferentes sectores por parte de estudiantes, estudiantes en educación dual, residentes y egresados para poder dar respuestas efectivas a las necesidades. Reconocer sus responsabilidades éticas y profesionales en situaciones relevantes para la ingeniería mecatrónica y realizar juicios informados que deben considerar el impacto de las soluciones de ingeniería en los contextos público, social y privado. Reconocer la necesidad permanente de conocimiento como educación continua, tener la habilidad para buscar, evaluar, desarrollar, integrar y aplicar el conocimiento adecuadamente. Trabajar efectivamente en equipos que establezcan objetivos, planeación, cumplimiento de metas analizando riesgos e incertidumbre [3].

Figura 1.3. Logo de la División de Ingeniería Mecatrónica del ITSA.

Figura 1.4. Docentes y Estudiantes de la Carrera de Ingeniería Mecatrónica.

1.3 Planteamiento del Problema

La categoría de Robots Seguidores de Línea Velocistas es una de las más importantes en la mayoría de los torneos de robótica; generando así una gran demanda en participación por parte de las escuelas. Dando como resultado, una mayor exigencia en el desempeño de las plataformas robóticas utilizadas en esta categoría. Siendo uno de los principales problemas el diseño de estos robots al tener que utilizar múltiples módulos de hardware independientes; tal es el caso del uso de una Tarjeta de Desarrollo, Controladores Puente H, Reguladores de Voltaje, entre otros. Dando como consecuencia una reducción considerable en la eficiencia del sistema robótico.

Por otro lado, el Instituto Tecnológico Superior de Atlixco actualmente no cuenta con plataformas robóticas tipo Seguidor de Línea Autónomo Profesional. Estas plataformas permiten a los estudiantes el aprendizaje e investigación acerca de la robótica móvil. Al no tener una plataforma robótica de este estilo, los estudiantes no pueden participar en los múltiples torneos de robótica que existen a nivel nacional e internacional; tales como: Talent Land, Robomatrix, Roboduca, Robot Challenge, Guerra de Robots del IPN, entre otros más. Cabe resaltar que, en las últimas ediciones de los torneos de robótica nacionales, la participación de los equipos poblanos ha sido casi nula.

1.4 Objetivos

1.4.1 Objetivo General

Diseñar e implementar una plataforma robótica tipo Autonomous Line Follower utilizando un microcontrolador ARM Cortex M4, para que los estudiantes del Tecnológico de Atlixco puedan participar de forma competitiva a nivel profesional en los diferentes torneos de robótica que existen tanto nacionales como internacionales.

1.4.2 Objetivos Específicos

- Diseñar el circuito electrónico del Robot Seguidor de Línea en el Software de Diseño KiCad.
- Reducir los costos de fabricación de los Robots Seguidores de Línea, cuyo valor en el mercado suele rondar entre los $5,000 a $15,000 por unidad.

- Diseñar e implementar un algoritmo de control el cual sea capaz de establecer la autonomía del robot móvil.
- Realizar las pruebas de validación del desempeño óptimo del Robot Seguidor de Línea.
- Participar en el ROBOTIC PEOPLE FEST Feria Universitaria y en el Intelibots Tournament Virtual 2020, los cuales se realizarán de manera virtual debido a la contingencia sanitaria ocasionada por el COVID-19.

1.5 Justificación

A lo largo de mi experiencia como estudiante en el Instituto Tecnológico Superior de Atlixco, siempre he podido ver una gran pasión por los estudiantes hacia la robótica. Sin embargo, una de las grandes limitantes que usualmente se presenta para poder llevar a cabo la creación de estas plataformas robóticas es el factor económico. Por si sola, la carrera de Ingeniería Mecatrónica es algo costosa puesto que se requiere de una gran inversión en materiales, a lo cual, los alumnos se ven en la necesidad de elegir entre pagar sus materiales escolares o poder construir robots de competencia. Lo que nos lleva a que los estudiantes desistan en la creación de robots y por tanto en dejar pasar la oportunidad de poder participar en los diferentes torneos de robótica.

Este proyecto ha sido creado con la finalidad de que los estudiantes puedan fortalecer sus habilidades en áreas tales como la programación de sistemas embebidos, electrónica, mecánica y control digital; las cuales son materias de suma importancia para el desarrollo de un ingeniero mecatrónico. Así mismo, se busca incentivar a los estudiantes para que puedan participar en los torneos de robótica tanto nacionales como internacionales.

Para el desarrollo de este proyecto se ha decidido utilizar como sistema embebido de los Robots Seguidores de Línea un microcontrolador STM32F407VGT6 de STMicroelectronics. El cual integra un procesador ARM Cortex-M4, cuya tecnología es utilizada actualmente tanto en teléfonos móviles, tabletas, sistemas de sonido, etc. Y cuyos periféricos incluidos, tales como módulos PWM, puertos de comunicación, entre

otros, nos permiten optimizar los recursos necesarios para el desempeño óptimo de estos robots.

En cuanto al software de diseño, se ha decidido utilizar KiCad puesto que es un paquete de software de código abierto para la automatización del diseño electrónico. El cual facilita el diseño de esquemáticos para circuitos electrónicos y su conversión a placa de circuito impreso (Printed Circuit Board, PCB) [4].

1.6 Alcances

Se desarrollarán dos plataformas completas del robot tipo Autonomous Line Follower para los estudiantes de la Ingeniería en Mecatrónica del Tecnológico de Atlixco (específicamente para la Rama Estudiantil IEEE ITSA y el Capítulo Estudiantil Mecatrolynxs Robotic). Así mismo, se construirá un Robot tipo Follower Line para la División de Ingeniería Mecatrónica que pueda ser utilizado en exhibiciones de la carrera. Todo esto gracias al apoyo económico brindado por parte de la noble y filantrópica organización YoDOC, la cual destino un fondo para este proyecto que asciende a los $11,000.00 MXN.

Capítulo 2: Marco Teórico

En este capítulo se abordará el marco teórico en el cual estará basado el diseño y control de nuestro Robot Seguidor de Línea Autónomo. Para ello, este capítulo se encuentra dividido en cuatro secciones:

- Tópicos de Robótica.
- Robot Seguidor de Línea Autónomo.
- KiCad.
- STM32CubeIDE.

2.1 Tópicos de Robótica

2.1.1 Definición de un Robot Inteligente

Un robot considerado inteligente deberá ser una máquina autónoma capaz de extraer selectivamente información de su entorno. La funcionalidad autónoma se refiere a que el robot puede operar auto contenido en condiciones razonables sin necesidad de recurrir a un operador humano. La autonomía incluye que el robot pueda utilizar el conocimiento sobre el mundo que le rodea para moverse de forma segura, útil e intencionada.

2.1.2 Paradigmas de la Robótica

Un paradigma es una filosofía o un conjunto de supuestos y/o técnicas que caracterizan un enfoque a una clase de problemas. Es tanto una forma de ver el mundo como un conjunto implícito de herramientas para resolver problemas. Ningún paradigma es correcto; más bien, algunos problemas parecen más adecuados para diferentes enfoques. Aplicar el paradigma correcto facilita la resolución de problemas [5]. Actualmente existen tres paradigmas para organizar la inteligencia en los robots:

- Paradigma Reactivo.
- Paradigma Jerárquico.
- Paradigma Hibrido.

Los paradigmas se describen de dos formas:

- Por la relación entre las tres primitivas de robótica comúnmente aceptadas: Percepción, Planificación y Actuación.
- Por la forma en que los datos de los sensores son procesados y distribuidos a través del sistema.

En la robótica existen tres funciones básicas ampliamente aceptadas:

- Percepción: Es cuando una función toma información de los sensores del robot y produce una salida útil para otras funciones.
- Planificación: Es cuando una función toma información (ya sea de sensores o de su propio conocimiento sobre cómo funciona el mundo) y produce una o más tareas para que las realice el robot.
- Actuación: Es cuando las funciones producen comandos de salida hacia los actuadores.

Paradigma Reactivo

En este paradigma el robot solo cuenta con dos primitivas, las cuales son la percepción y la actuación (ver figura 2.1) [5]. Las relaciones establecidas en este tipo de paradigma se observan en la tabla 1, siguiendo el flujo de las flechas.

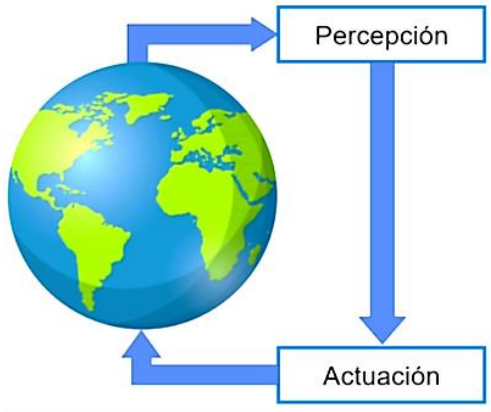

Figura 2.1. Paradigma Reactivo.

Tabla 1. Relaciones del Paradigma Reactivo.

Primitivas del Robot	Entradas	Salidas
Percepción	Datos de los sensores	Información de los sensores
Actuación	Información de los sensores	Comandos para los actuadores

Paradigma Jerárquico

En este paradigma el robot cuenta con las tres primitivas, las cuales son la percepción, la planificación y la actuación (ver figura 2.2) [5]. Las relaciones establecidas en este paradigma entre las tres primitivas son jerárquicas. Este comportamiento se puede observar en la tabla 2, siguiendo el flujo de las flechas.

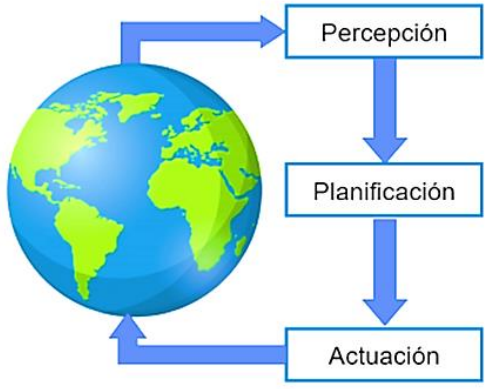

Figura 2.2. Paradigma Jerárquico.

Tabla 2. Relaciones del Paradigma Jerárquico.

Primitivas del Robot	Entradas	Salidas
Percepción	Datos de los sensores	Información de los sensores
Planificación	Información (sensores y cognitiva)	Directivas
Actuación	Directivas	Comandos para los actuadores

Paradigma Híbrido

Este paradigma es el resultado de la combinación de los paradigmas reactivo y jerárquico (ver figura 2.3) [5], en el cual el robot puede decidir como descomponer las tareas en subtareas. En la tabla 3 se muestran las relaciones que se establecen entre las primitivas de este paradigma, siguiendo el flujo de las flechas.

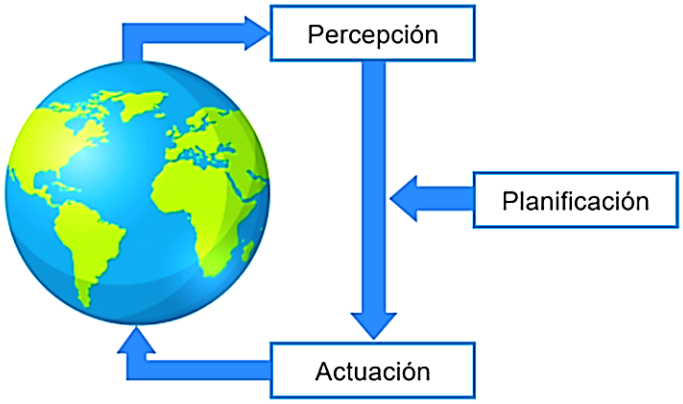

Figura 2.3. Paradigma Híbrido.

Tabla 3. Relaciones del Paradigma Híbrido.

Primitivas del Robot	Entradas	Salidas
Planificación	Información (sensores y cognitiva)	Directivas
Percepción – Actuación (Comportamiento)	Datos de los sensores	Comandos para los actuadores

2.1.3 Locomoción Mediante Ruedas en los Robots Móviles

El sistema de locomoción más empleado en la robótica es el basado en ruedas. Los robots móviles con ruedas son más sencillos y fáciles de construir, y, por ende, la carga que pueden transportar es mayor. Los sistemas basados en cadenas o en patas, son más complejos de diseñar y, generalmente, suelen utilizar una mayor cantidad de piezas.

En los robots móviles con ruedas, se destacan tres sistemas de locomoción, estos son: Diferencial, Triciclo y Ackerman.

Sistema de Locomoción: Diferencial

Este es el sistema más sencillo en muchos aspectos. Consta de dos ruedas situadas diametralmente opuestas en un eje perpendicular a la dirección del robot (ver figura 2.4.a). Estas ruedas permiten al robot avanzar o retroceder, girar sobre sí mismo, o seguir una trayectoria. El direccionamiento viene dado por la diferencia de velocidades de las ruedas laterales [6]. La tracción se consigue también con estas mismas ruedas. La configuración de direccionamiento diferencial es la que se ha estandarizado para el desarrollo de los Robots Seguidores de Línea Velocistas (ver figura 2.4.b).

a) b)

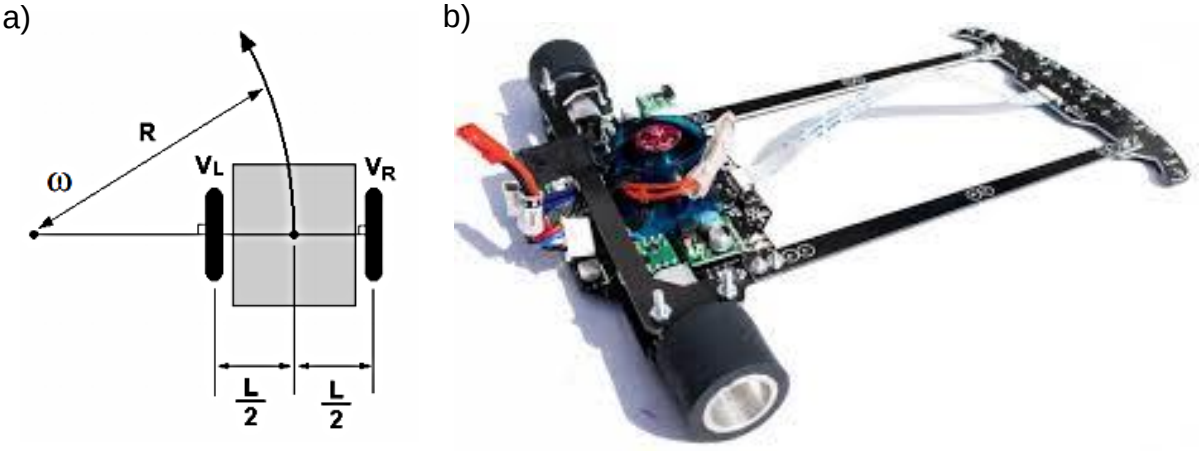

Figura 2.4. a) Sistema de Locomoción Diferencial, b) Ejemplo de un Robot Seguidor de Línea con Locomoción Diferencial.

Sistema de Locomoción: Triciclo

Este sistema de locomoción se ilustra en la figura 2.5. La rueda delantera sirve tanto para la tracción como para el direccionamiento. El eje trasero, con dos ruedas laterales, es pasivo y sus ruedas se mueven libremente. La maniobrabilidad es mayor que en los sistemas de locomoción diferencial, pero puede presentar problemas de estabilidad en terrenos difíciles. El centro de gravedad tiende a desplazarse cuando el vehículo se desplaza por una pendiente, causando la pérdida de tracción [6].

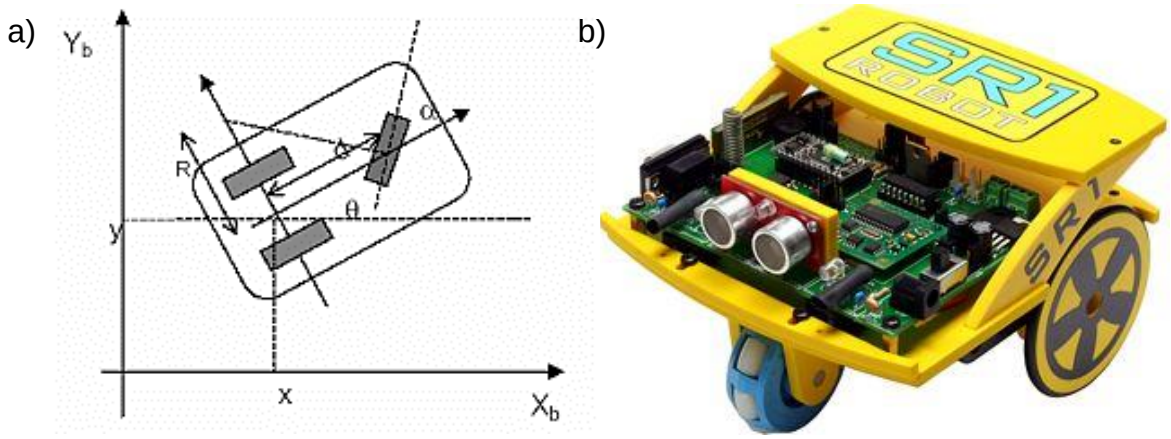

Figura 2.5. a) Sistema de Locomoción: Triciclo, b) Ejemplo de un Robot Multifunciones con Locomoción de Triciclo.

Sistema de Locomoción: Ackerman

Es el utilizado en vehículos de cuatro ruedas convencionales. Este sistema de locomoción se ilustra en la figura 2.6. Este sistema consta de cuatro ruedas, las cuales las traseras son las de tracción y las delanteras no motrices corresponden a las ruedas de dirección [6]. Esta configuración permite una gran estabilidad, pero en un robot seguidor de línea velocista no es suficiente la estabilidad, sino también la velocidad, y esta configuración no permite una respuesta rápida de cambio de dirección estable a grandes velocidades.

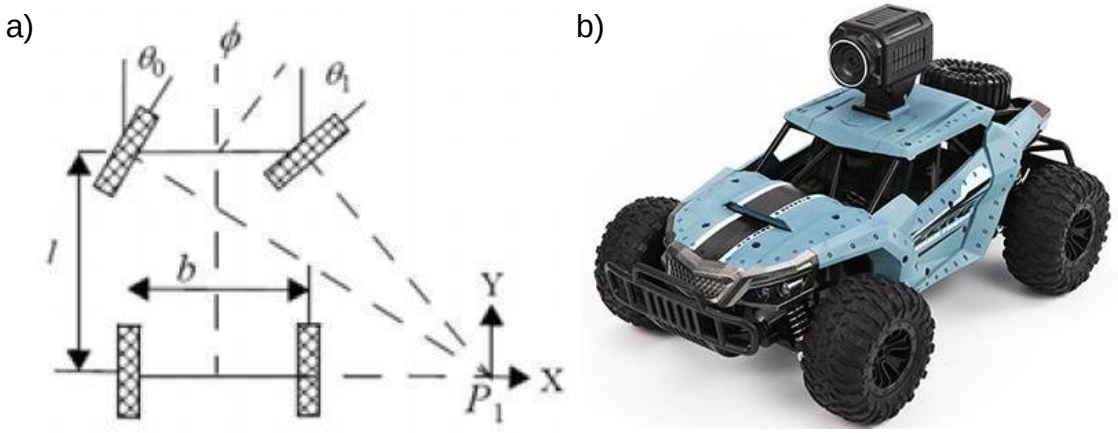

Figura 2.6. a) Sistema de Locomoción: Ackerman, b) Ejemplo de un Robot con Locomoción Tipo Ackerman.

2.1.4 Control PD

Un controlador PD (proporcional-diferencial) es un elemento de transferencia de un sistema de control de bucle cerrado que comprende componentes de elemento tanto P como D. El componente diferencial responde a la velocidad en la que el error de control cambia. El valor es multiplicado por el coeficiente de acción-derivada K_D y sumado al componente P (lo que, por su parte, actúa proporcionalmente en un error de control específico). Como resultado, el controlador PD puede responder a un error de control inminente y, por lo tanto, lograr una acción derivada durante el proceso de control [7].

El controlador PD es un controlador rápido y puede incluso corregir bucles de control con doble integración. Sin embargo, al igual que el controlador P, no puede corregir completamente un error de control. La ecuación del controlador es:

$$u(t) = K_P e(t) + K_D \frac{de(t)}{dt} \tag{1}$$

Utilizando la aproximación rectangular del control PD, tenemos lo siguiente:

- Término Proporcional $$K_P e(t) = K_P e(n) \tag{2}$$
- Término Derivativo $$K_D \frac{de(t)}{dt} = K_D \frac{e(n) - e(n-1)}{T} \tag{3}$$

Obteniendo de esta forma el algoritmo de posición del control PD [8], el cual consta de los siguientes parámetros:

$$u(n) = K_P e(n) + K_D \frac{e(n) - e(n-1)}{T} \tag{4}$$

Donde:

- $u(n)$: Es el valor de salida después de aplicar el control PD al sistema.
- K_P , K_D: Son las ganancias utilizadas para el control PD.
- T: Es el periodo de muestreo.
- $e(n)$: Es el error actual muestreado.
- $e(n-1)$: Es el error anterior muestreado.

2.2 Robot Seguidor de Línea Autónomo

Los robots seguidores de línea (ver figura 2.7), son máquinas móviles capaces de detectar y seguir una línea, la cual se encuentra ubicada en el suelo de una superficie [9]. Usualmente, la trayectoria que el robot debe de recorrer es marcada por una línea negra sobre una superficie blanca, lo cual permite obtener un gran contraste entre los dos colores. Sin embargo, existen también torneos de robótica donde la superficie de fondo es de color negro y la trayectoria que debe seguir el robot es marcada con una línea blanca.

Los seguidores de línea capturan la posición en la que se encuentran mediante la utilización de fotorreflectores. Para poder seguir la línea correctamente, estos robots deben implementar diferentes tipos de algoritmos de control, que logren mantener el robot sobre la línea y recorrer la trayectoria completa en el menor tiempo posible.

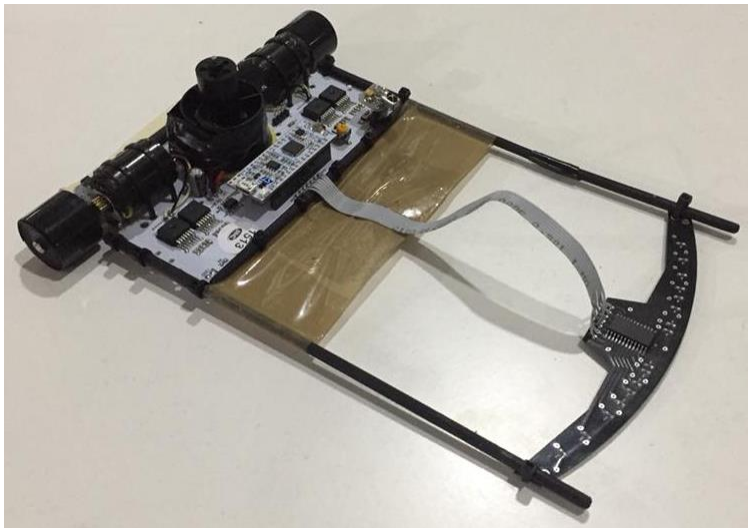

Figura 2.7. Robot Seguidor de Línea Velocista: CC-TOM (RoboChallenge 2019, Rumania).

2.2.1 Avances en el Desarrollo de los Robots Seguidores de Línea

Hoy por hoy, existe un gran avance en el desarrollo de los Robots Seguidores de Línea, esto gracias a la innovación que se ha tenido en los sistemas embebidos y la popularidad en aumento que ha tenido el “open source” durante la última década. Hasta ahora, los cambios de hardware que se realizaban en el diseño de estos robots, habían sido sin mucha trascendencia a excepción del arreglo de la barra de sensores.

Dado el diseño que presentaban los robots seguidores anteriormente, uno de los mayores problemas que se tenía era la inestabilidad a altas velocidades, esta dificultad se solucionó con la utilización de turbinas de succión para aumentar la adherencia del robot a la superficie.

Los microcontroladores que se usaban típicamente fueron remplazados por MCU de mayores capacidades, para la lectura y procesamiento de las señales provenientes de los sensores y el control general del robot.

2.2.2 Estructura del Robot Seguidor de Línea

Las partes fundamentales que conforman a un robot seguidor de línea son (ver figura 2.8): el Chasis (la propia PCB en algunos casos), la Barra de Sensores Ópticos, un Microcontrolador, un Par de Motores junto con sus Rines y Llantas, Controladores Puente H, un Arrancador, Reguladores de Voltaje, una Turbina EDF y su Controlador ESC, asi como también una Bateria LiPo.

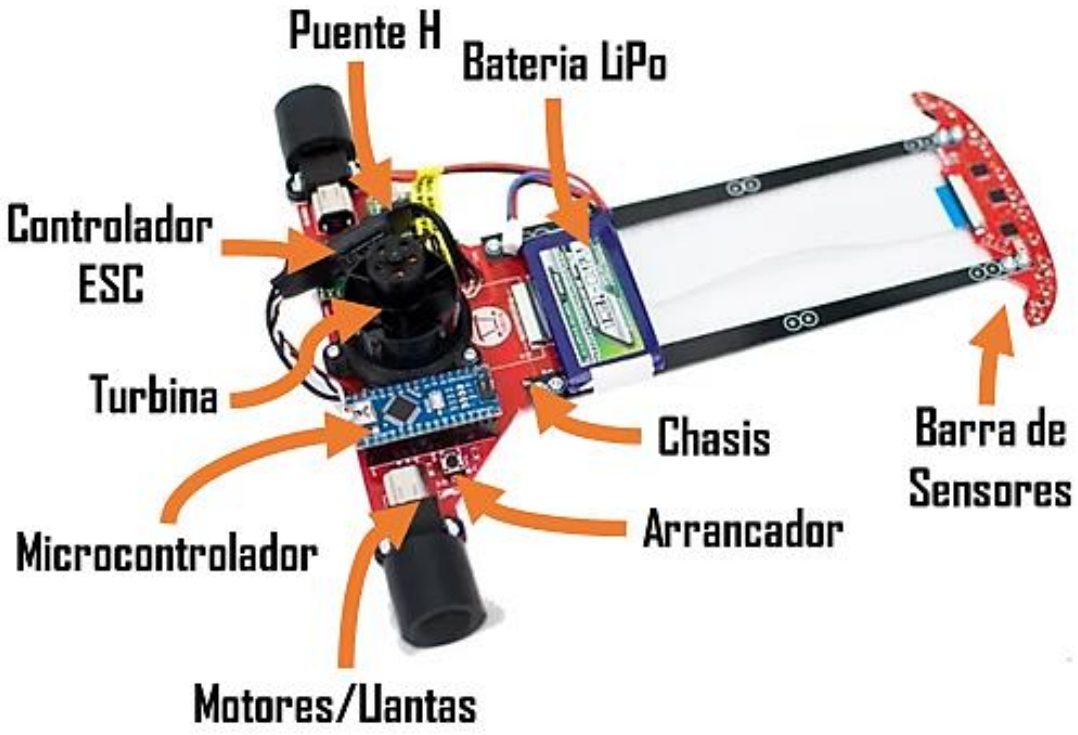

Figura 2.8. Estructura de un Robot Seguidor de Línea Velocista.

2.3 KiCad

KiCad es un paquete de software libre para la automatización del diseño electrónico (del inglés: Electronic Design Automation, EDA) (ver figura 2.9) [4]. Facilita el diseño de esquemáticos para circuitos electrónicos y su conversión a placa de circuito impreso (del inglés: Printed Circuit Board, PCB).

La primera fecha de lanzamiento fue en 1992 por su autor original, Jean-Pierre Charras, pero ahora está en desarrollo por el equipo de desarrolladores de KiCad. Cabe destacar que en 2013 la sección del CERN denominada BE-CO-HT (del inglés: Hardware and Timing), comenzó a aportar recursos a KiCad para ayudar a fomentar el desarrollo de hardware abierto al ayudar a mejorar KiCad a la par con las herramientas comerciales de EDA.

KiCad no presenta ninguna limitación en cuanto al tamaño de la placa de circuito y puede gestionar fácilmente hasta 32 capas de cobre, hasta 14 capas técnicas y hasta 4 capas auxiliares. KiCad puede crear todos los archivos necesarios para la construcción de placas de circuito impreso, archivos Gerber para foto-plotters, archivos para taladrado, archivos de ubicación de los componentes y mucho más. Al ser de código abierto (licencia GPL), KiCad representa la herramienta ideal para proyectos orientados a la creación de equipos electrónicos con estilo open-source [10].

Bajo su singular fachada, KiCad incorpora un elegante conjunto con las siguientes herramientas software:

- KiCad: Gestor del proyecto.
- Eeschema: Editor de esquemas y componentes.
- CvPcb: Selector de huellas (ejecutado siempre desde Eeschema).
- Pcbnew: Entorno de diseño de los circuitos impresos (PCB).
- GerbView: Visor de ficheros Gerber.

KiCad funciona en GNU/Linux, Apple OS X y Windows. Puede encontrarse la mayoría de instrucciones actualizadas y enlaces de descarga en:

- https://kicad-pcb.org/download/

Figura 2.9. Logo del Software de Diseño KiCad.

2.4 STM32CubeIDE

STM32CubeIDE es una herramienta de desarrollo de sistemas operativos múltiples todo en uno, que forma parte del ecosistema de software STM32Cube (ver figura 2.10).

STM32CubeIDE es una plataforma de desarrollo C / C ++ avanzada con configuración de periféricos, generación de código, compilación de código y funciones de depuración para microcontroladores y microprocesadores STM32. Se basa en el Eclipse ® / marco CDT y GCC cadena de herramientas para el desarrollo, y GDB para la depuración. Permite la integración de los cientos de complementos existentes que completan las características del Eclipse ® IDE [11].

STM32CubeIDE integra la configuración de STM32 y las funciones de creación de proyectos de STM32CubeMX para ofrecer una experiencia de herramienta todo en uno y ahorrar tiempo de instalación y desarrollo. Después de la selección de una MCU o MPU STM32 vacía, o un microcontrolador o microprocesador preconfigurado a partir de la selección de una placa o la selección de un ejemplo, se crea el proyecto y se genera el código de inicialización.

En cualquier momento durante el desarrollo, el usuario puede volver a la inicialización y configuración de los periféricos o middleware y regenerar el código de inicialización sin impacto en el código de usuario.

STM32CubeIDE incluye analizadores de compilación y pila que brindan al usuario información útil sobre el estado del proyecto y los requisitos de memoria.

STM32CubeIDE también incluye funciones de depuración estándar y avanzadas que incluyen vistas de registros del núcleo de la CPU, memorias y registros periféricos, así como vigilancia variable en vivo, interfaz de visor de cables en serie o analizador de fallas.

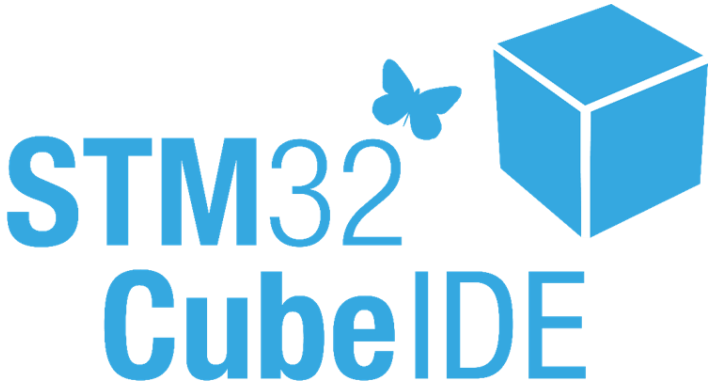

Figura 2.10. Logo de la Plataforma de Desarrollo STM32CubeIDE.

Capítulo 3: Metodología

En este capítulo se abordará el desarrollo a través del cual se diseñará, construirá y programará nuestro Robot Seguidor de Línea Autónomo. Para ello, este capítulo se encuentra dividido en tres secciones:

- Requerimientos Generales del Robot Seguidor de Línea (Hardware).
- Diseño de la PCB (Chasis) del Robot Seguidor de Línea en KiCad.
- Programación del Sistema Embebido del Robot Seguidor en STM32CubeIDE.

3.1 Requerimientos Generales del Robot Seguidor de Línea (Hardware)

A continuación, se describen parte por parte los componentes utilizados para la construcción del Autonomous Line Follower Robot, producto de la realización de este proyecto.

3.1.1 Chasis

Existe una gran variedad de materiales con los cuales se pueden diseñar los robots seguidores de línea como lo son el plástico, aluminio, aleaciones de latón e inclusive se puede llegar a utilizar madera. Para el diseño del robot de este proyecto se utilizó como material la misma placa impresa de circuitos PCB, la cual tiene 1mm de espesor (ver figura 3.1).

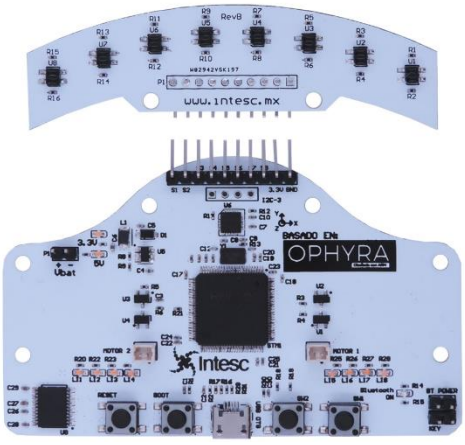

Figura 3.1. Ejemplo del Uso de una PCB como Chasis de un Robot Seguidor de Línea (Ophyra Racing, Intesc).

3.1.2 Sensores

Este tipo de robots utiliza sensores ópticos los cuales se encargan de detectar la cantidad de luz reflejada por una superficie. El sensor está conformado por un LED emisor infrarrojo, el cual genera luz en el espectro infrarrojo, este emisor hace rebotar la señal sobre la superficie que se encuentra recorriendo mientras que un optotransistor es el encargado de recibir la luz reflejada [9]. Estos sensores poseen un filtro de luz con la finalidad de evitar perturbaciones por luz ambiente.

Para la construcción del robot de este proyecto se utilizó una barra de 16 sensores optoreflectivos (QRE1113) multiplexados (74HC4067), diseñada especialmente para seguidores de línea de competencia (ver figura 3.2). Esto permite que se puedan leer los 16 sensores usando tan solo 5 pines del microcontrolador, generando así una mayor optimización en el procesamiento de datos [12].

Especificaciones técnicas:

- Voltaje de alimentación: 3.3V.
- Corriente de consumo: 120mA.
- Tamaño exterior: 140mm*30mm.
- Peso: 5g.

Figura 3.2. Barra de 16 Sensores Optoreflectivos (QRE1113) Multiplexados (74HC4067) (SR-Phoenix).

3.1.3 Microcontrolador

El dispositivo encargado de realizar todo el procesamiento dentro del sistema robótico de este proyecto es un microcontrolador STM32F407VGT6 LQFP100 (ver figura 3.3). Este microcontrolador integra un ARM Cortex-M4 de 32 bits, y cuenta con una Unidad de Punto Flotante (FPU) e integra un set de instrucciones orientados a procesamiento digital de señales (DSP, por sus siglas en inglés) [13].

Especificaciones técnicas:

- STM32F407 ARM Cortex-M4 de 32 bits.
- Voltaje de alimentación: 3.3V.
- Máxima frecuencia CPU: 168MHz.
- Convertidor analógico digital de 12 bits.
- 14 Timers de propósito General y Específico.
- Cristal de 8MHz.

Figura 3.3. Microcontrolador STM32F407VGT6 LQFP100.

3.1.4 Motores

Para este robot se utilizaron un par de motorreductores de la marca Pololu, con una relación de engranes de 10:1 HP (ver figura 3.4), voltaje nominal de 6V. Estos motorreductores alcanzan velocidades de hasta 3000 revoluciones por minuto (RPM). Tienen un consumo de 120mA de corriente sin carga y un torque de 0.3kg*cm [14].

Figura 3.4. Par de Motorreductores Tipo Pololu 10:1 HP 6V.

3.1.5 Controlador Puente H

Los controladores de puente H se encargan de recibir las señales de control provenientes del microcontrolador, entregando a su salida una señal equivalente, pero de mayor potencia capaz de controlar a los motores del robot [9]. Para el control de los

motores del robot seguidor de línea de este proyecto, se utilizaron un par de controladores puente H IFX9201SG de la marca infineon (ver figura 3.5). El IFX9201SG es un puente en H de 6 A de uso general, diseñado para el control de motores de DC u otras cargas inductivas. Las salidas pueden modularse por ancho de pulso a frecuencias de hasta 20 KHz [15].

Características técnicas:

- Voltaje de operación: 4.5V a 36V.
- Entradas lógicas: 3.3V y 5V TTL / 5V CMOS compatibles.
- Protección contra cortocircuitos y sobrecalentamiento.

Figura 3.5. Puente H IFX9201SG 6A.

3.1.6 Arrancador

Para el accionamiento del robot de este proyecto se utilizó un módulo KY-022 (ver figura 3.6). Este módulo es un sensor receptor infrarrojo IR, el cual está básicamente construido por un receptor infrarrojo 1838, un led indicador, una resistencia smd 102 y un header macho de ángulo de 3 pines. Esté sensor KY-022 infrarrojo receptor reacciona a la luz infrarroja de 38 KHz, transmitida por un módulo transmisor, como controles remotos IR, que se utilizan en muchos equipos domésticos [16].

Características técnicas:

- Voltaje de funcionamiento: 2.7V a 5.5V.
- Corriente de funcionamiento: 0.4mA a 1.5mA.
- Distancia de recepción: 18m.
- Ángulo de recepción: ± 45º.

- Frecuencia portadora: 38KHz.
- Voltaje de bajo nivel: 0.4V.
- Voltaje de alto nivel: 4.5V.
- Filtro de luz ambiente hasta: 500LUX.
- Dimensiones: 15 x 18.5 x 10 mm.

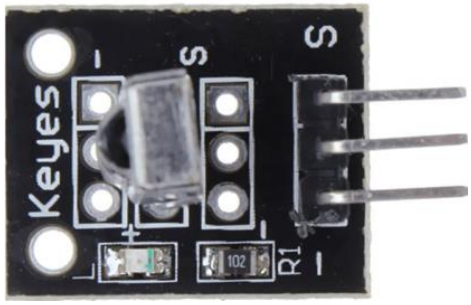

Figura 3.6. Módulo KY-022 Sensor Receptor Infrarrojo IR.

3.1.7 Regulador de Voltaje

Para poder alimentar el microcontrolador, así como la placa de sensores del robot seguidor de línea de este proyecto, se utilizó un regulador de voltaje LM1086CS-3.3V (ver figura 3.7) [17].

Características técnicas:

- Voltaje de entrada máximo hasta 29V.
- Voltaje de salida fijo de 3.3V.
- Corriente de salida 1.5A.
- Rango de temperatura: -40°C a 125°C.

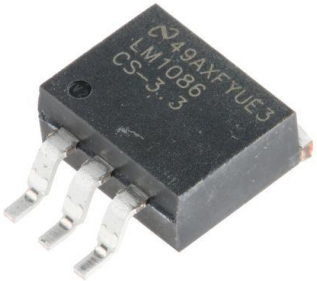

Figura 3.7. Regulador de Voltaje LM1086CS-3.3V.

3.1.8 Turbina

El diseño del robot de este proyecto incluye una turbina QX Motor QF1611 (ver figura 3.8). Esta turbina es utilizada como un soporte para mantener la estabilidad del robot a través de su recorrido en la pista de competencia [18].

Características técnicas:

- KV: 7000KV.
- Configuración: 9N6P.
- No. de celdas (LiPo): 2S.
- Corriente máxima de continuidad (A) / 60s: 12A.
- Potencia continua máxima (W) / 60s: 120W.

Figura 3.8. Turbina QX Motor QF1611.

3.1.9 Controlador ESC 12A

Para controlar el funcionamiento de la turbina utilizada en el robot de este proyecto se tuvo que utilizar un controlador de velocidad electrónico ESC 12A EMAX para motores Brushless (ver figura 3.9). Este controlador es quien define la velocidad de giro de la turbina mediante la generación de pulsos [19].

Especificaciones técnicas:

- Corriente continua: 12A.
- Corriente máxima: 15A.
- Salida Regulador BEC: 1A/5V.
- Tipo de batería: 2 - 4 Celdas.
- Programable: Sí.
- Dimensiones: 42 x 20 x 8 mm.

Figura 3.9. Controlador de Velocidad Electrónico ESC 12A EMAX.

3.1.10 Batería LiPo

Una de las partes más importantes a considerar en el diseño del robot seguidor de línea de este proyecto ha sido la alimentación tanto de la PCB, los motorreductores y la turbina. Ante ello, se ha utilizado una Batería LiPo de la marca Turnigy (ver figura 3.10). Estas baterías son conocidas en todo el mundo por su rendimiento, fiabilidad y precio. Además, estas baterías están equipadas con descarga de alta resistencia, lo que conduce a mantener altas cargas de corriente [20].

Especificaciones técnicas:

- Capacidad: 460mA.
- Voltaje: 3S1P / 3 Celdas / 11.1V.
- Descarga: 25C Constant / 40C Burst.
- Dimensiones: 67x30x15mm.
- Plug de carga: JST-XH.
- Plug de descarga: mini-JST.

Figura 3.10. Batería LiPo Turnigy 460mA 3S 25~40C.

A continuación, en la tabla 4 se puede observar la lista completa del hardware empleado y los costos por unidad de cada uno de los elementos utilizados para la construcción de un solo Robot Seguidor de Línea Autónomo.

Tabla 4. Lista de Materiales/Costo para la Construcción de un Robot Seguidor de Línea.

Materiales		Cantidad	Costo Individual (IVA incluido)	Subtotal (MXN)
Micro Motorreductor Tipo Pololu 3000rpm/6V Par		1	$852.60	$852.60
Driver IFX9201SG SMD		2	$139.20	$278.40
Diodo Zener 15V BZX84C15		1	$4.64	$4.64
Agarradera Motorreductor		2	$58.00	$116.00
Microcontrolador STM32F407VGT6 SMD		1	$266.80	$266.80
Regulador de Voltaje a 3.3V LM1086CS-3.3 SMD		1	$69.60	$69.60
Push Button SMD		3	$4.64	$13.92
Módulo KY-022 Sensor Receptor Infrarrojo		1	$17.40	$17.40
Tira de 40 Pines Headers Macho		1	$3.48	$3.48
Oscilador 8MHz SMD		1	$5.80	$5.80
Resistencia SMD 0603 10kΩ		2	$0.29	$0.58
Resistencia SMD 0603 100kΩ		3	$0.29	$0.87
Capacitor SMD 0603 100nF 50V		9	$0.58	$5.22
Capacitor SMD 0603 2.2uF 16V		2	$4.64	$9.28
Capacitor SMD 0603 20pF 50V		2	$0.58	$1.16
Capacitor SMD 1206 100uF 16V		1	$9.28	$9.28
Capacitor SMD 1206 10uF 16V		2	$5.80	$11.60
Conector Macho 2.54mm Pitch 4 Pin JST		1	$5.80	$5.80
Fabricación de PCB		1	$580.00	$580.00
Ensamble de Tarjetas PCB		1	$348.00	$348.00
Placa de 16 sensores con multiplexor		1	$556.80	$556.80
Turbina EDF 3S con Driver ESC 12a		1	$1,856.00	$1,856.00
Par de rines PLA para motorreductor		1	$92.80	$92.80
Par de llantas de caucho para motorreductor		1	$69.60	$69.60
Bateria Lipo 460mah 3s 11.1v Nano Tech		1	$555.00	$555.00
		Total		**$5,730.63**

3.2 Diseño de la PCB (Chasis) del Robot Seguidor de Línea en KiCad

A continuación, se describen parte por parte cada uno de los componentes usados en el diseño de la PCB del Robot Seguidor de Línea Autónomo, producto de la realización de este proyecto.

3.2.1 Botones de Propósito Específico y General

El diseño de este Robot incorpora tres botones tipo push de montaje superficial en la placa. Dos de estos botones son de propósito específico del microcontrolador: RESET Y BOOT0. El botón RESET es utilizado para reiniciar el microcontrolador, mientras que el botón BOOT0 es usado para forzar al microcontrolador a entrar en modo Bootloader; si se mantiene presionado durante un reinicio [21]. Por otro lado, el botón AM1 tiene la función de hacer un arranque manual del funcionamiento de nuestro robot, esto para que comience a correr.

La figura 3.11 muestra la conexión general de los botones con los que cuenta nuestra plataforma robótica en el software de diseño KiCad. Cabe resaltar que la configuración de conexión de cada uno de los pulsadores es normalmente abierta.

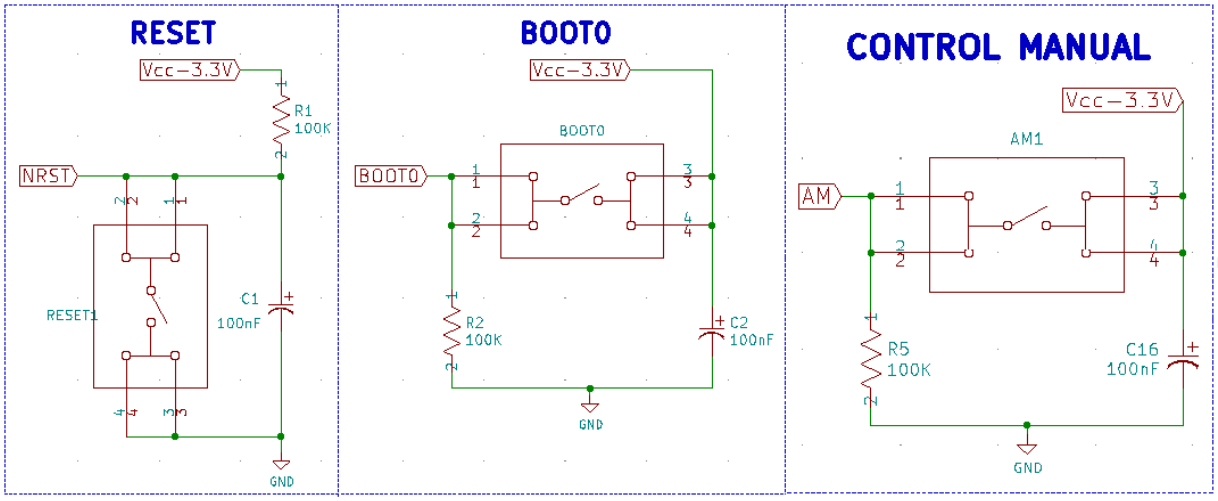

Figura 3.11. Diagrama de Conexión de los Botones de Propósito Específico y General en KiCad.

3.2.2 Regulador de Voltaje

Para poder alimentar todos los circuitos de la placa de nuestro Seguidor de Línea, el diseño de la PCB incorpora un regulador de voltaje LM1086CS-3.3V. Este regulador recibe a la entrada 7.4V de dos celdas de la batería y entrega un voltaje de 3.3V a la

salida. El circuito esquemático de las conexiones del regulador de voltaje, realizado en KiCAD, se puede apreciar en la figura 3.12.

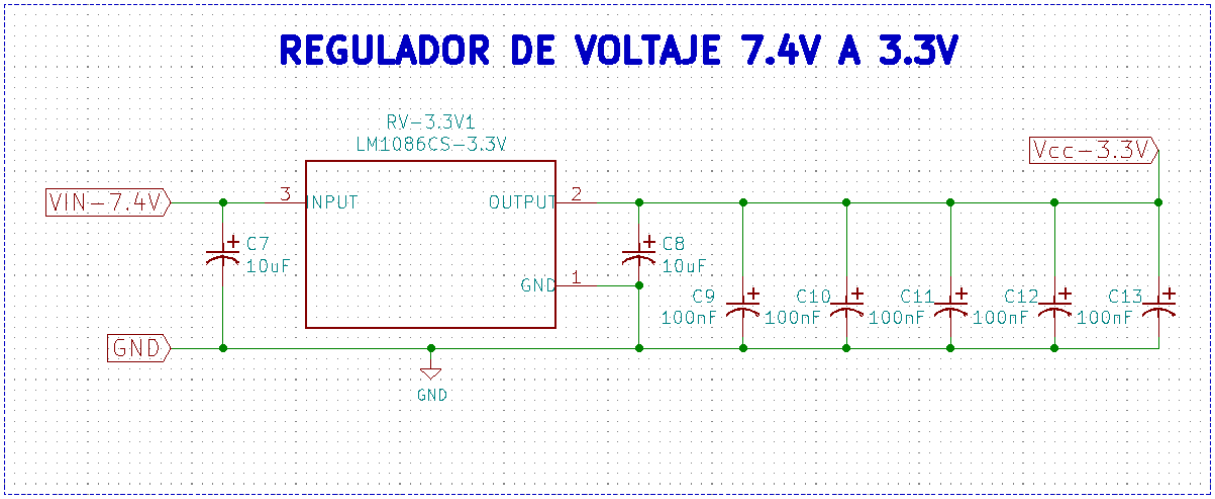

Figura 3.12. Diagrama de Conexiones del Regulador de Voltaje en KiCad.

3.2.3 Conexiones Básicas para el Funcionamiento del Microcontrolador

A continuación, se muestran las conexiones básicas que requiere el microcontrolador STM32F407VGT6 para que pueda funcionar correctamente (ver figura 3.13). Dentro de los requerimientos principales se encuentran las conexiones del oscilador de 8MHz y las conexiones de los pines del microcontrolador que necesitan ser alimentados.

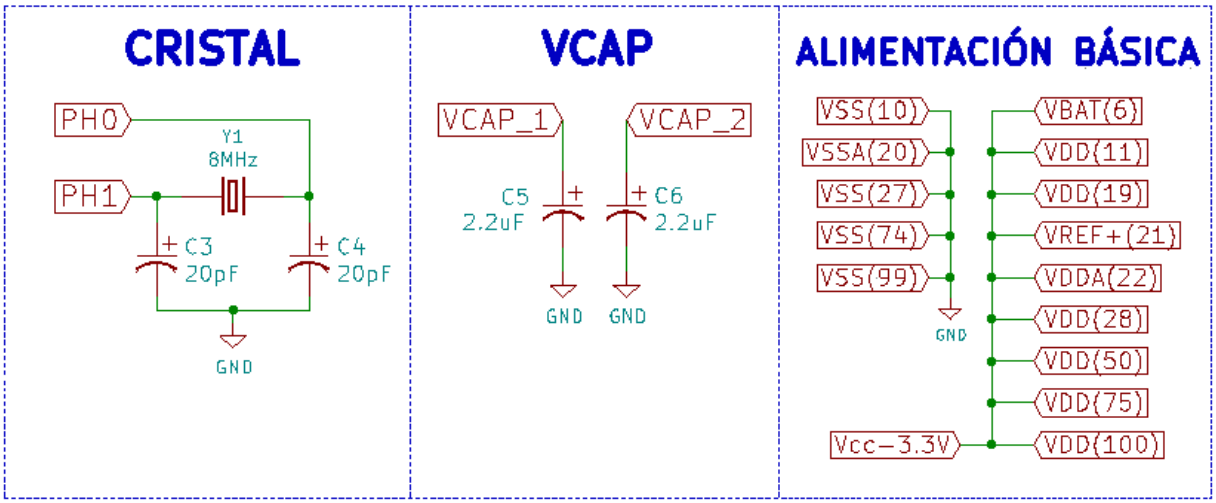

Figura 3.13. Diagrama de Conexiones Básicas que Requiere el Microcontrolador STM32F407VGT6 en KiCad.

3.2.4 Puerto para Depuración (Debug Port SWD)

Este puerto permite conectar nuestra plataforma robótica con la computadora, a través del depurador ST Link de la empresa STMicroelectronics. Al utilizar este puerto, se puede ejecutar un programa y observar el comportamiento de los registros internos del microcontrolador en tiempo real [22]. El circuito esquemático de las conexiones para el puerto de depuración SWD, realizado en KiCAD, se puede apreciar en la figura 3.14.

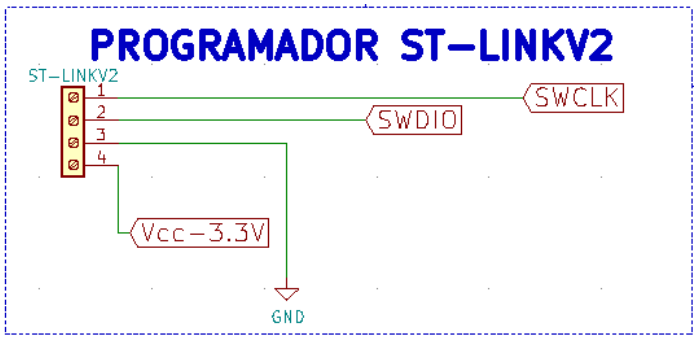

Figura 3.14. Diagrama de Conexiones del Puerto para Depuración (Debug Port SWD) en KiCad.

3.2.5 Controlador Puente H

Para poder llevar a cabo la etapa de control y sentido de giro de los motores, es necesario incorporar una etapa de potencia para así suministrar la corriente necesaria que necesitan los motores. En este caso, se ha utilizado un par de controladores puente H IFX9201SG de 6A. El circuito esquemático completo de las conexiones del par de controladores puente H, realizado en KiCad, se muestra en la figura 3.15.

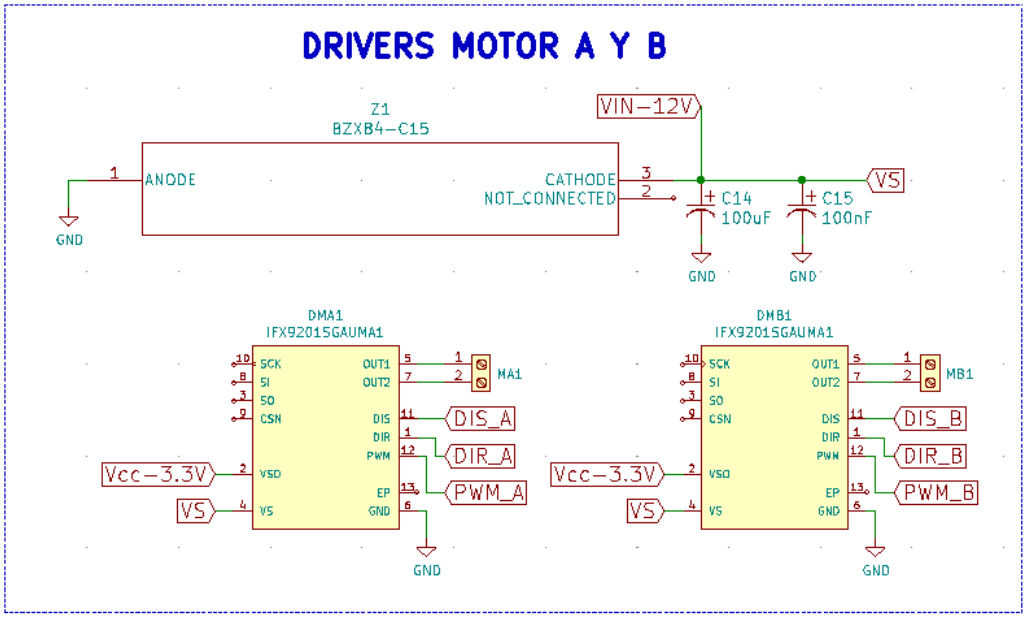

Figura 3.15. Diagrama de Conexiones de los Controladores Puente H en KiCad.

3.2.6 Turbina y Controlador ESC 12A

Para poder mantener la estabilidad del Robot Seguidor de Línea al ir a altas velocidades en la pista de competencia, es necesario el uso de una turbina junto con su controlador ESC. En este caso, se ha utilizado una turbina QX Motor QF1611 y un driver ESC 12A. El circuito esquemático completo de las conexiones de la turbina junto con su controlador, realizado en KiCad, se muestra en la figura 3.16.

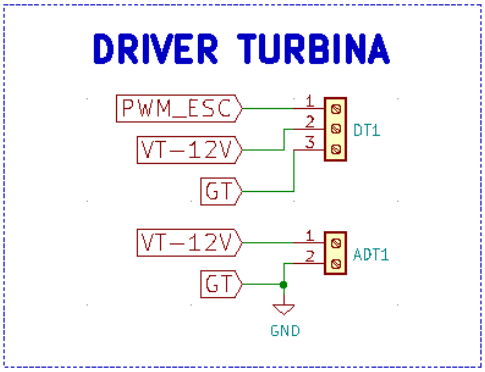

Figura 3.16. Diagrama de Conexiones de la Turbina Junto con su Controlador ESC 12A en KiCad.

3.2.7 Módulo Infrarrojo y Módulo Arrancador

Para poder accionar nuestro Robot Seguidor de Línea se incorpora en la PCB un módulo de control infrarrojo KY-022, así como también se dejan los pines disponibles para poder utilizar un módulo de arranque remoto de la marca Ingeniero Maker. El circuito esquemático completo de las conexiones tanto del módulo infrarrojo, así como del módulo de arranque, realizados en KiCad, se muestra en la figura 3.17.

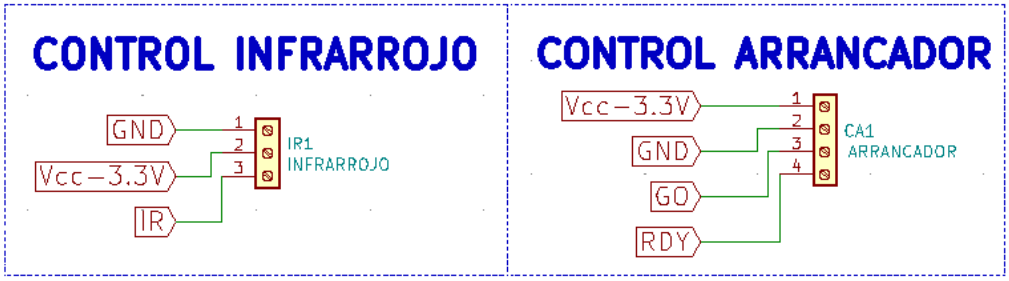

Figura 3.17. Diagrama de Conexiones tanto del Módulo Infrarrojo, así como del Módulo de Arranque en KiCad.

3.2.8 Barra de Sensores Optoreflectivos

El diseño de este Robot incorpora una barra de 16 sensores optoreflectivos (QRE1113) multiplexados con un multiplexor 74HC4067. Estos sensores ayudan a que el robot pueda seguir perfectamente la línea que marca la trayectoria a seguir en la pista. La

figura 3.18 muestra las conexiones de la placa de sensores hacia la plataforma robótica donde se encuentra el microcontrolador.

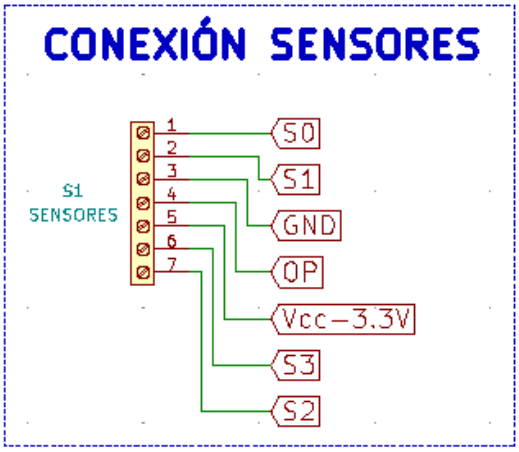

Figura 3.18. Diagrama de Conexiones de la Placa de Sensores Optoreflectivos en KiCad.

3.2.9 Batería LiPo 3s

Para poder alimentar nuestra plataforma robótica es necesario el uso de una batería LiPo de 3s (11.1V). En la figura 3.19, se pueden observar las conexiones necesarias para que se conecte la batería a nuestra PCB del Robot Seguidor de Línea.

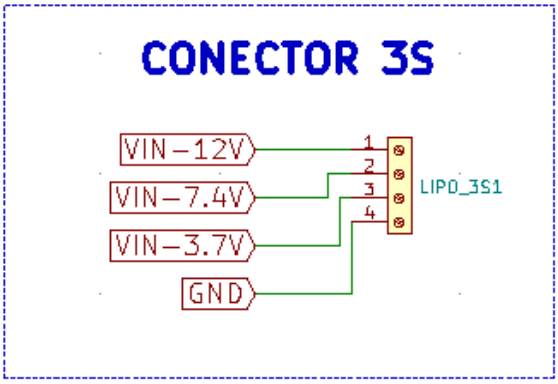

Figura 3.19. Diagrama de Conexiones de la Bateria Lipo 3s en KiCad.

3.2.10 Microcontrolador STM32F407VGT6

Para poder llevar a cabo todo el procesamiento dentro del sistema robótico se ha incorporado un microcontrolador STM32F407VGT6. La figura 3.20 muestra la conexión general hacia el microcontrolador de todos los componentes necesarios para el funcionamiento óptimo de nuestro Robot Seguidor de Línea Autónomo.

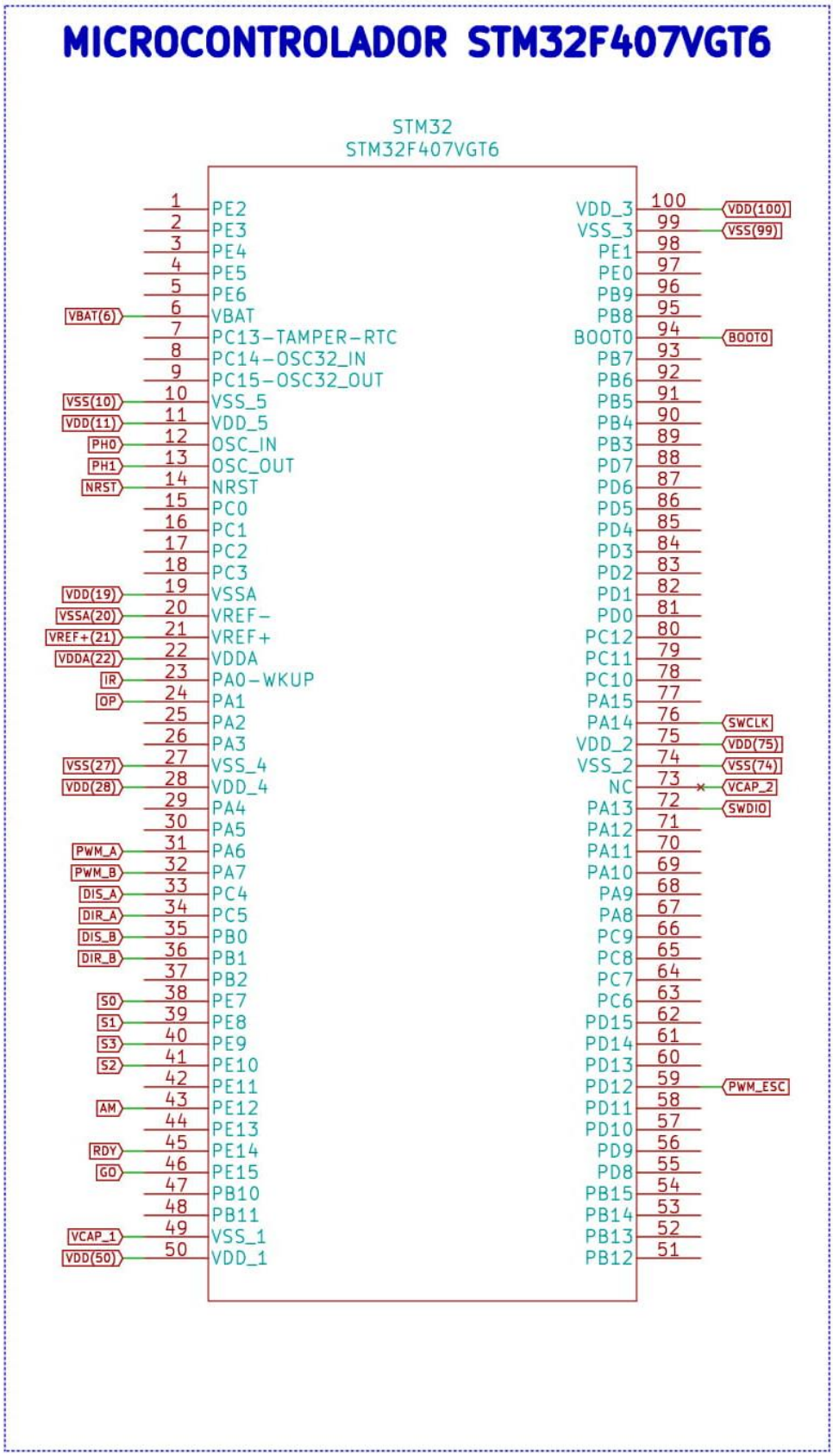

Figura 3.20. Diagrama de Conexiones de los Componentes Electrónicos Hacia el Microcontrolador en KiCad.

3.2.11 Diseño del Módulo Complementario para el Depurador y Programador ST-LINK V2

El ST-Link v2 (ver figura 3.21), es un programador/debugger construido por ST Microelectronics que permite programar y debuggear los procesadores STM8 y STM32. El ST-Link v2 implementa SWIM (Single Wire Interface Module) y JTAG/SWD (Serial Wire Debugging) para comunicarse con los procesadores de la placa de desarrollo [23].

Las características que posee el programador/debugger ST-Link v2 son las siguientes:

- Alimentación de 3,3 V o 5 V suministrada por un conector USB.
- Interfaz compatible con USB 2.0 de velocidad completa.
- Función de actualización directa de firmware compatible (DFU).
- LED de estado que parpadea durante la comunicación con la PC.
- Temperatura de funcionamiento 0 a 50 ° C.

Figura 3.21. Programador/Debugger ST-Link V2.

Sin embargo, para poder utilizar el ST-Link V2 como medio de programación del microcontrolador implementado en nuestro robot seguidor de línea, es indispensable hacer un arreglo de resistencias, tal como se muestra a continuación en la figura 3.22.

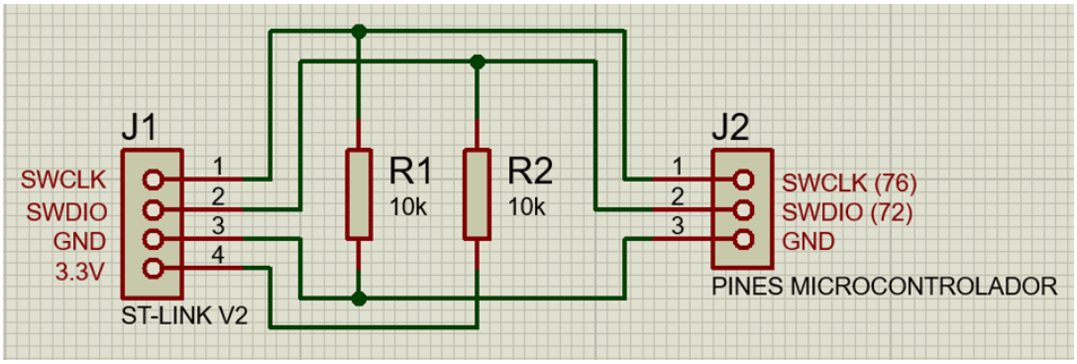

Figura 3.22. Circuito Eléctrico para el Desarrollo del Módulo Complementario del Programador/Debugger ST-Link V2.

3.3 Programación del Sistema Embebido del Robot Seguidor en STM32CubeIDE

A continuación, se describen las configuraciones que se deben realizar dentro del IDE de STM32CubeIDE para que pueda funcionar nuestro microcontrolador correctamente. Así mismo, se muestran los diferentes algoritmos empleados para el funcionamiento óptimo de nuestro Robot Seguidor de Línea Autónomo.

3.3.1 Configuraciones a realizar dentro del IDE de STM32CubeIDE

Selección de los pines a utilizar del microcontrolador STM32F407VG

En la tabla 5 se pueden observar los pines que se usaron del STM32F407VG LQFP100 y a que elementos del sistema corresponden.

Tabla 5. Pines Utilizados del Microcontrolador STM32F407VG LQFP100 para el Funcionamiento Óptimo del Robot Seguidor de Línea.

Pines del Microcontrolador	Elementos del Sistema Robótico
AM (PE12)	Control de arranque manual por botón
ADC1_IN1 (PA1)	Lectura del ADC arrojada por el multiplexor de la barra de sensores
S0 (PE7), S1 (PE8), S2 (PE10), S3 (PE9)	Pines de combinación para la selección del sensor a leer por el multiplexor de la barra de sensores
PWM_A (PA6)	PWM para controlar la velocidad del motor izquierdo
DIS_A (PC4), DIR_A (PC5)	Pines de control para el funcionamiento del driver del motor izquierdo
PWM_B (PA7)	PWM para controlar la velocidad del motor derecho
DIS_B (PB0), DIR_B (PB1)	Pines de control para el funcionamiento del driver del motor derecho
PWM_ESC (PD12)	PWM para controlar la velocidad de giro de la turbina
SYS_JTMS-SWDIO (PA13), SYS_JTCK-SWCLK (PA14)	Pines para el uso del Programador/Debugger ST-Link V2

Configuración del ADC1 IN1

Para poder muestrear correctamente la lectura de nuestra barra de sensores, es de suma importancia configurar adecuadamente el ADC que se utilizará por parte del microcontrolador (ver figura 3.23). Para ello, es indispensable tener en cuenta los siguientes requerimientos:

- El modo de configuración que se utilizará es el: Independent mode; puesto que, nos permitirá configurar el ADC según sean nuestras necesidades.
- La resolución del ADC que se manejará es de 12 bits.
- La frecuencia con la cual se trabajará es de 8MHz.

Para ver la frecuencia correspondiente a PCLK2 (16MHz), ver la imagen 3.24.

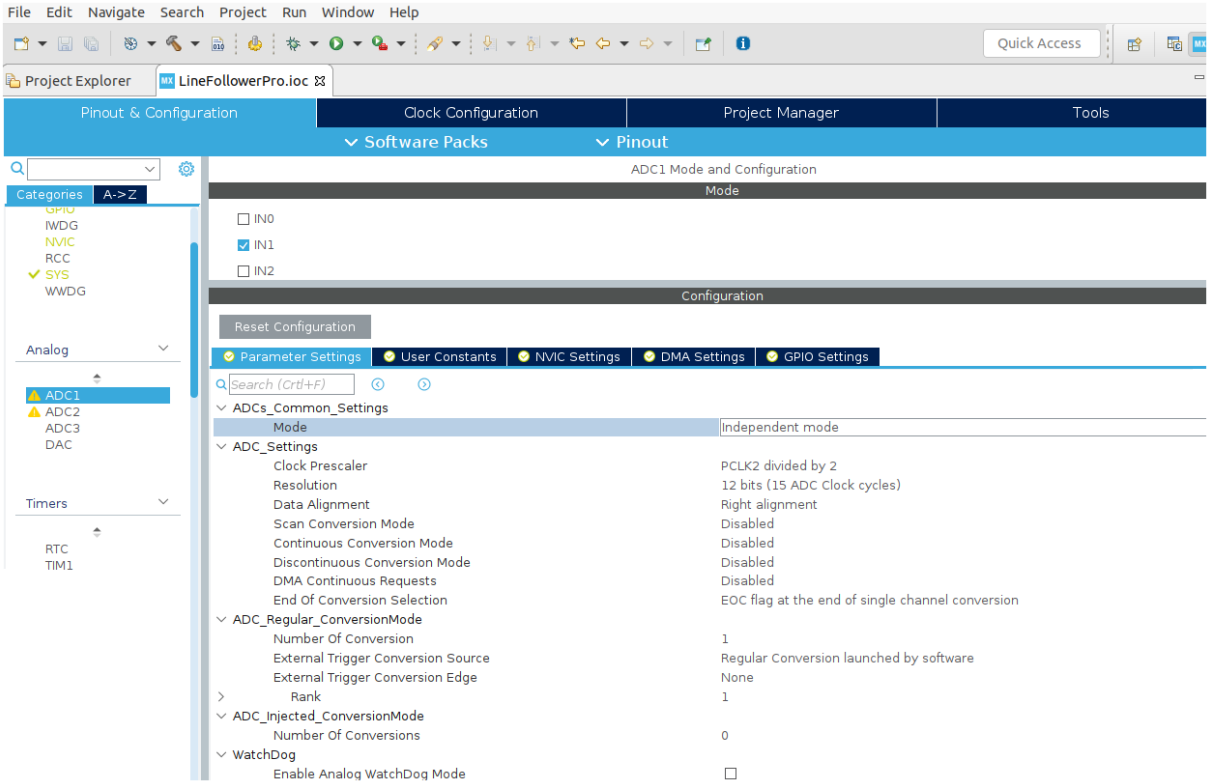

Figura 3.23. Configuración del ADC1 IN1 en el IDE de STM32CubeIDE.

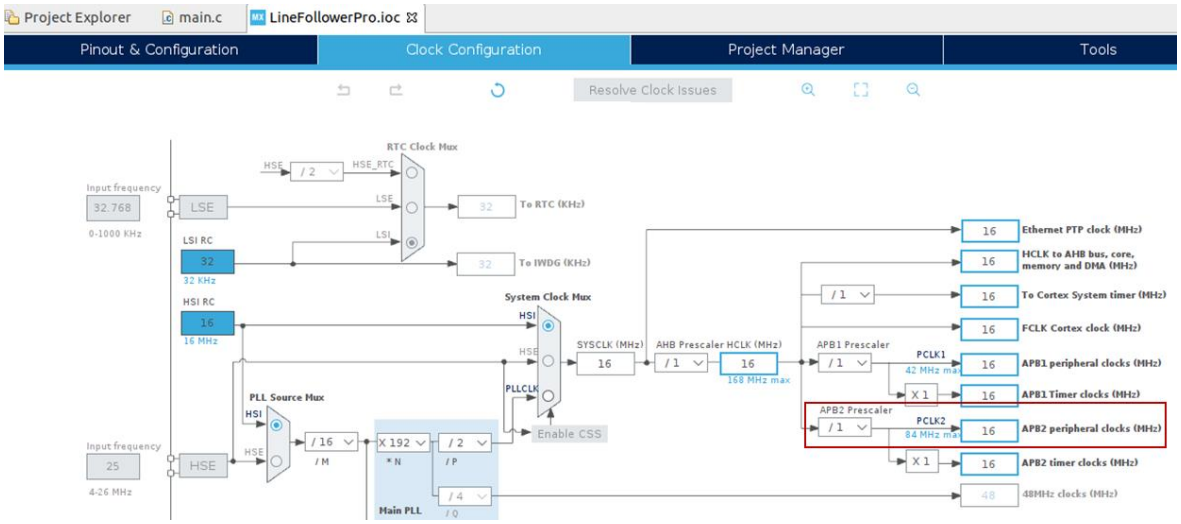

Figura 3.24. Frecuencia Correspondiente a PCLK2, la Cual es Equivalente a 16MHz.

Configuración de los pines a utilizar del microcontrolador STM32F407VG

A continuación, se muestran las configuraciones aplicadas a los pines que se usarán tanto para la selección del sensor a leer, como para el botón de arranque manual, en el IDE de STM32F407VG. Así mismo, se presentan las configuraciones aplicadas a los pines que controlarán a los drivers tanto del motor derecho como del motor izquierdo (ver figura 3.25).

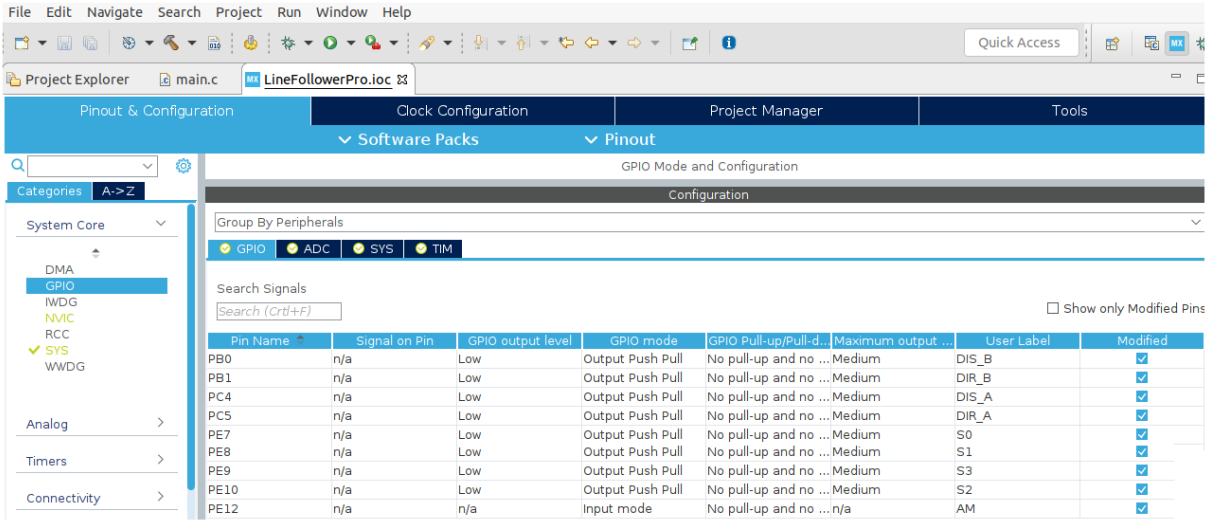

Figura 3.25. Configuración de los Pines a Utilizar del Microcontrolador STM32F407VG en el IDE de STM32CubeIDE.

Configuración de los PWM utilizados para el control de velocidad de los motores

Para controlar la velocidad de los motores se decidió utilizar los PWM correspondientes al Timer 3 (TIM3) del microcontrolador. Los Timers que posee el STM32F407VG tienen una frecuencia por default a 16MHz (ver figura 3.27). Sin embargo, para poder generar nuestros PWM se optó por emplear una frecuencia a 4kHz para que los motores respondieran correctamente.

Para obtener los 4kHz se tuvo que utilizar un Prescaler (divisor de frecuencia programable) de valor 8. Sin embargo, el programa considera por defecto un valor 1 para el Prescaler, por lo cual, solo se tiene que ingresar un valor de 7. Teniendo como resultado la siguiente operación:

$$\frac{Frecuencia\ Timer}{1 + Prescaler} = \frac{16MHz}{1 + 7} = 2MHz \quad (5)$$

De acuerdo con el resultado de la ecuación 5, aún nos falta dividir la frecuencia preescalada entre otro valor para obtener finalmente nuestra frecuencia a 4kHz. El valor que necesitamos para dividir la frecuencia preescalada es el del Counter Period. Para ello, hacemos la siguiente operación:

$$4kHz = \frac{Frecuencia\ Preescalada}{Counter\ Period} \Rightarrow Counter\ Period = \frac{Frecuencia\ Preescalada}{4kHz} \quad (6)$$

$$Counter\ Period = \frac{2MHz}{4kHz} = 500 \quad (7)$$

Por lo cual, necesitamos un Counter Period de valor 500. Sin embargo, el programa considera por defecto un valor 1 para el Counter Period, por lo cual, solo se tiene que ingresar un valor de 499. De tal forma que, la configuración final del TIM3 para los PWM_A y PWM_B en el IDE de STM32CubeIDE se puede apreciar en la figura 3.26.

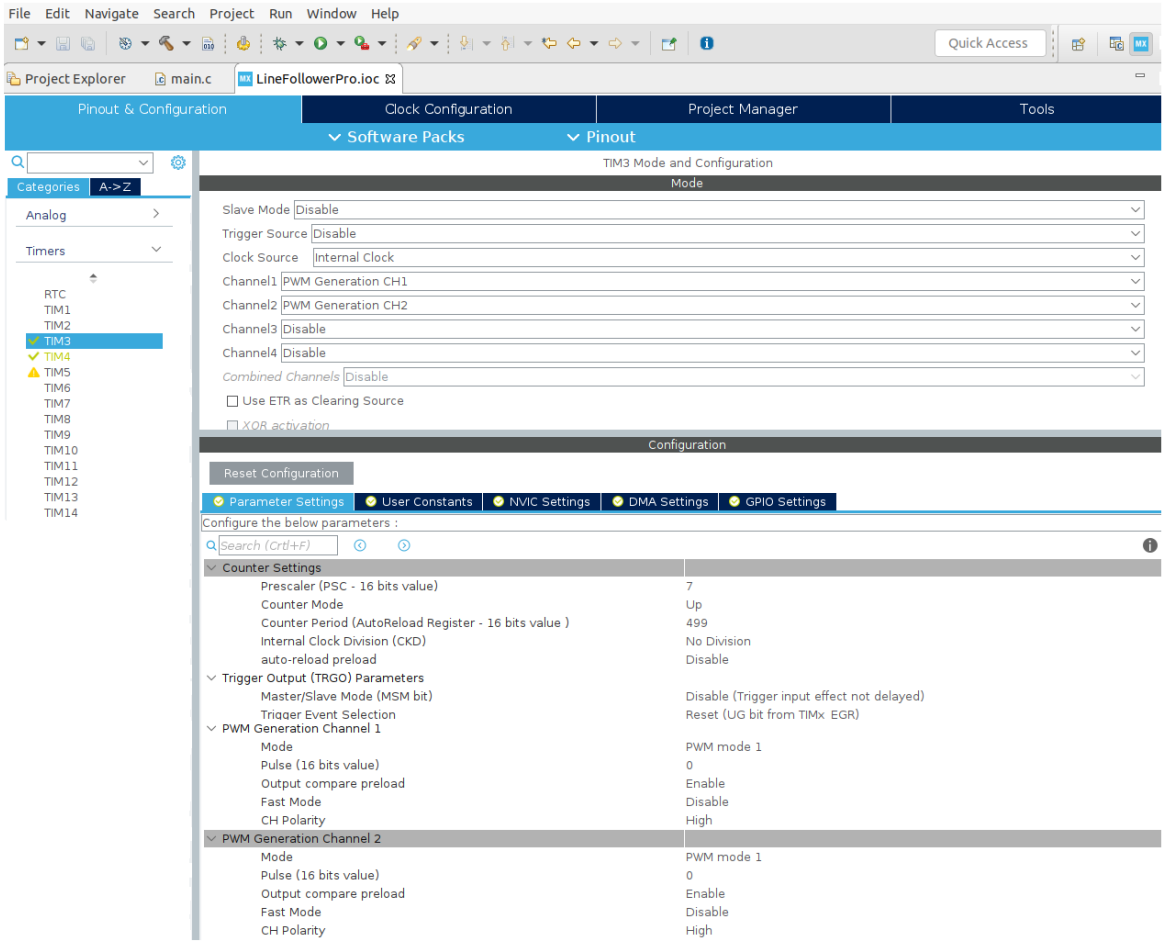

Figura 3.26. Configuración Final del TIM3 para los PWM_A y PWM_B en el IDE de STM32CubeIDE.

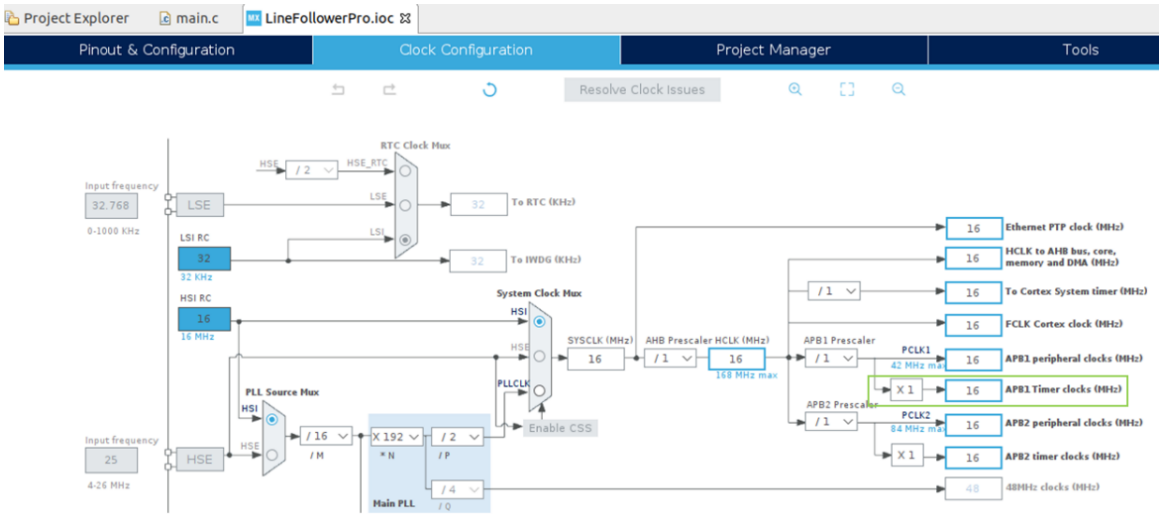

Figura 3.27. Frecuencia Correspondiente a los Timers, la Cual es Equivalente a 16MHz.

Configuración del PWM utilizado para controlar la velocidad de giro de la turbina

Para controlar la velocidad de giro de la turbina se debe utilizar un controlador conocido comúnmente como ESC. Los ESC se controlan mediante pulsos de entre 1 y 2 milisegundos y a una frecuencia de 50Hz (ver figura 3.28) [24].

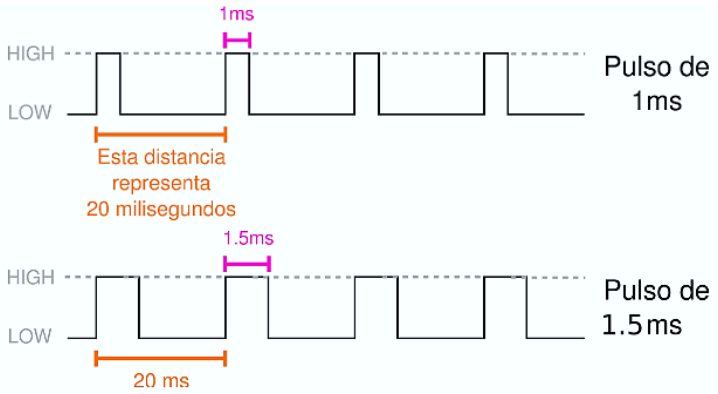

Figura 3.28. Diagrama de Pulsos para Controlar el Driver ESC 12A.

Para controlar la velocidad de giro de la turbina, se decidió utilizar uno de los PWM correspondientes al Timer 4 (TIM4) del microcontrolador. Los Timers que posee el STM32F407VG tienen una frecuencia por default a 16MHz. Sin embargo, para poder controlar correctamente nuestro driver ESC 12A, es importante generar un PWM a una frecuencia de 50Hz.

Para obtener los 50Hz se tuvo que utilizar un Prescaler (divisor de frecuencia programable) de valor 160. Sin embargo, el programa considera por defecto un valor 1 para el Prescaler, por lo cual, solo se tiene que ingresar un valor de 159. Teniendo como resultado la siguiente operación:

$$\frac{Frecuencia\ Timer}{1 + Prescaler} = \frac{16MHz}{1 + 159} = 100kHz \quad (8)$$

De acuerdo con el resultado de la ecuación 8, aún nos falta dividir la frecuencia preescalada entre otro valor para obtener finalmente nuestra frecuencia a 50Hz. El valor que necesitamos para dividir la frecuencia preescalada es el del Counter Period. Para ello, hacemos la siguiente operación:

$$50Hz = \frac{Frecuencia\ Preescalada}{Counter\ Period} \Rightarrow Counter\ Period = \frac{Frecuencia\ Preescalada}{50Hz} \quad (9)$$

$$Counter\ Period = \frac{100kHz}{50Hz} = 2000 \quad (10)$$

Por lo cual, necesitamos un Counter Period de valor 2000. Sin embargo, el programa considera por defecto un valor 1 para el Counter Period, por lo cual, solo se tiene que ingresar un valor de 1999. De tal forma que, la configuración final del TIM4 para el PWM_ESC en el IDE de STM32CubeIDE se puede apreciar en la figura 3.29.

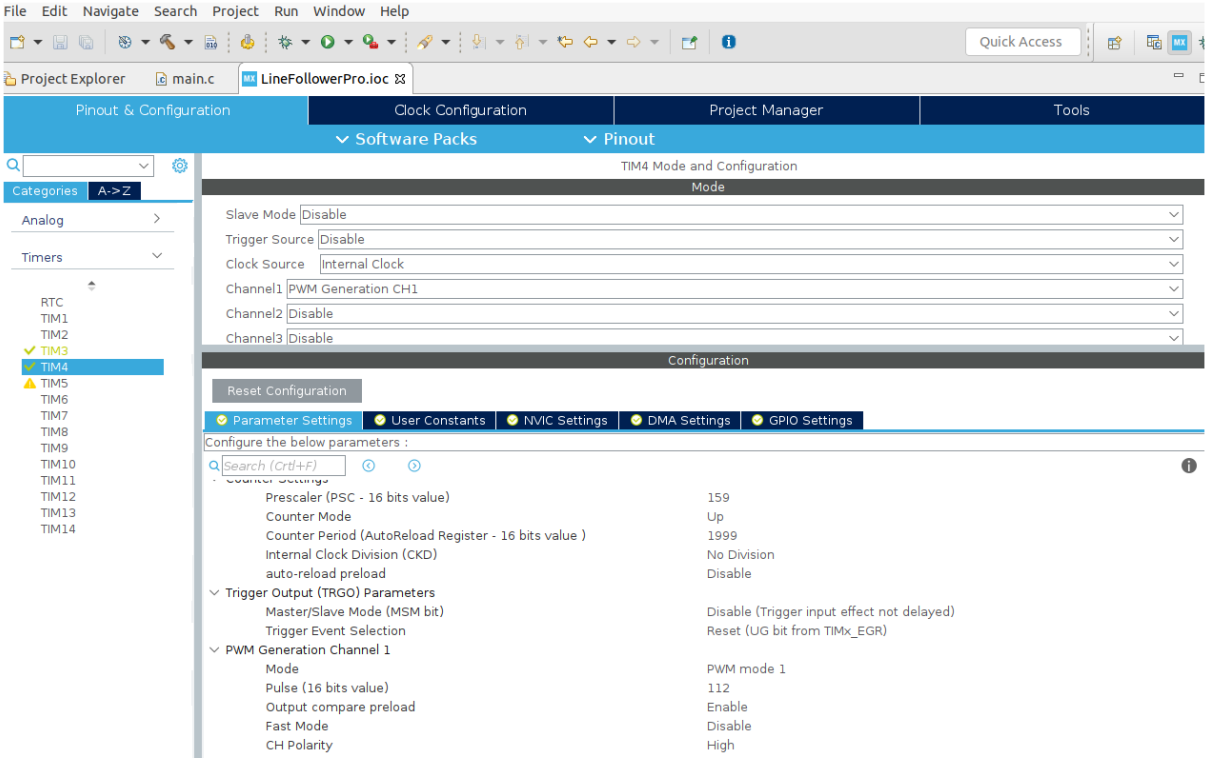

Figura 3.29. Configuración Final del TIM4 para el PWM_ESC en el IDE de STM32CubeIDE.

3.3.2 Diseño de la Arquitectura del Robot Seguidor de Línea

El paradigma robótico seleccionado para nuestro Robot Seguidor de Línea Velocista es el paradigma jerárquico. Este paradigma tiene una relación jerárquica entre las primitivas del robot. Primero la percepción del entorno (pista de competencia), después la planificación de acciones a realizar y por último la ejecución de estas acciones (ver figura 3.30).

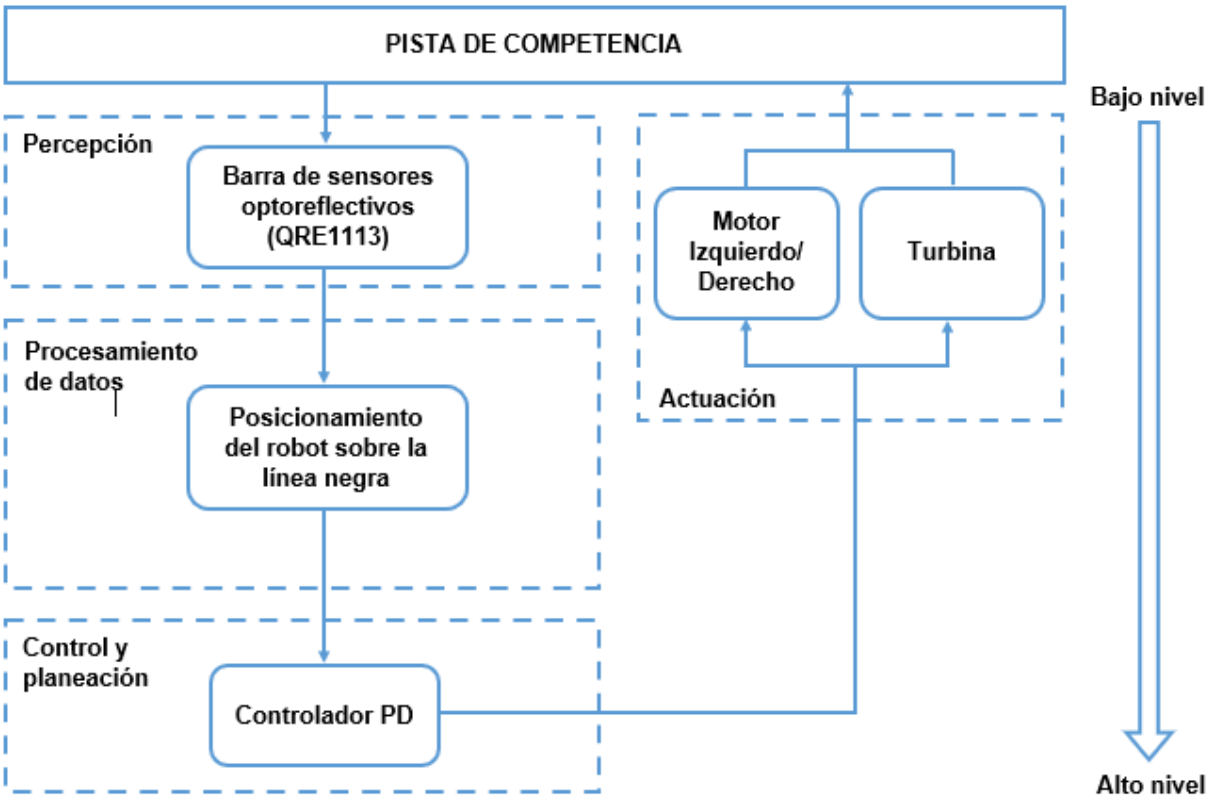

Figura 3.30. Arquitectura del Robot Seguidor de Línea Velocista.

Esta arquitectura cuenta con las tres primitivas del paradigma jerárquico, más una capa intermedia entre las etapas de percepción y de planeación. La capa intermedia es la del procesamiento de datos, la cual toma los datos crudos de la etapa de percepción y los transforma en información valiosa para la etapa de planificación. Las primitivas de más baja jerarquía son la percepción y la actuación, siendo estas las etapas que tienen una interacción directa con el entorno de la pista de competencia. La etapa de planificación es la de mayor jerarquía, siendo también la etapa con un mayor nivel de abstracción del entorno.

3.3.3 Algoritmos del Funcionamiento del Robot Seguidor de Línea

A continuación, se presentan los algoritmos empleados para que el Robot Seguidor de Línea pueda realizar correctamente sus funciones y así tener un desempeño optimo sobre la pista de competencias. De forma general, se ha decidido dividir en tres secciones el funcionamiento del robot. La primera sección consiste en la lectura de la barra de sensores optoreflectivos. La segunda sección corresponde a la obtención del error de la posición del seguidor de línea con respecto a la línea negra de la pista de competencias. Y finalmente, la tercera sección abarca la parte del controlador PD para la corrección del error mediante la actuación de los motores y la activación de la turbina.

1. Algoritmo para la lectura de la barra de sensores

read_sensors_qtr (sensors):

Input:
sensors is an array of integers
Output:
the array of sensors with digital values from 0 to 4095

```
1: for i in 0 to 15 do
2:     pin_S0 ← i & 0x01
3:     pin_S1 ← i & 0x02
4:     pin_S3 ← i & 0x04
5:     pin_S4 ← i & 0x08
6:     sensors[i] ← read_value_adc
7: end for
```

2. Algoritmo para la obtención del error de la posición del seguidor de línea

get_error (sensors, background):

Input:
sensors is an array of integers, background is the type of track color
Output:
error is the error of the position of the line follower with respect to the track and out_state is the last line follower position before exiting the track

```
1:  error ← 0.0
2:  weigth [8] ← {8,7,6,5,4,3,2,1}
3:  errorLeft ← 0
4:  errorRight ← 0
5:  read_sensors_qtr(sensors)
6:  max ← sensors [0]
7:  min ← sensors [0]
8:  for i in 1 to 15 do
```

```
9:      if sensors[i] > max then
10:       max ← sensors[i]
11:     end if
12:     if sensors[i] < min then
13:       min ← sensors[i]
14:     end if
15:   end for
16:   range ← max - min
17:   if range > 400 then
18:     threshold ←(range/2)+min
19:     for i in 0 to 15 do
20:       if background then
21:         if sensors[i] < threshold then
22:           bit_sensor[i] ← 1
23:         else then
24:           bit_sensor[i] ← 0
25:         end if
26:       else then
27:         if sensors[i] > threshold then
28:           bit_sensor[i] ← 1
29:         else then
30:           bit_sensor[i] ← 0
31:         end if
32:       end if
33:     end for
34:     for i in 0 to 7 do
35:       errorLeft ← errorLeft + (bit_sensor[i]*weigth[i])
36:       errorRight ← errorRight + (bit_sensor[15-i]*weigth[i])
37:     end for
38:     sum ← 0
39:     for i in 0 to 15 do
40:       sum ← sum + bit_sensor[i]
41:     end for
42:     error ← (errorRight - errorLeft) / sum
43:     if error <= 1.0 and error >= -1.0 then
44:       out_state ← CENTER
45:     else then
46:       out_state ← out_state
47:     end if
48:     if error > 1.0 and error <= 8.0 then
49:       out_state ← RIGHT
50:     else then
51:       out_state ← out_state
52:     end if
53:     if error < -1.0 and error >= -8.0 then
54:       out_state ← LEFT
55:     else then
56:       out_state ← out_state
57:     end if
58: else then
59:     error ← OUT_LINE
60: end if
```

3. Algoritmo para la implementación del controlador PD (actuación de los motores y activación de la turbina)

Control (error, out_state**):**

Input:
error is the error of the position of the line follower with respect to the track and out_state is the last line follower position before exiting the track
Output:
speed_1 is the speed of the left engine and speed_2 is the speed of the right motor

```
1:  state ← HOME
2:  error_now ← 0.0
3:  error_last ← 0.0
4:  speed_1 ← 0
5:  speed_2 ← 0
6:  U ← 0
7:  KP ← 78.778
8:  KD ← 216.6371
9:  MAX_SPEED ←225
10: MED_SPEED ←175
11: turbine ← 12
12: delay_seconds ← 8.0
13: while True do
14:       if AM_Pin == 1 then
15:          state ←STARTING
16:          delay_seconds ← 0.25
17:       end if
18:       if state == HOME then
19:          turbine ← 12
20:          speed_1 ← 0
21:          speed_2 ← 0
22:       else if state == STARTING then
23:          turbine ← 42
24:          state ← RUN
25:          delay_seconds ← 0.65
26:       else if state == RUN then
27:          error_now ← get_error (sensors, white)
28:          if error_now == OUT_LINE then
29:             if out_state == CENTER then
30:                speed_1 ← MED_SPEED
31:                speed_2 ← MED_SPEED
32:             else if out_state == RIGHT then
33:                speed_1 ← MAX_SPEED
34:                speed_2 ← 0 - MAX_SPEED
35:             else if out_state == LEFT then
36:                speed_1 ← 0 - MAX_SPEED
37:                speed_2 ← MAX_SPEED
38:             end if
39:          else then
40:             U ← KP*error_now + KD*(error_now-error_last)
41:             speed_1 ← MED_SPEED + U
42:             speed_2 ← MED_SPEED – U
43:             error_last ← error_now
```

```
44:     end if
45:     Right_Motor ← speed_2
46:     Left_Motor ← speed_1
47: end while
```

Para poder visualizar el Código implementado del funcionamiento del Robot Seguidor de Línea en el IDE de STM32CubeIDE, correspondiente al apartado de Main, ver el Anexo 1.

Capítulo 4: Resultados

En este capítulo se abordarán los resultados obtenidos del diseño, construcción y programación nuestro Robot Seguidor de Línea Autónomo. Para ello, este capítulo se encuentra dividido en tres secciones:

- Diseño de la PCB y Ensamble del Robot Seguidor de Línea.
- Diseño/Construcción de la Pista de Competencias y Pruebas de Validación del Funcionamiento del Robot.
- Participación en el ROBOTIC PEOPLE FEST Feria Universitaria y en el Intelibots Tournament Virtual 2020.

4.1 Diseño de la PCB y Ensamble del Robot Seguidor de Línea

Una vez teniendo los esquemáticos de cada uno de los módulos que conforman nuestra plataforma robótica, se procedió a importar todos los componentes al editor de PCB en el software KiCad. La figura 4.1 muestra la distribución que tienen los componentes en el editor de PCB en un área de aproximadamente 140x100mm.

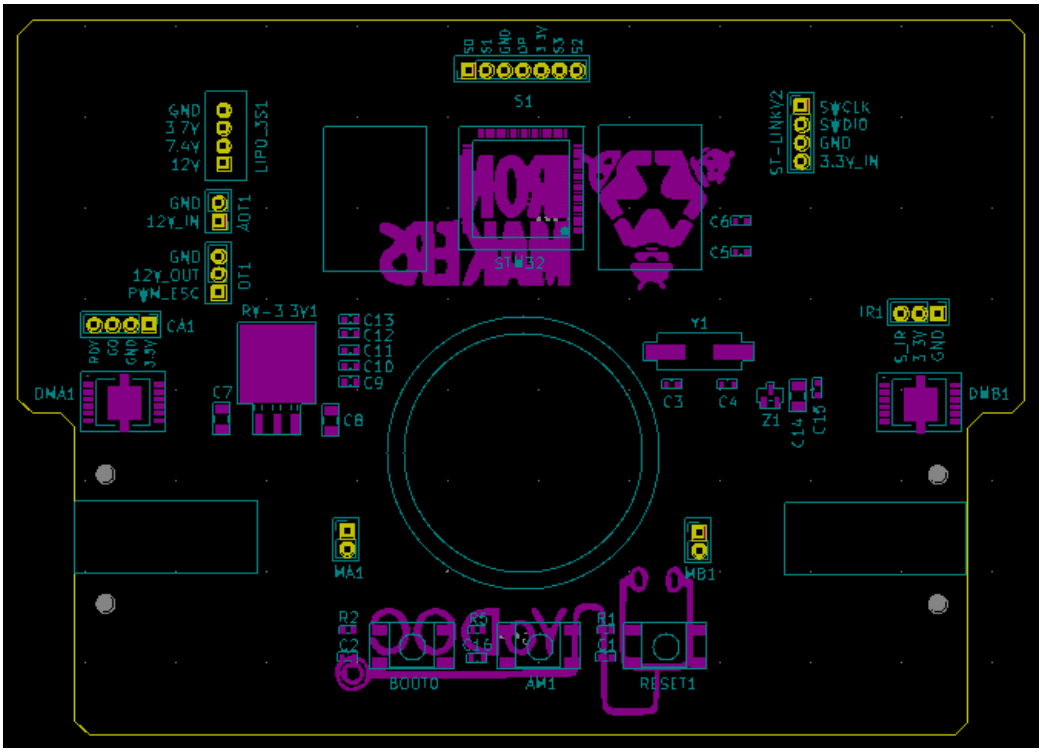

Figura 4.1. Distribución de los Componentes del Robot Seguidor de Línea en el Editor de PCB de KiCad.

Una vez teniendo la distribución de los componentes, se realizó el ruteo de las pistas de forma manual. En la figura 4.2 se muestran las conexiones de los componentes por medio de las pistas utilizando dos capas: F.Cu (pistas de color rojo ubicadas en la cara frontal de la PCB) y B.Cu (pistas de color verde ubicadas en la cara trasera de la PCB). Para realizar algunas conexiones entre los componentes se utilizó la herramienta Via, la cual es una perforación en la placa que nos permite realizar conexiones entre dos capas diferentes; en este caso la cara frontal y la cara trasera de nuestra PCB.

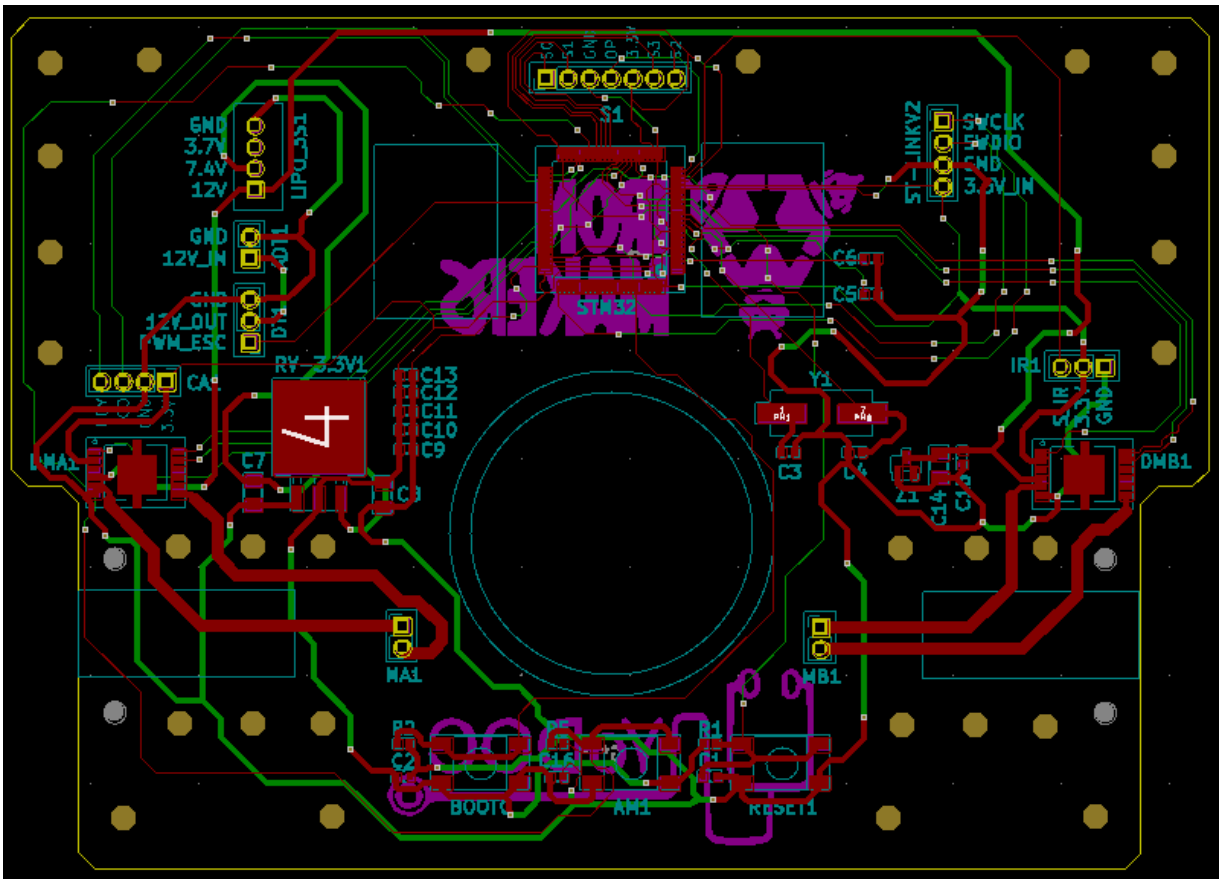

Figura 4.2. Ruteo de Pistas en el Editor de PCB de KiCad.

Una de las ventajas con las que cuenta el Software de KiCad es que proporciona una vista en 3D del proyecto que se está realizando. La figura 4.3 y 4.4 muestran un bosquejo en 3D de la cara frontal y la cara trasera de nuestra PCB.

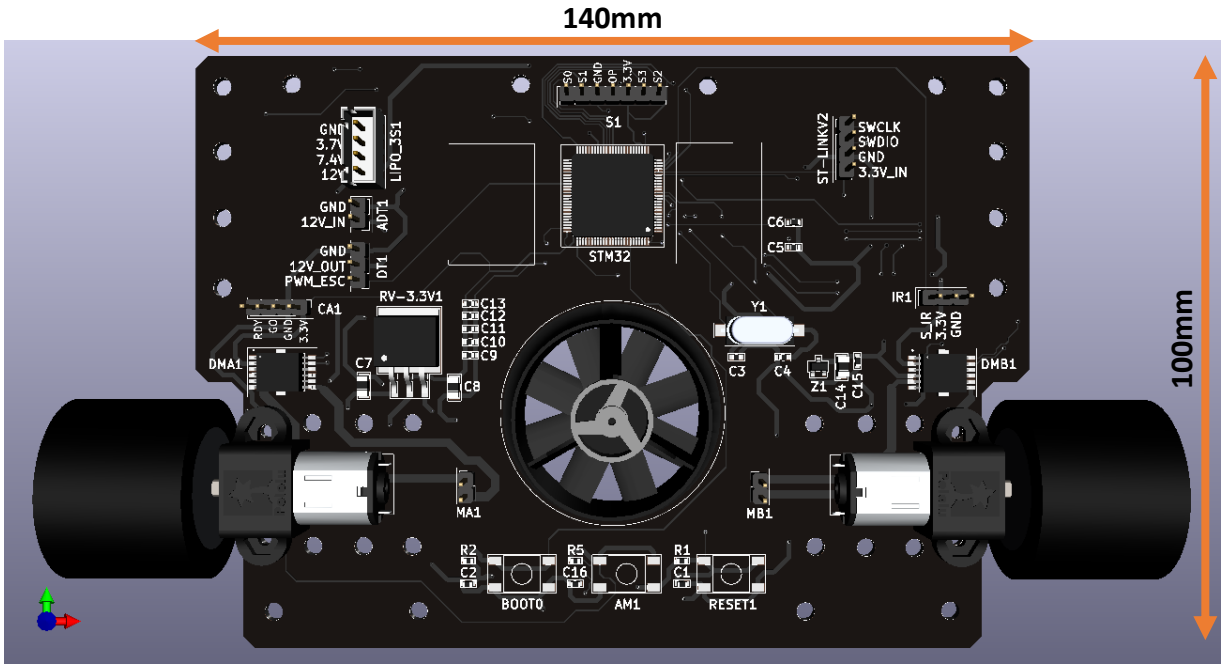

Figura 4.3. Vista Frontal en 3D de la PCB del Robot Seguidor de Línea en KiCad.

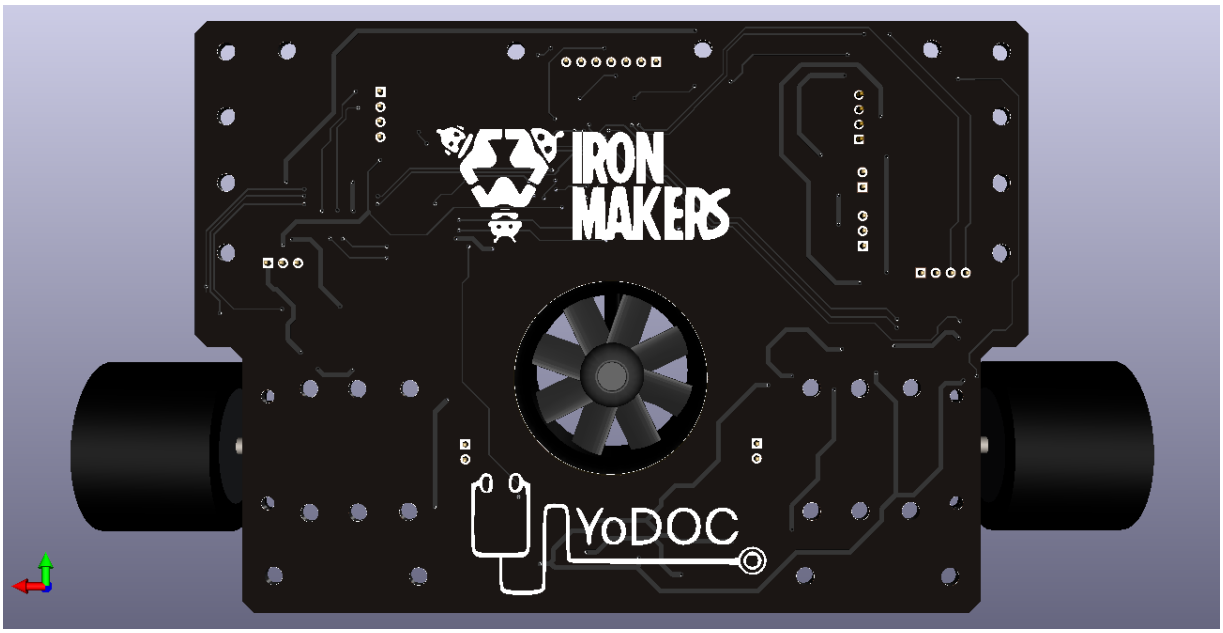

Figura 4.4. Vista Trasera en 3D de la PCB del Robot Seguidor de Línea en KiCad.

En las figuras 4.5 y 4.6 se pueden apreciar los resultados obtenidos de la fabricación de la PCB de nuestro Robot Seguidor de Línea. Específicamente la cara frontal y la cara trasera de la PCB.

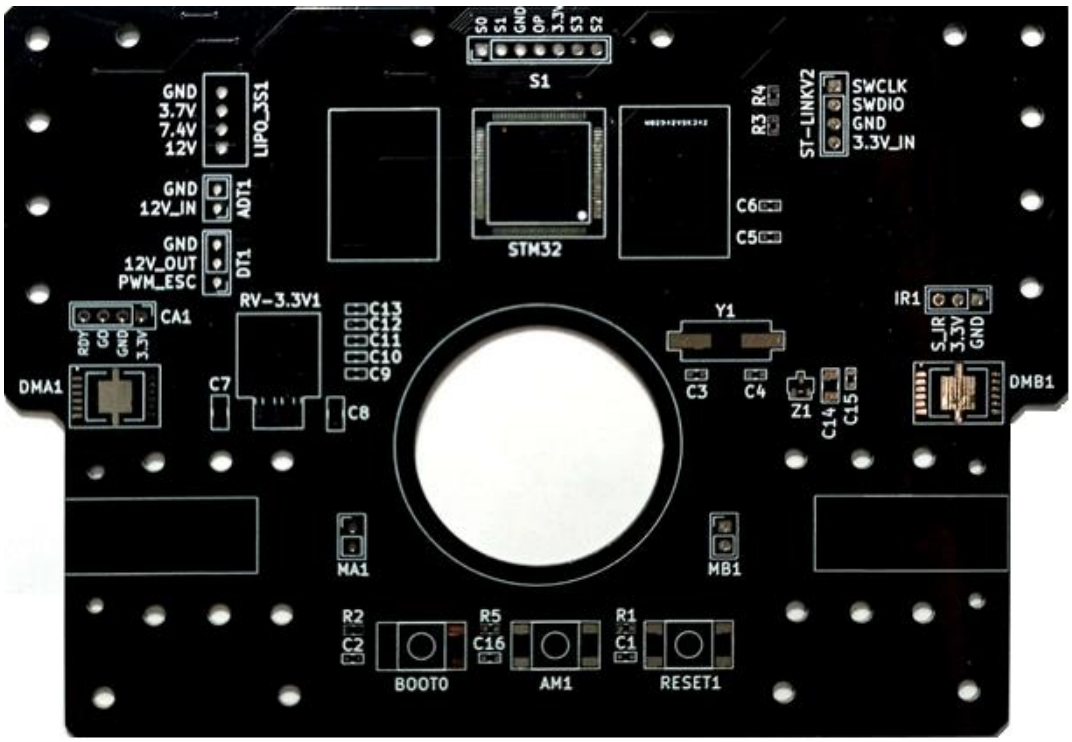

Figura 4.5. Vista Frontal de la PCB Fabricada del Robot Seguidor de Línea.

Figura 4.6. Vista Trasera de la PCB Fabricada del Robot Seguidor de Línea.

En las figuras 4.7 y 4.8 se pueden apreciar los resultados obtenidos del ensamble final de nuestro Robot Seguidor de Línea. Específicamente la parte frontal y la parte trasera del Robot.

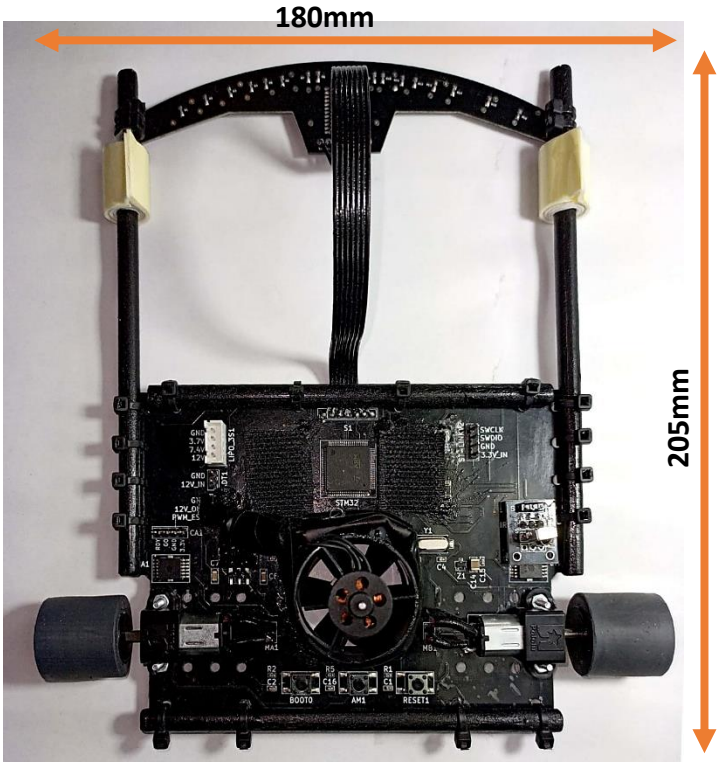

Figura 4.7. Vista Frontal del Robot Seguidor de Línea ya Ensamblado.

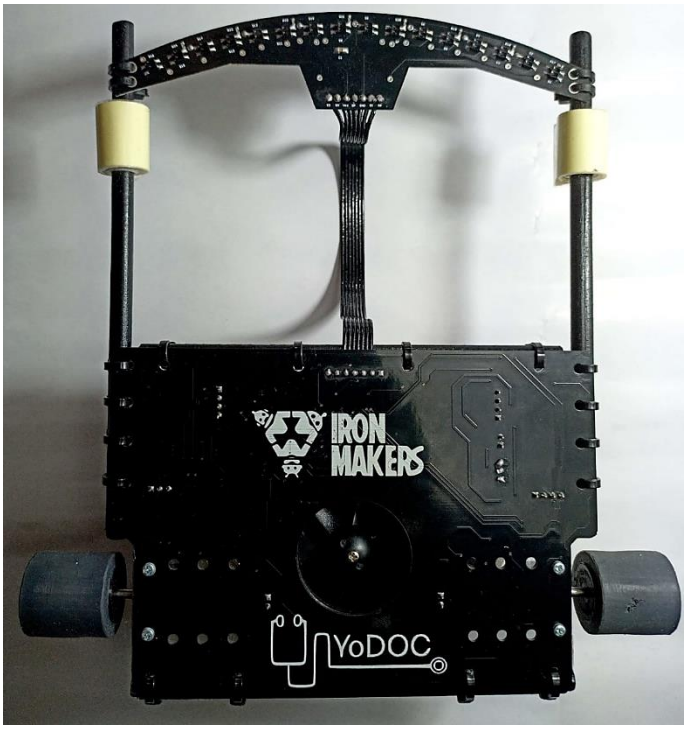

Figura 4.8. Vista Trasera del Robot Seguidor de Línea ya Ensamblado.

En la figura 4.9 se puede apreciar el resultado obtenido de la construcción del Módulo Complementario del Programador/Debugger ST-Link V2; cuyo circuito se puede apreciar en la figura 3.22.

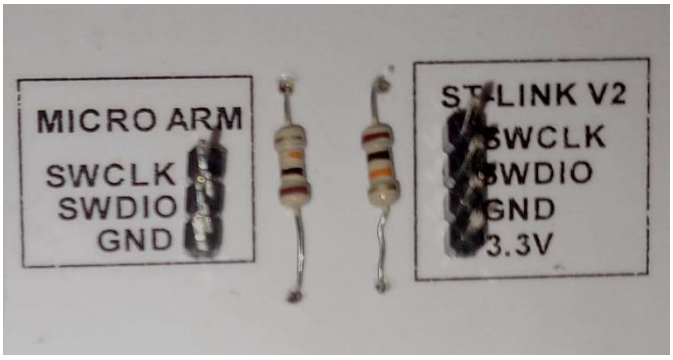

Figura 4.9. Módulo Complementario del Programador/Debugger ST-Link V2.

En la figura 4.10 se muestran las dos plataformas completas del robot tipo Autonomous Line Follower, las cuales serán donadas para los estudiantes de la Ingeniería en Mecatrónica del Tecnológico de Atlixco (específicamente para la Rama Estudiantil IEEE ITSA y el Capítulo Estudiantil Mecatrolynxs Robotic). Esto gracias al apoyo económico brindado por parte de la noble y filantrópica organización YoDOC, la cual destino un fondo para este proyecto que asciende a los $11,000.00 MXN.

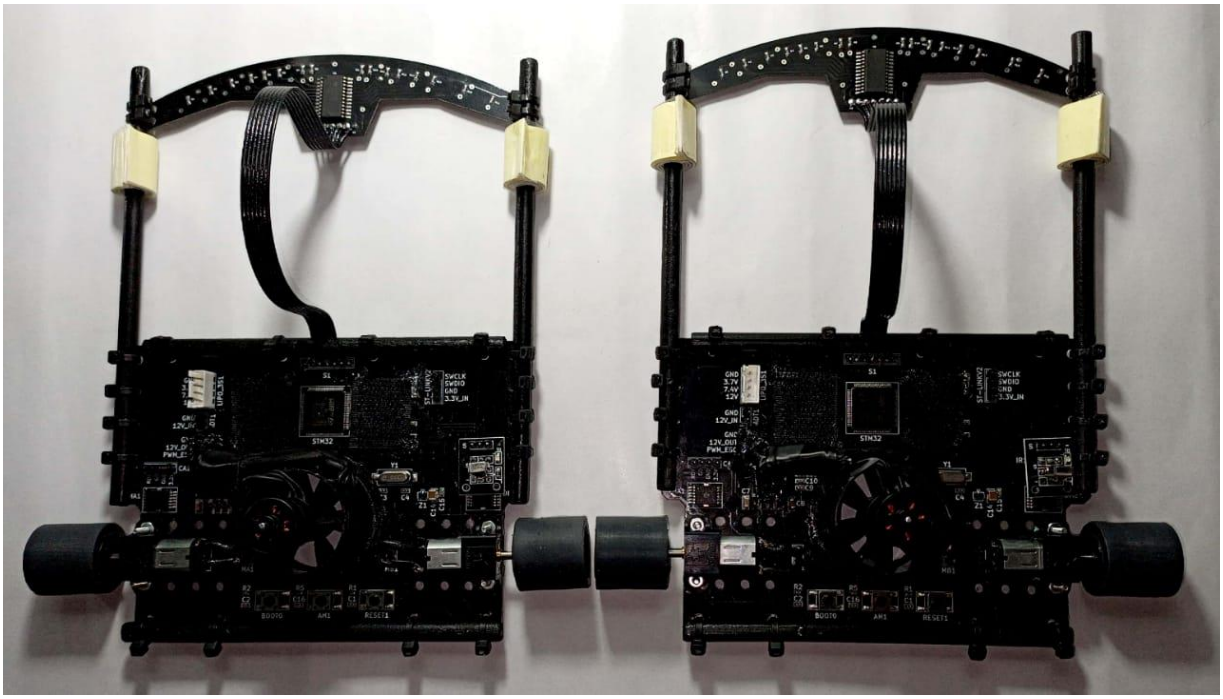

Figura 4.10. Robots Seguidores de Línea que Serán Donados a los Estudiantes de la Ingeniería en Mecatrónica del Tecnológico de Atlixco.

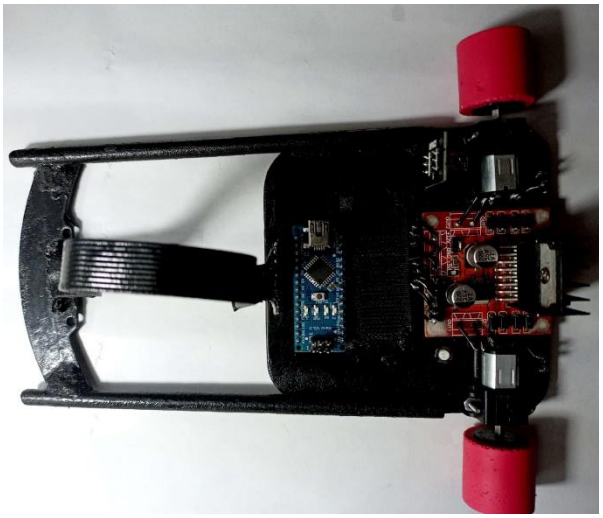

Figura 4.11. Robot Seguidor de Línea que Será Donado a la División de Ingeniería en Mecatrónica del Tecnológico de Atlixco.

4.2 Diseño/Construcción de la Pista de Competencias y Pruebas de Validación del Funcionamiento del Robot

Para la realización de las pruebas de funcionamiento del Robot Seguidor de Línea, se optó por emplear dos diferentes pistas: la pista del Torneo ROBOTIC PEOPLE FEST Feria Universitaria Colombia 2020 y la pista del Torneo Intelibots Tournament Virtual 2020; ambas pistas correspondientes a la categoría de Velocista PRO con Turbina.

La figura 4.12 muestra la pista acotada del Torneo ROBOTIC PEOPLE FEST Feria Universitaria Colombia 2020. La cual consta de un óvalo construido a partir de 2 semi círculos de 50 cm de radio, conectados por líneas rectas. Donde, la línea del recorrido debe ser de 1.5cm (±0.5cm) de ancho [25].

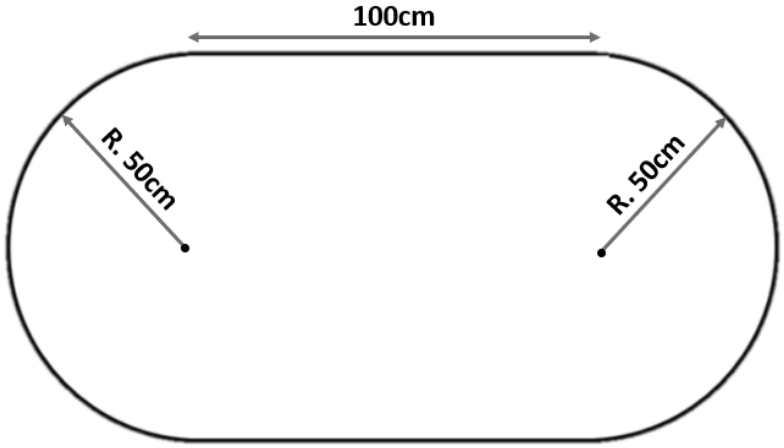

Figura 4.12. Pista Acotada del Torneo ROBOTIC PEOPLE FEST Feria Universitaria Colombia.

La figura 4.13 muestra la pista construida del Torneo ROBOTIC PEOPLE FEST Feria Universitaria Colombia 2020 en una placa de melamina blanca de 245x124cm.

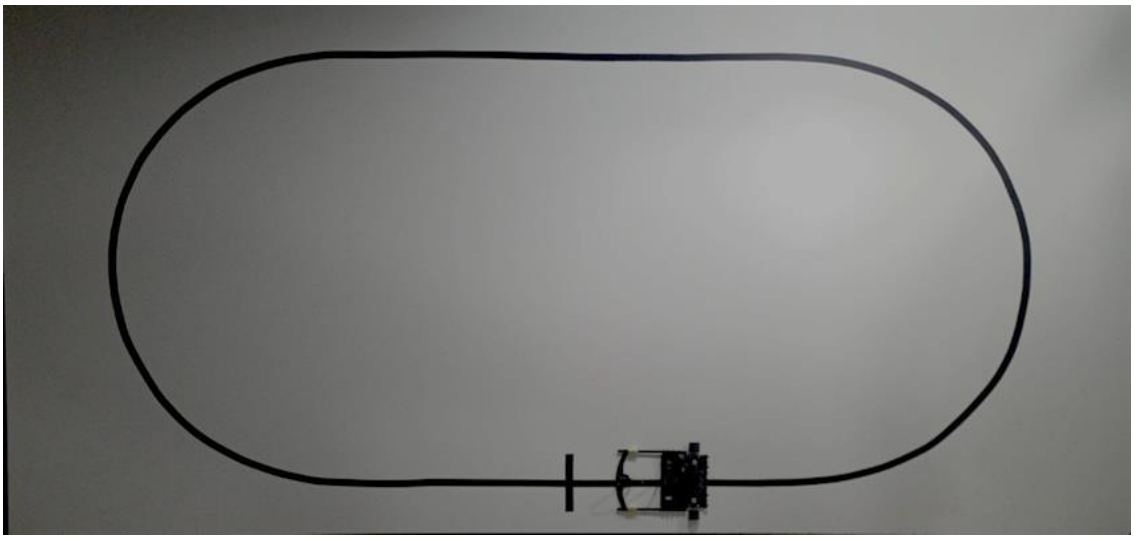

Figura 4.13. Pista Construida del Torneo ROBOTIC PEOPLE FEST Feria Universitaria Colombia.

La figura 4.14 muestra la pista acotada del Torneo Intelibots Tournament Virtual 2020. Donde, la línea del recorrido debe ser de 1.5cm (±0.5cm) de ancho [26].

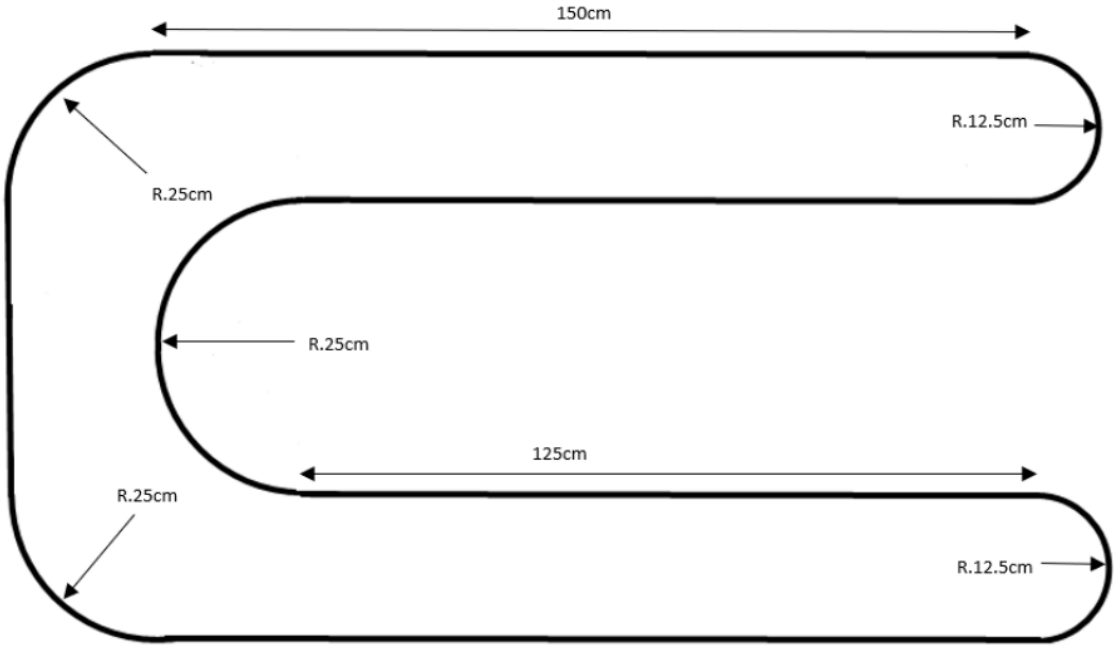

Figura 4.14. Pista Acotada del Torneo Intelibots Tournament Virtual 2020.

La figura 4.15 muestra la pista construida del Torneo Intelibots Tournament Virtual 2020 en una placa de melamina blanca de 245x124cm.

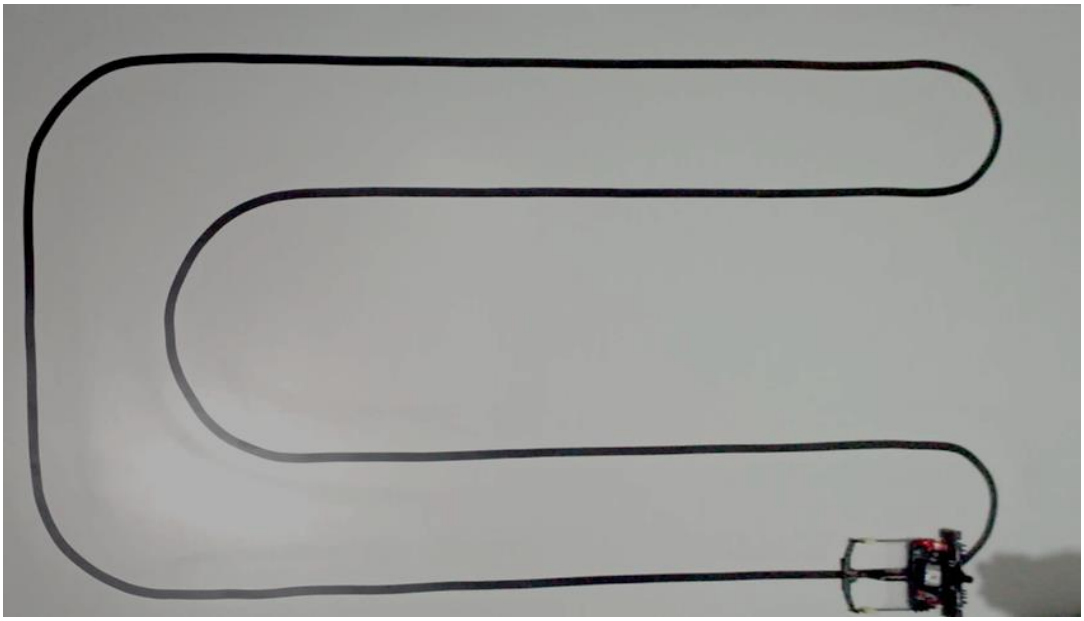

Figura 4.15. Pista Construida del Torneo Intelibots Tournament Virtual 2020.

A continuación, en la tabla 6 se presentan las pruebas realizadas del Robot Seguidor de Línea sobre la pista de competencias diseñada para el Torneo ROBOTIC PEOPLE FEST Feria Universitaria Colombia 2020. Cabe resaltar que el tiempo fue medido con un cronómetro, tal y como se realiza en los torneos presenciales de seguidores de línea.

Tabla 6. Pruebas de Funcionamiento del Robot Seguidor de Línea Sobre la Pista del Torneo ROBOTIC PEOPLE FEST Feria Universitaria Colombia 2020.

Pista para el Torneo ROBOTIC PEOPLE FEST Feria Universitaria Colombia 2020			
No. de Prueba	**No. de Vueltas**	**Distancia Recorrida**	**Tiempo Realizado**
1	5	5.14m	13.84s
2	5	5.14m	14.07s
3	5	5.14m	15.01s
4	5	5.14m	14.50s
5	5	5.14m	14.74s

Teniendo un promedio de tiempo de recorrido de 14.43 segundos por una distancia de 5.14m.

En la tabla 7 se presentan las pruebas realizadas del Robot Seguidor de Línea sobre la pista de competencias diseñada para el Torneo Intelibots Tournament Virtual 2020. Cabe resaltar que el tiempo fue medido con un cronómetro, tal y como se realiza en los torneos presenciales de seguidores de línea.

Tabla 7. Pruebas de Funcionamiento del Robot Seguidor de Línea Sobre la Pista del Torneo Intelibots Tournament Virtual 2020.

Pista para el Torneo Intelibots Tournament Virtual 2020			
No. de Prueba	**No. de Vueltas**	**Distancia Recorrida**	**Tiempo Realizado**
1	5	8.35m	22.74s
2	5	8.35m	23.02s
3	5	8.35m	23.15s
4	5	8.35m	22.87s
5	5	8.35m	22.93s

Teniendo un promedio de tiempo de recorrido de 22.94 segundos por una distancia de 8.35m.

4.3 Participación en el ROBOTIC PEOPLE FEST Feria Universitaria y en el Intelibots Tournament Virtual 2020

A continuación, en la figura 4.16 se puede observar la participación que se tuvo con el Robot Seguidor de Línea desarrollado en este proyecto, en el ROBOTIC PEOPLE FEST Feria Universitaria; el cual fue realizado de manera virtual desde Colombia del 06 al 13 de noviembre de 2020. La categoría en la que se tuvo participación fue Velocistas Pro con Turbina. Cabe destacar que esta es la categoría más profesional dentro de las competencias de seguidores de línea. El resultado que se obtuvo en este Torneo fue el Tercer Lugar con un número de 5 vueltas en un tiempo de 13.6 segundos (ver figura 4.17).

Figura 4.16. Participación en el ROBOTIC PEOPLE FEST Feria Universitaria Colombia 2020.

Figura 4.17. Reconocimiento por Haber Obtenido el 3er Lugar en la Categoría Seguidor Velocista Pro Virtual con Turbina en el ROBOTIC PEOPLE FEST Feria Universitaria Colombia 2020.

A continuación, en la figura 4.18 se puede observar la participación que se tuvo con el Robot Seguidor de Línea desarrollado en este proyecto, en el Intelibots Tournament Virtual 2020; el cual fue realizado de manera virtual desde la ciudad de Teotihuacán, México del 13 al 15 de noviembre de 2020.

La categoría en la que se tuvo participación fue Velocistas Pro con Turbina. Cabe destacar que en esta categoría es de suma importancia que el robot autónomo sea capaz de recorrer el circuito establecido en el menor tiempo posible. El resultado que se obtuvo en este Torneo fue el Segundo Lugar con un número de 5 vueltas en un tiempo de 22 segundos (ver figura 4.19).

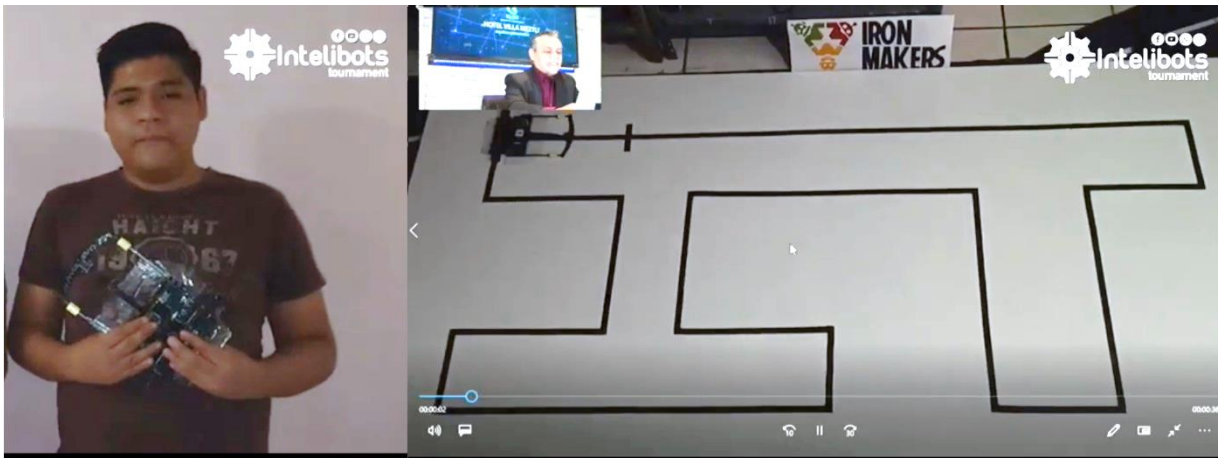

Figura 4.18. Participación en el Intelibots Tournament Virtual 2020.

Figura 4.19. Reconocimiento por Haber Obtenido el 2do Lugar en la Categoría Seguidor Velocista Pro con Turbina en el Intelibots Tournament Virtual 2020.

Conclusiones

La categoría de Robots Seguidores de Línea Velocistas Pro con Turbina es una de las más importantes en los torneos de robótica a nivel Nacional como Internacional. Generando con ello una gran demanda en participación tanto de escuelas de educación pública como de escuelas especializadas en robótica. Esta categoría resulta sumamente fascinante para los amantes de la robótica competitiva, puesto que implica una gran complejidad de diseño y construcción de robots.

A través de la realización de este proyecto, se pudo diseñar e implementar una plataforma robótica tipo Autonomous Line Follower utilizando un microcontrolador ARM Cortex M4, con el propósito de que los estudiantes del Tecnológico de Atlixco puedan participar de forma competitiva a nivel profesional en los diferentes torneos de robótica que existen tanto nacionales como internacionales. Reduciendo de esta forma, los costos de fabricación que poseen los Robots Seguidores de Línea, cuyo valor en el mercado suele rondar entre los $5,000 (calidad media-baja) a $15,000 (calidad media-alta) por unidad. Dando como resultado, un Robot Seguidor de Línea Velocista Pro con Turbina con un valor de $5,730.63 (calidad media-alta).

Por otro lado, se logró diseñar e implementar un algoritmo de control, el cual es capaz de establecer la autonomía del robot móvil. Concluyendo de esta forma que el proyecto cumple los objetivos planteados. Cabe destacar que el controlador PD es el más adecuado para este tipo de robots y es con el que cuenta nuestro Robot Seguidor de Línea velocista. Es preciso señalar que con el Robot producto de este proyecto, se participó en dos de los torneos más importantes del año dentro de la categoría “Seguidores de Línea con Turbina”, donde se obtuvieron los primeros lugares. Evidenciando de esta manera, el gran potencial que posee nuestro Robot a nivel profesional.

Dados los resultados obtenidos, se ha decidido realizar la siguiente propuesta para continuar y mejorar con el trabajo realizado:

- La principal área de oportunidad identificada es la mejora en cuanto a la potencia de los motores. Es decir, buscar unos motores que tengan una cantidad mayor de RPM y un mayor torque, ya que en las competencias de velocidad y destreza (ángulos de 90°) se requiere de un alto torque y de una alta velocidad para lograr llegar a obtener el 1er Lugar. Sin embargo, es importante considerar también el peso de los motores, debido a que aún menor peso, se requiere menos esfuerzo y al hacer menos esfuerzo, se necesita menos corriente, y al necesitar menos corriente, se podrán ocupar baterías más livianas.

Competencias Desarrolladas

A través de la realización de este proyecto, se desarrollaron y aplicaron tanto competencias personales como competencias técnicas. La habilidad de trabajar en equipo fue una competencia indispensable empleada a lo largo del proyecto. Puesto que, se requirió de la colaboración de diversas personas, por lo cual fue muy importante la coordinación y participación en equipo, a pesar de que la asignación del proyecto fue individual. Asi mismo, una de las habilidades más importantes que se desarrolló como producto de este proyecto fue el aprender a ser autodidacta, esto debido a las condiciones desfavorables que se tenían derivado de la pandemia.

Dentro de las habilidades técnicas aplicadas, se encuentran materias como instrumentación y electrónica, las cuales ayudaron al análisis del funcionamiento de los sensores optoreflectivos para poder emplearlos de manera correcta en el proyecto. Materias como microcontroladores, control digital y robótica, fueron el eje central en el desarrollo del programa con el cual fue posible establecer la autonomía del robot móvil. Asi mismo, asignaturas como formulación y evaluación de proyectos fueron fundamentales para la planeación, estructuración y ejecución del proyecto.

Fuentes de Información

[1] "Antecedentes", *Itsatlixco.edu.mx*. [En línea]. Disponible en: https://www.itsatlixco.edu.mx/antecedentes.html. [Accedido: 05- octubre- 2020].

[2] "Política, Misión, Visión y Valores", *Itsatlixco.edu.mx*. [En línea]. Disponible en: https://www.itsatlixco.edu.mx/misionvv.html. [Accedido: 05- octubre- 2020].

[3] "Ingeniería Mecatrónica", *Itsatlixco.edu.mx*. [En línea]. Disponible en: https://www.itsatlixco.edu.mx/mecatronica.html. [Accedido: 06- octubre- 2020].

[4] "About KiCad", *Kicad.org*. [En línea]. Disponible en: https://kicad.org/about/kicad/. [Accedido: 06- octubre- 2020].

[5] R. Murphy, *Introduction to AI robotics*. Cambridge, MA: MIT Press, 2005, pp. 1-40.

[6] A. Ollero Baturone, *Robótica: Manipuladores y Robots Móviles*. Barcelona: Marcombo, 2001, pp. 8-13.

[7] "Definiciones - Controlador PD - item Glossar", *Glossar.item24.com*. [En línea]. Disponible en: https://glossar.item24.com/es/indice-de-glosario/articulo/item//controlador-pd.html#:~:text=Un%20controlador%20PD%20(proporcional%2Ddiferencial,el%20error%20de%20control%20cambia. [Accedido: 27- octubre- 2020].

[8] R. Ramos Lara, "Sistemas Digitales de Control en Tiempo Discreto", *Upcommons.upc.edu*, 2007. [En línea]. Disponible en: https://upcommons.upc.edu/bitstream/handle/2117/6123/TEMA6.pdf. [Accedido: 28- octubre- 2020].

[9] M. Pakdaman, M. M. Sanaatiyan & M. R. Ghahroudi, *A line follower robot from design to implementation: Technical issues and problems*. The 2nd International Conference on Computer and Automation Engineering (ICCAE), Singapore, 2010, pp. 5-9.

[10] D. Jahshan & P. Hutchinson, "Comenzando en KiCad", *Docs.kicad.org*, 2015. [En línea]. Disponible en: https://docs.kicad.org/4.0/es/getting_started_in_kicad/getting_started_in_kicad.pdf. [Accedido: 03- noviembre- 2020].

[11] "STM32CubeIDE - STMicroelectronics", *STMicroelectronics*. [En línea]. Disponible en: https://www.st.com/en/development-tools/stm32cubeide.html#overview. [Accedido: 03- noviembre- 2020].

[12] "Barra de 16 sensores optoreflectivos (QRE1113) multiplexados (74HC4067)", *Ingenieromaker.com*. [En línea]. Disponible en: https://www.ingenieromaker.com/product-page/barra-de-16-sensores-optoreflectivos-qre1113-multiplexados-74hc4067-1. [Accedido: 04- noviembre- 2020].

[13] "STM32F405xx STM32F407xx", *st.com*, 2020. [En línea]. Disponible en: https://www.st.com/resource/en/datasheet/dm00037051.pdf. [Accedido: 04- noviembre- 2020].

[14] "Pololu - 10:1 Micro Metal Gearmotor HP 6V", *Pololu.com*. [En línea]. Disponible en: https://www.pololu.com/product/999. [Accedido: 04- noviembre- 2020].

[15] "IFX9201SG 6 A H-Bridge with SPI", *Infineon.com*, 2015. [En línea]. Disponible en: https://www.infineon.com/dgdl/Infineon-IFX9201SG-DS-v01_01-EN.pdf?fileId=5546d4624cb7f111014d2e8916795dea. [Accedido: 04- noviembre- 2020].

[16] "Sensor Receptor Infrarrojo IR", *cdmxelectronica.com*. [En línea]. Disponible en: https://cdmxelectronica.com/producto/sensor-receptor-infrarrojo-ir-modulo-ky-022/. [Accedido: 04- noviembre- 2020].

[17] "LM1086 1.5-A Low Dropout Positive Voltage Regulators", *Ti.com*, 2015. [En línea]. Disponible en: https://www.ti.com/lit/ds/symlink/lm1086.pdf?HQS=TI-null-null-mousermode-df-pf-null-wwe&ts=1598737255586&ref_url=https%253A%252F%252Fwww.mouser.mx%252F. [Accedido: 04- noviembre- 2020].

[18] "QX Motor QF1611 1311 7000KV 30mm EDF 6 Blades Brushless Motor for RC Airplane - RcMoment.com", *rcmoment.com*. [En línea]. Disponible en: https://www.rcmoment.com/p-rm7352.html. [Accedido: 04- noviembre- 2020].

[19] "Controlador de velocidad electrónico ESC 12A EMAX para Brushless", *taloselectronics.com*. [En línea]. Disponible en: https://www.taloselectronics.com/products/controlador-de-velocidad-electronico-esc-12a-emax-para-brushless. [Accedido: 04- noviembre- 2020].

[20] "Batería LiPo Turnigy nano-tech 460mAh 3S 25~40C | SANDOROBOTICS", *Sandorobotics.com*. [En línea]. Disponible en: https://sandorobotics.com/producto/n460-3s/. [Accedido: 05- noviembre- 2020].

[21] J. García Mejía, "Diseño En Altium De La PCB De Un Robot Seguidor De Línea Basado En La Plataforma Electrónica OPHYRA De La Empresa INTESC", Instituto Tecnológico Superior De Atlixco, Atlixco, Puebla, 2019.

[22] "OPHYRA", *Intesc.mx*, 2019. [En línea]. Disponible en: https://www.intesc.mx/wp-content/uploads/2019/05/ManualOphyraRevD.pdf. [Accedido: 05- noviembre- 2020].

[23] "ST-Link V2", *Sunrom.com*. [En línea]. Disponible en: https://www.sunrom.com/p/st-link-v2. [Accedido: 05- noviembre- 2020].

[24] "Programación de un ESC con Arduino", *Robologs.net*. [En línea]. Disponible en: https://robologs.net/2016/02/01/programacion-de-un-esc-con-arduino/. [Accedido: 05- noviembre- 2020].

[25] "Reto Virtual Velocista Pro Robotic People Fest Virtual", *Drive.google.com*. [En línea]. Disponible en: https://drive.google.com/drive/folders/12omTCAsbC2I_RezfpK6Wmq17y5fs6R4z. [Accedido: 30- noviembre- 2020].

[26] "Reglamento Seguidor de Líneas con Turbina", *Intelibotstournament.com*. [En línea]. Disponible en: https://intelibotstournament.com/wp-content/uploads/2020/09/REGLAMENTO-SEGUIDOR-DE-LINEAS-CON-TURBINA.pdf. [Accedido: 30- noviembre- 2020].

Anexos

Anexo 1. Código del STM32F407VG en el IDE de STM32CubeIDE: LineFollowerPro.c

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * <h2><center>&copy; Copyright (c) 2020 STMicroelectronics.
  * All rights reserved.</center></h2>
  *
  * This software component is licensed by ST under BSD 3-Clause license,n
  * the "License"; You may not use this file except in compliance with the
  * License. You may obtain a copy of the License at:
  *                        opensource.org/licenses/BSD-3-Clause
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "adc.h"
#include "tim.h"
#include "gpio.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
```

```
/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */
#define white 1
#define black 0
#define OUT_LINE  100.0
#define MAX_SPEED 225
#define MED_SPEED 175
//----------------------
#define ON 33456254
#define OFF 33441974
/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/

/* USER CODE BEGIN PV */
/**************************************/
enum {CENTER,RIGHT,LEFT} out_state = CENTER;
enum {HOME,STARTING,RUN} state = HOME;
int sensors_qtr[16] = {0};
float error_now = 0.0;
float error_last = 0.0;
int       speed_1 = 0;
int speed_2 = 0;
int U = 0;
float KP = 78.7781;//82.717;//78.7781;//
float KD = 216.6371;//227.469;//216.6371;//
//float KB = 0.25;//
int brake = 0;
/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

void Button();
int read_adc();
```

```
void read_sensors_qtr(int *sensors);
float get_error(int *sensors,int background);
void Motor_L (signed int speed);
void Motor_R(signed int speed);
void turbine(int speed);
/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_ADC1_Init();
  MX_TIM3_Init();
  MX_TIM4_Init();
  /* USER CODE BEGIN 2 */
  HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_1);
  HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_2);
  HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_1);
  HAL_GPIO_WritePin(GPIOC, DIS_A_Pin,GPIO_PIN_RESET);
```

```
HAL_GPIO_WritePin(GPIOB, DIS_B_Pin,GPIO_PIN_RESET);
turbine(12);
HAL_Delay(8000);
/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
  /* USER CODE END WHILE */
  /* USER CODE BEGIN 3 */
        Button();
        switch (state)
        {
                case HOME:
                        turbine(12);
                        Motor_R(0);
                        Motor_L(0);
                        break;
                case STARTING:
                        turbine(42);
                        state = RUN;
                        HAL_Delay(650);
                        break;
                case RUN:
                        //Read real error -10 to 10
                        error_now =  get_error(sensors_qtr,white);
                        //Out line
                        if(error_now == OUT_LINE)
                        {
                                //Out line state machine//
                                switch (out_state)
                                {
                        case CENTER:
                                speed_1 = MED_SPEED;
                speed_2 = MED_SPEED;
                break;
                        case RIGHT:
                                speed_1 = MAX_SPEED;
                                speed_2 = (0-MAX_SPEED);
                                break;
                        case LEFT:
                speed_1 = (0-MAX_SPEED);
```

```
                        speed_2 = MAX_SPEED;
                        break;
                                }
                        }
                        //On line
                        else
                        {
                                U  = (int)(KP*error_now + KD*(error_now-error_last));
                        //brake = (int)(50.0*(1.0-exp(0.0 - KB*fabs(error_now))));
                        //speed_1  = (MED_SPEED-brake) + U;
                //speed_2  = (MED_SPEED-brake) - U;
                                speed_1  = MED_SPEED + U;
                                speed_2  = MED_SPEED - U;
                                error_last = error_now;
                        }
                        Motor_R(speed_2);
                        Motor_L(speed_1);
                        break;
                  }
          }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);
  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
```

```
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSI;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
  {
    Error_Handler();
  }
}

/* USER CODE BEGIN 4 */
int read_adc()
{
	int sensor_value = 4095;
	HAL_ADC_Start(&hadc1);
	if(HAL_ADC_PollForConversion(&hadc1,5) == HAL_OK)
		sensor_value  = (int)(HAL_ADC_GetValue(&hadc1));
	//HAL_ADC_Stop(&hadc1);
	return sensor_value;
}
//---------------------------------------------------------------------
void read_sensors_qtr(int *sensors)
{
	for(int i = 0;i<16;i++)
	{
		HAL_GPIO_WritePin(GPIOE, S0_Pin, i & 0x01);
		HAL_GPIO_WritePin(GPIOE, S1_Pin, i & 0x02);
		HAL_GPIO_WritePin(GPIOE, S2_Pin, i & 0x04);
		HAL_GPIO_WritePin(GPIOE, S3_Pin, i & 0x08);
		sensors[i] = read_adc();
	}
}
//---------------------------------------------------------------------
float get_error(int *sensors,int background)
```

```
{
        int max;
        int min;
        int threshold;
        int range;
        int bit_sensor[16];
        int sum = 0;
        int weigth[8] = {8,7,6,5,4,3,2,1};
        int errorLeft = 0;
        int errorRight = 0;
        float error = 0.0;
        //Read 8 samples from each sensor
        read_sensors_qtr(sensors);
        max = min = sensors[0];
        for(int i=1;i<16;i++)
        {
                if(sensors[i] > max)
                        max = sensors[i];
                if(sensors[i] < min)
                        min = sensors[i];
        }
        range = max-min;
        if(range > 400)
        {
                threshold = (range/2)+min;
                for(int i=0;i<16;i++)
                {
                        if(background)
                                bit_sensor[i] = (sensors[i] < threshold) ? 1 : 0;
                        else
                                bit_sensor[i] = (sensors[i] > threshold) ? 1 : 0;
                }
                for(int i=0;i<8;i++)
                {
                        errorLeft += bit_sensor[i]*weigth[i];
                        errorRight += bit_sensor[15-i]*weigth[i];
                }
                for(int i=0;i<16;i++)
                        sum += bit_sensor[i];
                error = (float)(errorRight-errorLeft)/(float)(sum);
      out_state = ((error <= 1.0)&&(error >=(0-1.0)))  ? CENTER : out_state;
      out_state = ((error > 1.0)&&(error <=8.0))       ?  RIGHT: out_state;
      out_state = ((error <(0-1.0))&&(error >=(0-8.0))) ? LEFT: out_state;
```

```
        return error;
          }
          else
                  return OUT_LINE;
          return error;
}

//-------------------------------------------------------------------
void Motor_L(signed int speed)
{
          if(!speed)
          {
                  HAL_GPIO_WritePin(GPIOC, DIR_A_Pin,GPIO_PIN_RESET);
                  __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_1,0);

          }
          else
          {
                  speed = (speed >= MAX_SPEED) ? MAX_SPEED : speed;
                  if (speed >=1)
                  {
                          HAL_GPIO_WritePin(GPIOC, DIR_A_Pin,GPIO_PIN_SET);
                          __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_1,speed);
                  }
                  else
                  {
                          speed *= (0-1);
                          speed = (speed >= MAX_SPEED) ? MAX_SPEED : speed;
                          HAL_GPIO_WritePin(GPIOC, DIR_A_Pin,GPIO_PIN_RESET);
                          __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_1,speed);
    }
  }
  return;
}
//-------------------------------------------------------------------
void Motor_R(signed int speed)
{
          if(!speed)
          {
                  HAL_GPIO_WritePin(GPIOB, DIR_B_Pin,GPIO_PIN_RESET);
                  __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_2,0);

          }
```

```
        else
        {
                speed = (speed >= MAX_SPEED) ? MAX_SPEED : speed;
                if (speed >=1)
                {
                        HAL_GPIO_WritePin(GPIOB, DIR_B_Pin,GPIO_PIN_SET);
                        __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_2,speed);
                }
                else
                {
                        speed *= (0-1);
                        speed = (speed >= MAX_SPEED) ? MAX_SPEED : speed;
                        HAL_GPIO_WritePin(GPIOB, DIR_B_Pin,GPIO_PIN_RESET);
                        __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_2,speed);
    }
  }
  return;
}
//-------------------------------------------------------------------
void turbine(int speed)
{
        if(speed < 0)
                speed = 0;
        else if(speed > 100)
                speed = 100;
        speed += 100;
        __HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_1,speed);
        return;
}
//-----------------------------------------------------------------------------------
void Button()
{
        if(HAL_GPIO_ReadPin(AM_GPIO_Port, AM_Pin))
        {
          state = STARTING;
          HAL_Delay(250);
        }
}
/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
```

```
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */

  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     tex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */

/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```