Control de velocidad de motores para navegación de robots (Diciembre de 2020)

Autor: G. Cervetti, Docente: Ing. A. Aguilar, Asignatura: Sistemas de control, Carrera: Ing. Electrónica

Resumen – El presente informe describe la implementación de un control PID para controlar la velocidad y dirección de dos motores DC. El objetivo de dicho sistema es manipular de forma precisa los movimientos de un robot que se desplaza mediante el sistema de manejo diferencial o *differential drive*. Además, el sistema de proveer una cuenta de pasos o *ticks* que permita conocer el desplazamiento y dirección del robot luego de moverse. La gestión de todo el sistema se hace mediante microcontrolador ARM de 32bits y una Raspberry PI utilizando ROS.

Índice de Términos – PID, *differential drive*, ARM, ROS

I. INTRODUCCION

Durante mi carrera como estudiante de Ingeniería Electrónica tuve la oportunidad de trabajar en algunas empresas de desarrollo enfocado en robótica. Es así como puedo mencionar mis pasantías en Alemania en Metralabs, una empresa que desarrolla robots para diversas aplicaciones. Seguidamente trabajé en el grupo de investigación GIRE que se dedica a capacitar en el uso de la robótica como herramienta de enseñanza. Finalmente, me encuentro trabajando en Ekumen desarrollando software para robots móviles.

El primer gran inconveniente por tratar en los robots móviles es precisamente como se maneja la movilidad. Se requiere gran conocimiento mecánico, así como también electrónico para lograr un desarrollo que se pueda considerar funcional. Debemos tener en cuenta que, a diferencia de un robot educativo, un robot de grado comercial para uso en situaciones reales requiere un excelente sistema de posicionamiento. Este requisito no sólo implica tener un método fiable de medición, sino que también la aplicación de la velocidad de los actuadores debe ser precisa.

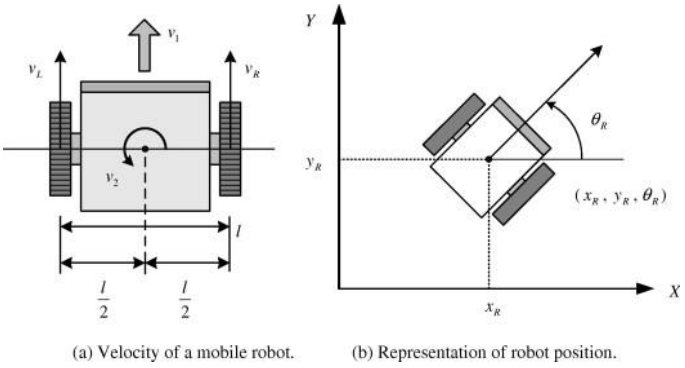

Fig. 1. Diseño típico de un robot differential drive.

Es así como nace este proyecto, desarrollar una plataforma accesible que permita replicar a escala el comportamiento de un robot real. Como apuntamos a bajo costo y a una implementación minimalista, utilizaremos un tipo de desplazamiento denominado ***differential drive***. Consiste en un par de ruedas colocadas paralelamente y un punto de apoyo o ***caster joint*** cuyo rozamiento puede despreciarse.

En el presente informe me centraré particularmente

en el diseño, control y pruebas realizadas al control de velocidad implementado a ambos motores utilizando PID.

II. Alcance

Antes de comenzar con los cálculos, será prudente establecer el alcance del proyecto. Como se mencionó anteriormente, se busca diseñar un robot de bajo costo que emule de manera precisa el comportamiento de un robot móvil de aplicaciones reales.

El diseño elegido será un sistema ***differential drive***. Los motores deben ser de tipo DC (se eligen debido a su bajo costo y facilidad para conseguirlos en el país). Cada motor se controlará mediante PID. Se debe proveer, además de algún método para asignar los valores de velocidad individualmente, una cuenta de la distancia que avanza en cada motor. Esto es para permitir la utilización de algoritmos de odometría.

Por último, para lograr emular robots comerciales de una manera más precisa, este proyecto debe permitir alojar una placa de desarrollo Raspberry PI o Jetson nano, así como también utilizar ROS (Robot Operating System) para su funcionalidad

III. Diagramación de la arquitectura

En este capítulo se definirá como se implementará la funcionalidad del robot, así como también los límites que abarcará el presente informe.

En la ***Figura 2*** se observa el diagrama general del proyecto. Para poder afrontar el diseño de este sistema tan complejo, se decidió distribuirlo en capas. Se comienza desde la capa más física, llamada ***capa 1***, para finalizar con el desarrollo de software en la ***capa 4***, que es la más abstracta.

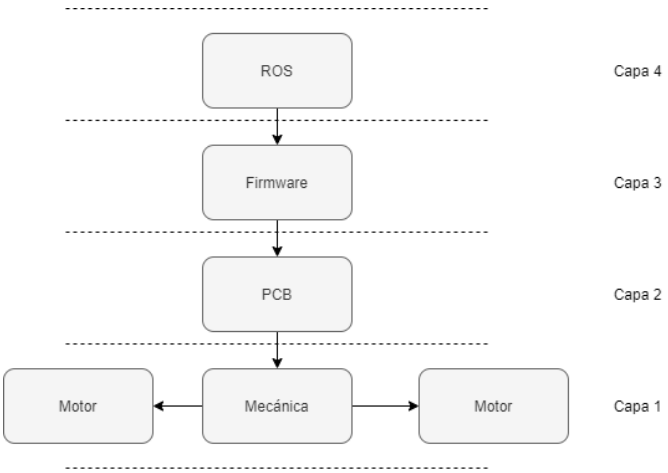

Fig. 2. Diagrama general del proyecto

A. Capa 1: Mecánica

Aquí se detallan la mecánica del sistema. Como el objetivo final de este proyecto es crear un sistema que sirva para desarrollar, se optará por un diseño minimalista, abierto para permitir el fácil acceso al interior y adaptable que permita la inclusión de futuros módulos.

Las tecnologías utilizadas para la fabricación se limitarán a impresión 3D y corte por láser para MDF/Acrílico. Esto es así ya que permitirá un diseño fácil de replicar, fácil de construir y reparar. Además, como los modelos estarán abiertos al público, será más sencillo editarlos y adaptarlos a los requerimientos del investigador. Hoy en día, es mucho más sencillo conseguir filamento para imprimir e impresoras 3D en nuestra ciudad. Incluso se cuenta con esta tecnología en nuestra facultad. El MDF por el contrario es económico, fácil de trabajar y por lo tanto excelente para esta aplicación.

El diseño del modelo se basará en robots de similares aplicaciones, se mencionará aquí al ***Turtlebot3 Burger*** [1] (ver ***Figura 3***) y al ***N6 Max*** [2] (ver ***Figura 4***).
Para dimensionar el robot se deberá tener en cuenta que deben alojar las baterías, la placa de control y una placa de desarrollo que podrá ser Raspberry Pi, Raspberry Pi Zero o Jetson Nano. Debe proveerse de los soportes necesarios para poder instalar tales elementos.

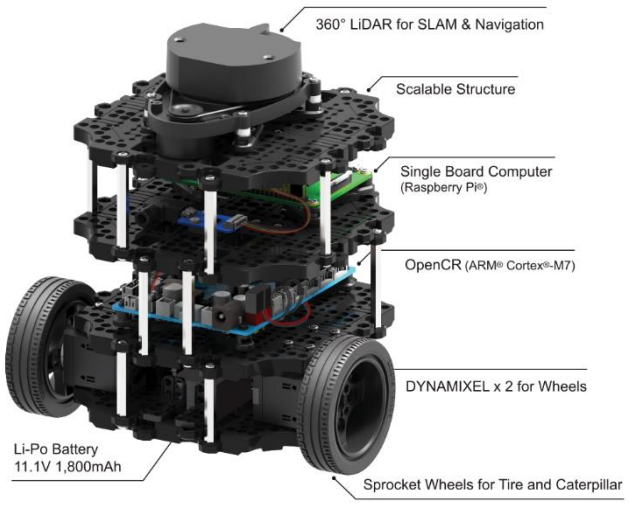

Fig. 3. Turtlebot3 Burger

Fig. 4. N6 Max

Los motores encargados de impulsar al robot serán ***JGA25-371***. Cuentan con encoders relativos en cuadratura de tipo hall. Los sensores se alimentan

con 5v y el motor se puede alimentar con hasta 24v. Se detallará más información de ellos en siguiente capítulo.

B. Capa 2: Electrónica

En esta capa se encuentra toda la electrónica que hace de interfaz entre la mecánica y las capas superiores.

El primer asunto para atender es el sistema de alimentación. La energía provendrá de ***baterías de tipo ion-litio***. Se utilizará este tipo debido a la ***excelente relación peso/almacenamiento***. Además, el costo de ellas es bajo. Como contrapartida, ***se necesitará un sistema especial para el cargado de las baterías***. El robot debe poseer algún botón para su encendido y debe existir una secuencia especial al momento del apagado (no puede ser una simple llave de 2 posiciones). Esto es así ya que la capa 4 correrá sobre un minicomputador, y los cortes repentinos de energía pueden provocar ***pérdidas de información o corrupción de la memoria***. Se trabajará con 2 niveles de tensión. Uno de ellos moverá los motores. Esta tensión será la que proviene de las 4 baterías de 3.7v en serie y será, por lo tanto, 14.8v. El otro será ***5v*** para la parte lógica. Se estima una corriente de ***1 Amper***.

En cuanto al control general, será comandado por un microcontrolador de la línea ***STMicroelectronics***. Se decidió por esta marca ya que se contaba con experiencia previa trabajando con dichos dispositivos. La comunicación con las capas superiores se realizará mediante puerto ***UART*** e implementando un simple protocolo de comunicación.

C. Capa 3: Firmware de microcontrolador

Aquí se tiene el firmware que será grabado en el microcontrolador. Como se trabajar con un microcontrolador de ***STmicroelectronics***. Se preferirá trabajar con las herramientas que este provee. El código se implementará en ***ANSI C***. Para lograr una medición fiable de la velocidad de los motores, se utilizarán ***Timers*** en lugar de

retardos por software. Será requisito el uso de ***RTOS*** (Real Time Operating System) ya que se ejecutarán diversas acciones simultáneamente que se deben ejecutar en tiempos exactos.

D. Capa 4: Software

Por último, parte de la implementación se hará en un microcomputador. Aquí se agrega a los requisitos utilizar ***ROS*** (Robot Operative System). Es un conjunto de paquetes y librerías open source que facilita el desarrollo de robots. Cuenta con numerosos paquetes que ayudan a resolver problemas de navegación, así como también métodos de comunicación entre paquetes [3]. Como se utiliza ampliamente en aplicaciones que requieren robots, el uso de esta plataforma acercará a los potenciales usuarios de este proyecto a situaciones reales. Se busca como prioridad utilizar la mayor parte del código en C++ 11 o superior, ya que las capacidades de procesamiento de las microcomputadoras son muy reducidas. Las tareas para realizar por parte de esta capa son la de navegación, monitoreo de los sensores y generar la interfaz de usuario.

IV. DESARROLLO MATEMÁTICO

Teniendo definido como será la arquitectura con la que se trabajará, es momento de comenzar con los desarrollos matemáticos relevantes. Se seguirá un proceso desde lo más general a lo más específico.

A. Modelo matemático para control diferencial

El primer paso es definir el comportamiento del robot. Se dijo anteriormente que la forma de desplazarse utilizada por el robot es ***differential drive*** y se explicó brevemente en qué consiste en el capítulo anterior.

Para el análisis que se sigue, referirse a la ***Figura 5***. El análisis se basa en [4].

El desarrollo se basa en considerar que todo movimiento puede definirse como un desplazamiento del robot sobre un círculo, cuyo centro se denomina ***ICC*** (***Instantaneous Center of Curvature***). Los valores que se deberán proveer a la función de cinemática serán entonces el ***radio de curvatura*** y ***la velocidad angular con la que se desplazará sobre ese radio***.

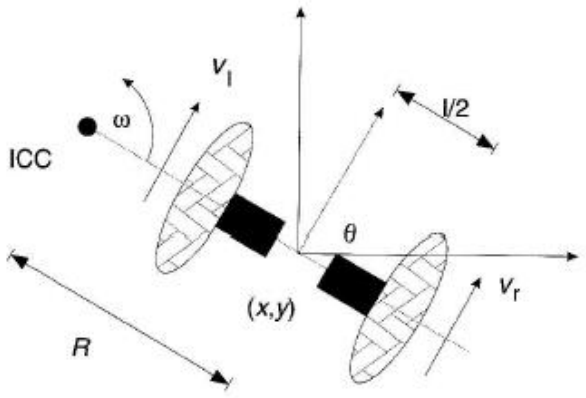

Fig. 5. Esquema de control diferencial

Variando las velocidades V_r y V_l se pueden obtener las trayectorias requeridas. Como la velocidad de rotación ω debe ser la misma para ambas ruedas, se pueden definir las siguientes ecuaciones:

$$V_r = \omega \left(R + \frac{l}{2}\right) \qquad (1)$$

$$V_l = \omega \left(R - \frac{l}{2}\right) \qquad (2)$$

Donde l es la distancia entre los centros de las dos ruedas, V_l y V_r son las velocidades tangenciales sobre el piso y R es la distancia ***con signo*** desde el ***ICC al punto medio entre las ruedas.***

Se analizarán algunos casos críticos:

- Si $V_l = V_r$, entonces el robot se moverá en línea recta, ya que R se volverá infinito y no habrá rotación.
- Si $V_l = -V_r$, entonces $R = 0$ y el robot rotará sobre el eje central del robot (rotación sobre el lugar).
- Si $V_l = 0$, habrá rotación del robot sobre el eje izquierdo de la rueda. En este caso $R = \frac{1}{2}$. Lo mismo aplica para el otro motor.

Un dato muy importante a tener en cuenta es que, ***como los motores se encuentran ubicados en antiparalelo, las velocidades positivas en el motor derecho implican que el robot avanza, pero implican lo opuesto en el motor izquierdo.***

B. Obtención de valores de motor

Se requieren los valores característicos para el correcto modelado del motor utilizado. Se utilizará dos ***JGA25-371*** que cuentan con una caja reductora con una velocidad final de 216rpm y encoders incorporados para realizar la medición de velocidad. Para más información referirse al siguiente link.

Fig. 6. Motor JGA25-371

Lamentablemente, no fue posible conseguir los datos requeridos, por lo tanto, los valores fueron estimados de la siguiente forma:

- Constante de torque y la Constante de fuerza electromotriz:
  Se divide el torque del motor en un determinado punto de su curva característica por sobre la corriente que circula en ese momento. Ambos valores siempre coinciden:

$$K_t = K_e = \frac{T}{I} = \frac{0.0834\ N.m}{0.3A} = 0.278\frac{N.m}{A} \quad (3)$$

- Constante de fricción del motor:
  La fricción del motor se puede estimar midiendo el torque del motor y su velocidad angular para cuando no hay carga. Se sabe además que el torque y la corriente se relacionan mediante la constante K_e. Se cumplen así las siguientes ecuaciones:

$$T_0 = K_e * I_0 \quad (4)$$

$$T_0 = b * \omega_0 \quad (5)$$

$$b = \frac{K_e * I_0}{\omega_0} = \frac{0.278\frac{N.m}{A} * 0.046A}{\frac{126}{60}} = 0.006\ N.M.s \quad (6)$$

- Para la obtención del momento de inercia se calcula a partir de la masa y el radio del rotor y analizándolo como un cilindro:

$$J = \frac{1}{2}m * r^2 = \frac{1}{2} * 0.057Kg * 0.0172^2 = 8{,}43x10^{-6}\ \text{Kg. m}^2 \quad (7)$$

Los valores restantes se midieron con multímetro UNI-T70A. Se muestra en la siguiente tabla el resumen de los valores obtenidos.

TABLA I
VALORES PARA CÁLCULO DE PLANTA DE MOTOR

Símbolo	Descripción	Unidades
J	Momento de inercia del rotor	8,43e-6 Kg.m²
b	Constante de fricción del motor	0.006 N.m.s
Ke	Constante de fuerza electromotriz	0.278 V/rad/sec
Kt	Constante de torque	0.278 N.m/A
R	Resistencia de la bobina	14 ohm
L	Inductancia de la bobina	8 mH

C. Ecuaciones de planta de motor

En la ***Figura 7*** podemos observar el circuito equivalente de un motor DC típico.

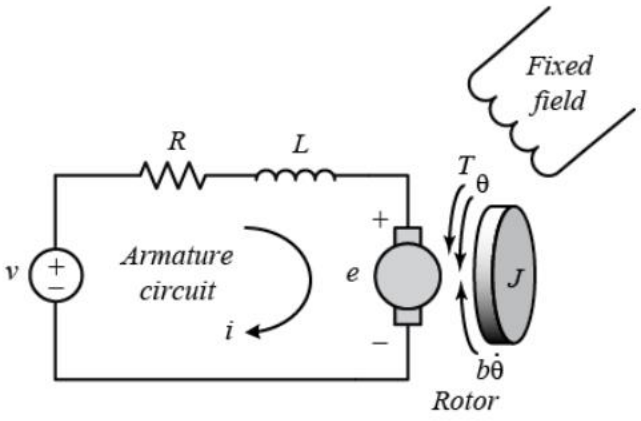

Fig. 7. Circuito equivalente de un motor DC.

El torque de un motor DC es proporcional a la corriente de armadura y la intensidad del campo magnético. Se supondrá que la intensidad del campo es constante y que, el torque dependerá únicamente de la corriente de armadura ***i*** multiplicada por el factor K_t. Esto se denomina ***motor controlado por armadura***:

$$T = K_t * i \qquad (8)$$

La contra fuerza electromotriz ***e*** es proporcional a la velocidad angular del eje multiplicada por $\boldsymbol{k_e}$:

$$e = K_e * \dot{\theta} \qquad (9)$$

Tenemos que:

$$K_t = K_e \qquad (10)$$

Por lo tanto, se utilizará simplemente ***k*** para representar ambos valores.
Si se observa nuevamente la figura 2 y, siguiendo la segunda ley de Newton y la ley de voltajes Kirchoff se deduce:

$$T = b * \omega \qquad (11)$$

$$J\ddot{\theta} + b\dot{\theta} = K_i \qquad (12)$$

$$L\frac{di}{dt} + Ri = V - K\dot{\theta} \qquad (13)$$

D. Análisis de lazo abierto

Con la ecuación de planta y, teniendo los valores característicos del motor, estamos en condiciones de analizar la función de transferencia. Aplicando la transformada de Laplace, las ecuaciones (12) y (13) pueden expresarse en términos de la variable ***s*:**

$$s(Js + b)\Theta(s) = KI(s) \qquad (14)$$

$$(Ls + R)I(s) = V(s) - Ks\Theta(s) \qquad (15)$$

Para lograr la función de transferencia de lazo abierto se elimina $I(s)$ de entre las ecuaciones (14) y (15), donde la velocidad de rotación se considera la salida y el voltaje de armadura es considerado como entrada.

$$P(s) = \frac{\dot{\Theta}(s)}{V(s)} = \frac{K}{(Js+b)(Ls+R)+K^2}\left[\frac{rad/sec}{V}\right] \qquad (16)$$

Es así como se puede analizar el comportamiento del sistema en lazo abierto. Para ello, se utiliza la herramienta de cálculo matemático Octave. Para los detalles sobre como se realizaron cada uno de los cálculos, referirse al archivo ***PID.m***.
La respuesta al escalón unitario para lazo abierto, así como también el diagrama de polos se observa en la ***Figura 8***.

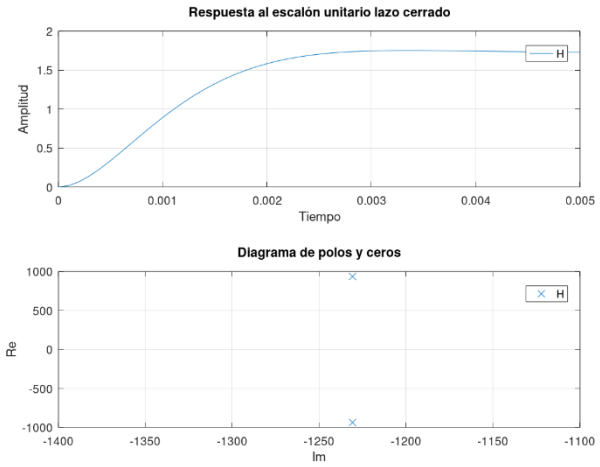

Fig. 8. Respuesta al escalón unitario y diagrama de polos y ceros para lazo abierto

Como se observa aquí, contamos con 2 polos con valores reales negativos y, por lo tanto, el sistema es ***estable***. Como los polos son conjugados se puede deducir que la respuesta en estos extremos será una onda senoidal que decae en el tiempo. El ancho de banda obtenido es ***2 rad/s***.

E. Análisis de lazo cerrado campo contínuo

Continuando con el análisis, se procede a analizar el sistema, pero en este caso en lazo cerrado. Para lograr dicho objetivo se realiza una convolución en el numerador y denominador de la función de transferencia, aplicando como coeficientes los valores estimados de ***PID***. Luego de esto se resuelve la función como un sistema cerrado con realimentación unitaria.

Los valores mostrados en la ***Figura 9*** corresponden a las respuestas de escalón unitario y diagrama de polos y ceros en lazo cerrado con los siguientes coeficientes:

- Kp = 100
- Ki = 200
- Kd = 10

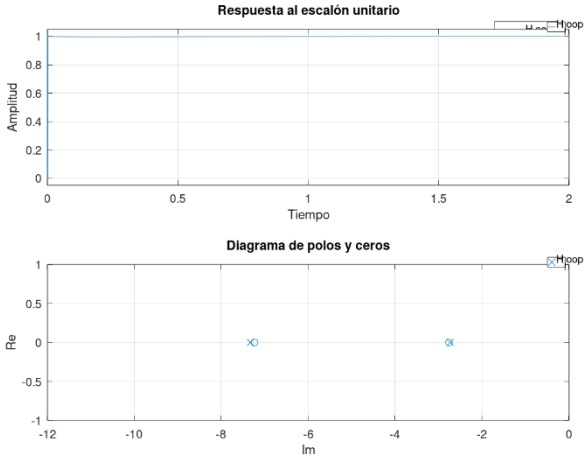

Fig. 10. Respuesta al escalón unitario lazo cerrado.

Se observa en este caso ***3 polos*** y ***2 ceros***. Todos ellos se encuentran en el semieje negativo de los imaginarios, por lo tanto, el sistema es ***estable***. Como se encuentran sobre el eje imaginario, el ***decaimiento de la función será exponencial***.

Se muestra además diagrama de bode para este caso en la ***Figura 10.*** El ancho de banda obtenido es ***3 rad/s***.

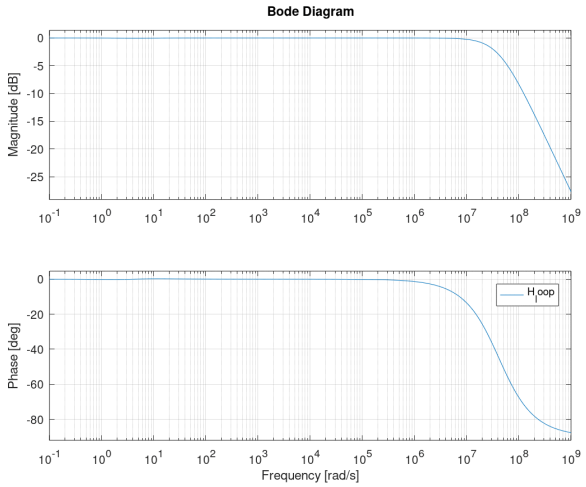

Fig. 9. Diagrama de bode para lazo cerrado

F. Análisis de lazo cerrado campo discreto

Como la implementación será sobre un microcontrolador, es preciso determinar el comportamiento del sistema para un sistema discreto. El análisis se hace para un muestreo de 25Hz. Se observa en la ***Figura 11*** Se observa la respuesta al escalón unitario es de ***40ms*** como era de esperarse.

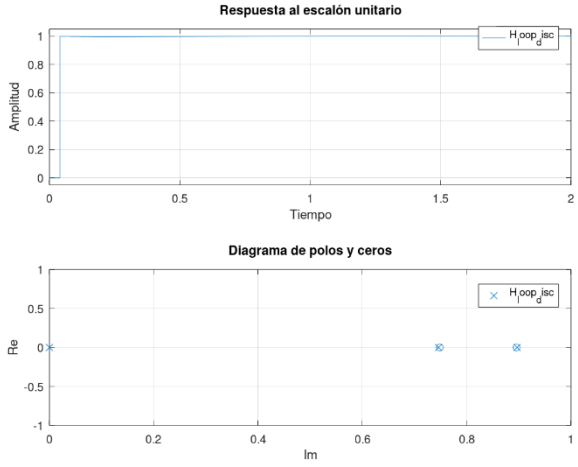

Fig. 11. Respuesta al escalón unitario lazo cerrado en tiempo discreto.

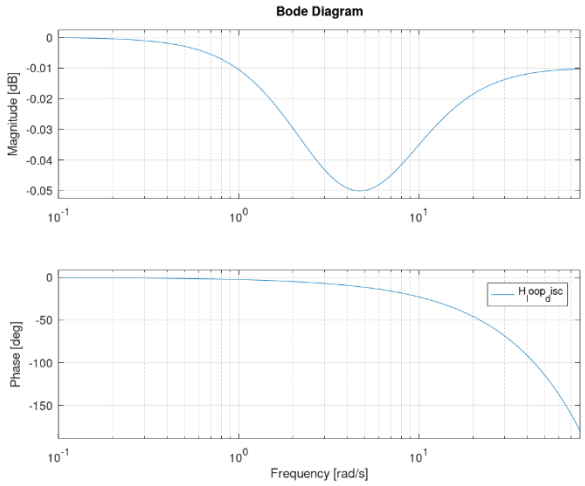

Fig. 12. Diagrama de bode para lazo cerrado en tiempo discreto.

Encontramos también ***3 polos*** y ***2 ceros*** coincidente con el caso de espectro continuo. Todos ellos se encuentran dentro de la circunferencia de radio unitario y, por lo tanto, el sistema es estable.

V. METODOLOGÍA DE TRABAJO

Para llevar adelante este trabajo, se tomarán las siguientes decisiones en cuanto a gestión:

- El proyecto se llevará adelante utilizando software de versionado git. Se utilizará la plataforma ***Github***.
- Se priorizará el uso de software libre por sobre el software privativo.
- Toda la información se mantendrá privada hasta que se alcance un nivel de funcionalidad satisfactorio. Pasado este punto, el software será de libre acceso.
- El software ROS se empaquetará utilizando ***Docker***.
- Toda la información relacionada a hardware, diseño mecánico y documentación se alojará en: https://github.com/GonzaCerv/noah-hardware .
- La información relacionada al software que correrá bajo ROS se almacenará en: https://github.com/glpuga/ros-robot-software
- Toda la información relacionada a los contenedores de Docker se encontrará en: https://github.com/GonzaCerv/noah-docker

VI. DISEÑO

A. *Mecánica*

El completo diseño de la mecánica se hizo sobre ***Solidworks 2020***. Como se mencionó anteriormente, el diseño se basó en modelos preexistentes. Se requirieron numerosas iteraciones para llegar al modelo actual que se puede apreciar en la ***Figura 13.*** Está formado por ***3 estructuras*** ranuradas cortadas en ***MDF de 3mm*** (piezas con textura símil madera). El resto del robot está formado por ***9 piezas*** que se fabricaron con ***impresora 3D***. Se utilizó plástico de tipo PLA para la estructura. Las ruedas son de 82mm y están formadas por un plástico duro en su llanta y una capa impresa en Flex para la cubierta. Esto le otorga una mejor adherencia que si se implementaba de otra forma. Para lograr el punto de apoyo con bajo rozamiento, se utilizó una pelota de ***ping pong de 40mm dura*** (ver ***Figura 14***). La pelota reposa sobre unas esferas de acero de 6mm. Este simple pero muy efectivo diseño se basa en

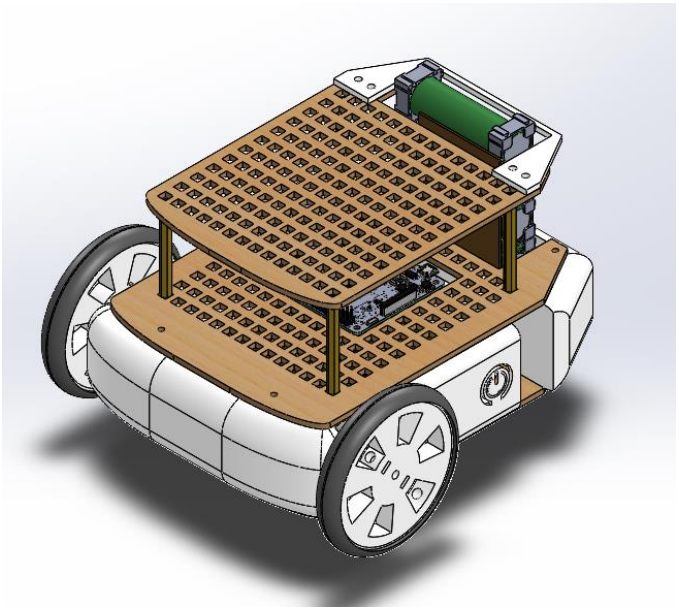

Fig. 13. Vista en perspectiva del diseño final

Jetbot developper kit [5]. Para vincular los distintos estantes de MDF, se utiliza un ***separador hexagonal de 40mm de alto***. Todas las uniones se

hacen con ***tornillos de 3mm***. Cuenta con un ***botón de encendido impreso*** (cuenta con receptáculo para

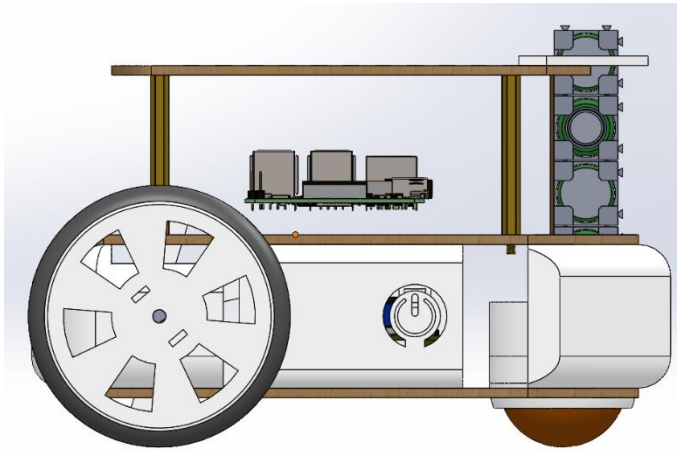

Fig. 14. Vista lateral del robot

el botón y led) y un puerto de carga de tipo ***barrel-jack***.

Las dimensiones finales del modelo son ***200x182x130mm***. Se comprobó que las dimensiones son suficientes para alojar una ***Raspberry PI3*** o una ***Jetson Nano***, como era requisito.

B. Electrónica

La ***Figura 15*** muestra con detalle el esquema eléctrico del PCB. Los esquemas fueron realizados con ***KiCad***.

Con respecto a la fuente de alimentación, se utilizaron 2 integrados. Uno de ellos es un LM317, un regulador lineal muy conocido en el mundo de la electrónica. Su salida es regulable mediante un simple divisor resistivo, es económico y requiere poca cantidad de componentes adicionales para ponerlo en funcionamiento. El circuito fue tomado de la ***Figura 21 de su datasheet*** [6]. Este simple diseño tiene la particularidad de que, para poder además de regular la tensión, utiliza un resistor formado por el paralelo de ***R4*** y ***R6*** para regular la corriente. En condiciones normales el regulador entrega a su salida ***16.8v***, que es la tensión requerida para cargar ***4 celdas de ion-litio***. Esta tensión se ajusta mediante ***R1*** y el ***potenciómetro RV1***. Cuando el consumo supera aproximadamente ***1 Amper*** de consumo, la caída de tensión sobre las resistencias mencionadas es suficiente para polarizar ***Q1***, reducir la tensión de realimentación del ***LM317*** y así reducir la tensión de salida. De esta forma, se logra un control Tensión constante – Corriente constante o ***CV-CC*** que asegura cargar la batería de un modo seguro, incrementando significativamente los ciclos de uso, reduciendo el exceso de temperatura en la batería al momento de la carga y reduciendo el riesgo de explosión de la batería. La curva de carga característica se puede observar en la ***Figura 15***.

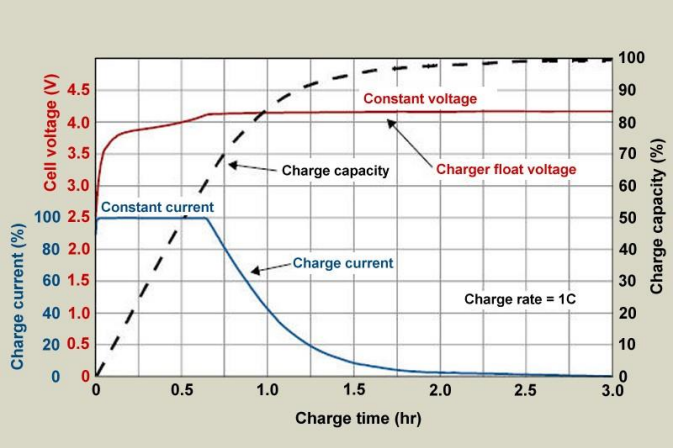

Fig. 15. Ciclo de carga de una batería Ion-litio

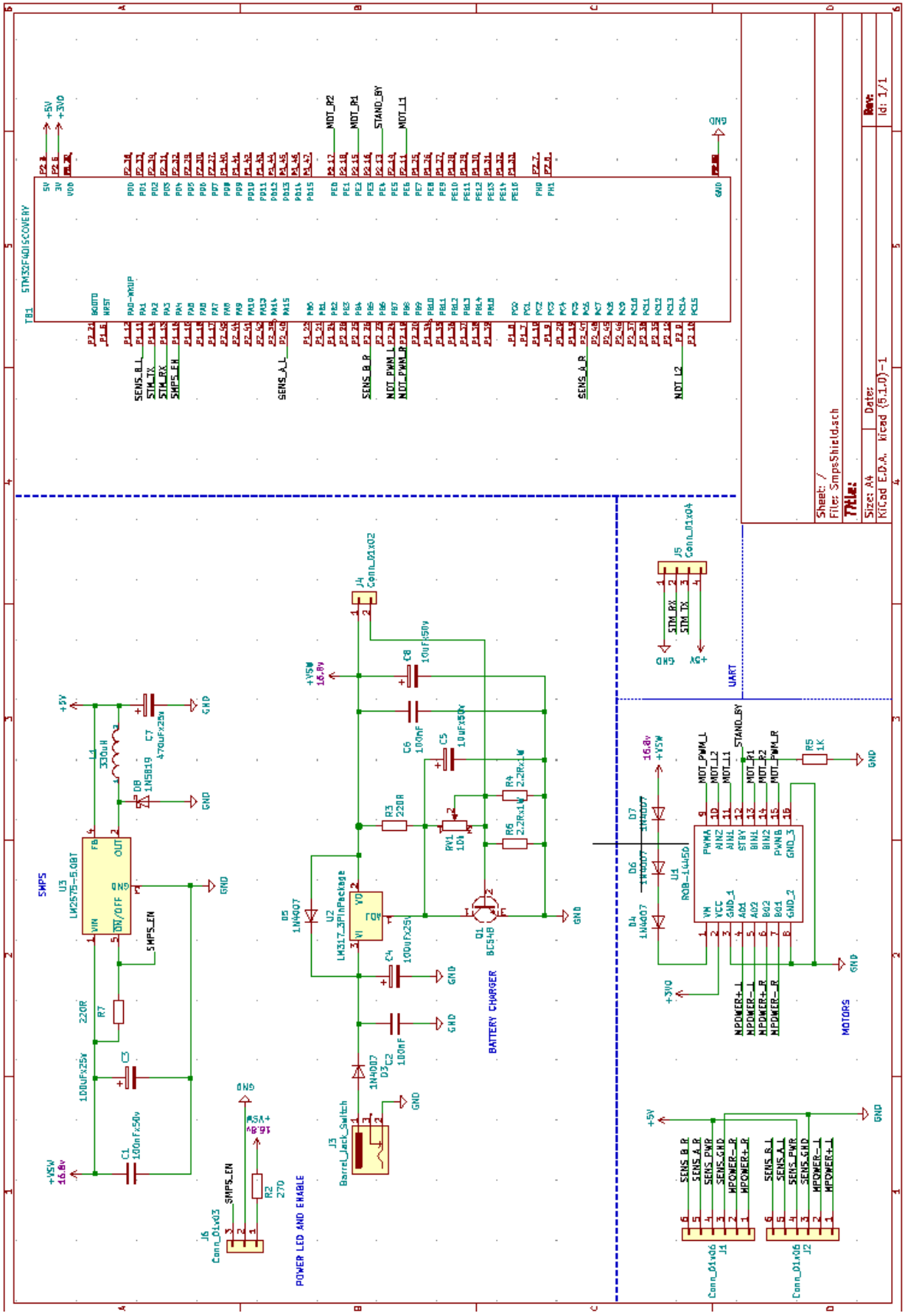

Fig. 16. Esquema PCB. Capa 2

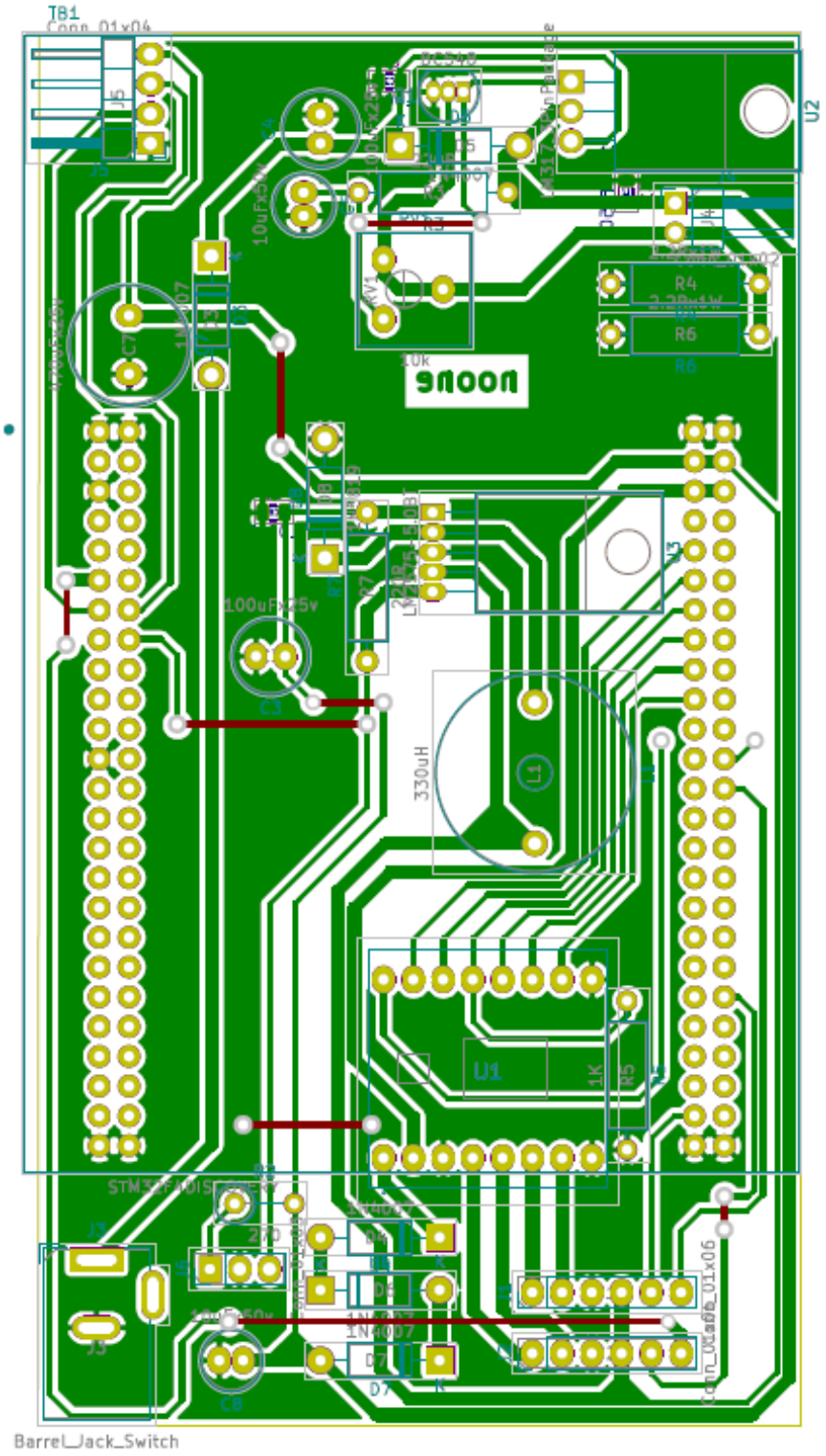

Fig. 17. Diseño PCB

Como se detalla claramente en [7], para una correcta carga de las baterías, se debe cargar con una ***corriente constante*** para luego continuar con un ciclo de ***tensión constante***. Esto se pudo lograr gracias al circuito explicado. Para Alimentar toda la lógica, se utiliza una fuente switching formada por un ***LM2575*** en configuración ***step-down***. Se alimenta con la tensión de salida de la batería y se reduce a ***5v 1A***. El encendido de esta fuente se hace simplemente puenteando los pines ***3*** y ***2*** de ***J5***. Se puede conectar un led al ***pin 1*** restante para utilizarlo como testigo de encendido. Luego de haber presionado el botón, el microcontrolador enciende y se encarga de mantener aplicada la tensión baja para continuar con el funcionamiento. Es así como el microcontrolador tiene la capacidad de apagar la fuente y así también el sistema. La fuente alimenta además una Raspberry Pi sobre la cual trabajará la capa 4.

Los motores son controlados por un módulo doble puente H. El modelo es ***Tb6612***. Permite manejar motores hasta ***15v – 1.2A***. Posee una baja caída de tensión de entrada entre la alimentación y los motores, esto es útil para reducir la disipación de calor. Se prefirió utilizarlo como un módulo separado, ya que facilita su reemplazo y, al manejar potencia, es propenso a fallas.

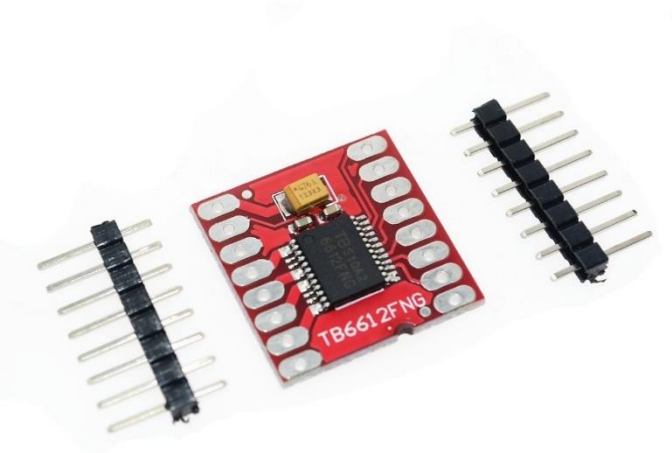

Fig. 18. Doble puente H. Tb6612

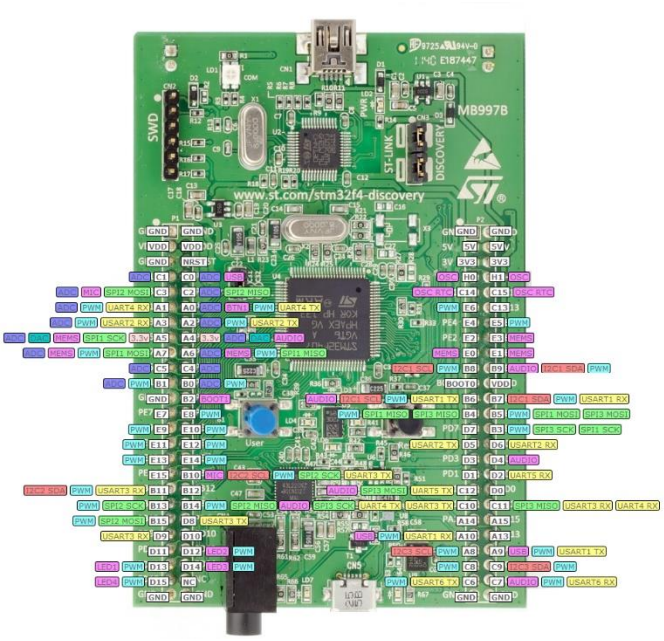

Fig. 19. STM32F4 Discovery con su correspondiente pinout.

Sobre la placa fabricada se monta una STM32F4Discovery. Es una placa de desarrollo de un costo asequible, conocida y con mucho soporte por parte de la comunidad. Posee numerosos sensores que serán oportunamente utilizados en el futuro. Cuenta con un microcontrolador ***STM32F407VGT6***. Este chip es un ARM Cortex M4 con núcleo ***DSP***, el cual tiene un encapsulado de 100 pines, alcanza una velocidad de hasta ***168Mhz***, tiene una memoria ***RAM de 192Kb*** y una ***FLASH de 1Mb***. Esta placa de desarrollo se alimenta también de la fuente de alimentación de 5v, posee un regulador ***LDO*** que reduce la tensión a 3.3v.

C. Firmware

El microcontrolador utilizado posee la potencia de procesamiento suficiente como para administrar el manejo completo de la mecánica del robot. Se utilizará ***FreeRTOS,*** ya que es un sistema operativo de tiempo real que es soportado por la marca del chip. Gracias a la utilización de esta API, será posible ejecutar numerosas tareas al mismo tiempo, asegurando que los tiempos de ejecución sean los requeridos. Todo el código con el que se trabajará será en ***ANSI C***, generado con la herramienta que

provee STMicroelectronics: ***CubeMX***. Esta herramienta provee una forma gráfica de configurar los periféricos de una manera gráfica.

Luego el código se programa utilizando ***Visual Studio Code.*** El código se compila mediante ***Makefile*** y luego es grabado en el microcontrolador.

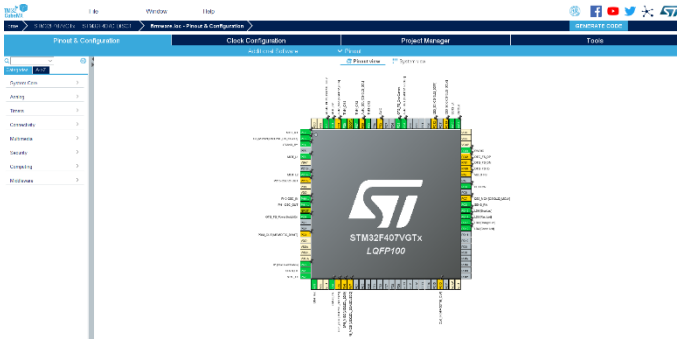

Fig. 20. Interfaz de CubeMx

El código será manejado por 4 hilos o ***Threads***. Una de ellas, llamada ***StartDefaultTask*** se encarga de inicializar los periféricos, mantener encendida la fuente de alimentación y comprobar cuando se presiona el pulsador nuevamente para apagar el robot. ***tsk_pid_l*** y ***tsk_pid_l*** se encargan de recibir realizar todo el cálculo del PID y así asignar los valores correspondientes de PWM a cada uno de los motores. Por último, está ***tsk_usart*** que se encarga de procesar toda la información que ingresa al microcontrolador por medio del puerto ***USART***.

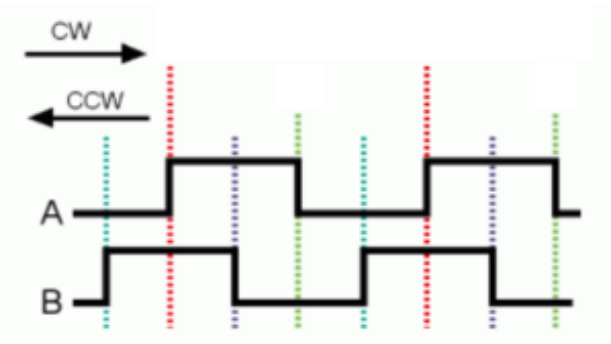

Fig. 21. Señales en cuadratura de encoder

Para explicar el funcionamiento del firmware, se comenzará por la medición de los datos del encoder. Esta tarea se realiza en ***tsk_pid_l*** y ***tsk_pid_l***. Cada uno de los encoders es de tipo incremental. Tienen 2 salidas que entregan señales cuadradas y desfasadas 90 grados, como se ve en la ***Figura 21***. Los pasos del encoder se cuentan en cada cambio de flanco de alguna de las señales. Si la señal A se adelanta 90 grados con respecto a B, el sentido de giro es positivo. En caso contrario, el sentido de giro es negativo. De esta forma se tiene ***posición y sentido de giro del motor***. Para poder procesar estas señales de manera rápida, se utilizó un timer por cada motor. Algunos modelos de timers permiten trabajar en ***modo encoder.*** Lo que hace este módulo es comprobar los cambios de cuadratura del sensor de posición y detectar el desfasaje. Si el desfasaje es positivo, incrementa en 1 unidad la cuenta del timer y si ocurre lo contrario, decrementa. Cuenta además con un modo filtro de entrada integrado, que previene de contar pulsos generados por ruido.
La cuenta de pulsos permite obtener una aproximación de la distancia recorrida. Mediante varias pruebas, se concluyó que cada vuelta al final de la caja reductora equivale a ***918 pulsos***. El radio de la rueda del robot es de ***41mm***. Con estos datos, podemos utilizar la siguiente fórmula para calcular el desplazamiento del robot:

$$Distancia = \frac{ticks}{918} * 2\pi * R \qquad (17)$$

Esta información aún no es suficiente, se necesita calcular la velocidad de cada rueda. Para lograr esto, se almacenan valores de cada encoder a intervalos constantes. La velocidad es entonces:

$$V = \frac{ticks_0 - ticks_{-1}}{918} * \frac{1}{T} * 2\pi * R \qquad (18)$$

Es crítico lograr que las mediciones entre intervalos sean constantes, ya que de lo contrario se observarán variaciones entre las distintas velocidades instantáneas. Es por eso por lo que se utilizó otro timer que realice interrupciones periódicas y sea el disparador de la toma de ticks de cada encoder. El diagrama de la ***Figura 22*** ilustra el funcionamiento de las tareas relacionadas a medición de datos de encoder y actualización de PID.

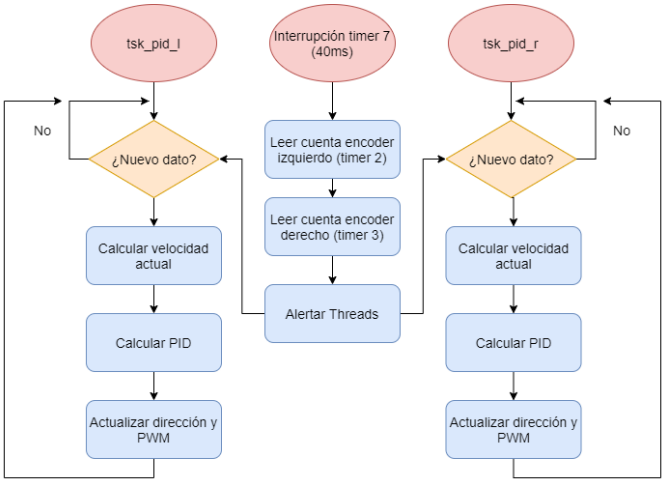

Fig. 22. Diagrama simplificado de tsk_pid_l y tsk_pid_r

Por último, se explicará el protocolo de comunicación entre el firmware y la microcomputadora. Como se dijo anteriormente, se utilizará protocolo UART debido a su simplicidad. La configuración es ***115200 baudios, 1 bit de parada y sin control de flujo***. Se implementó un protocolo simplificado que permite extraer y enviar información muy básica al microcontrolador.

La comunicación se basa en envío de paquetes. Cada paquete se conforma por un byte de inicio y de parada, cuyos valores corresponden a ***0xA0*** y ***0xB0*** respectivamente. Seguido de byte de inicio, se incluye un byte para identificar el comando a ejecutar. El nibble alto de comando indica cuantos argumentos se enviarán (cuantos bytes serán enviados en esa trama) y el nibble bajo a qué comando corresponde. Seguidamente, se envía la cantidad de argumentos requeridos y, por último, un byte de parada. La ***Figura 23*** ilustra lo explicado en este párrafo.

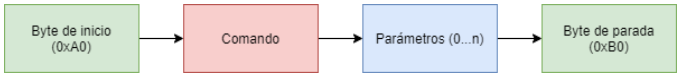

Fig. 23. Trama para comunicación UART

La lista de comandos actualmente soportados se ilustra en la siguiente tabla.

TABLA 2
COMANDOS VÁLIDOS UART

Comando	Descripción	Argumentos
0x00	Sirve para testeo de comunicación. Devuelve un paquete igual.	-
0x02	Apaga el robot	-
0x20	Cantidad de pasos del encoder izquierdo	int_16t: cuenta de pasos
0x21	Cantidad de pasos encoder derecho	int_16t: cuenta de pasos
0x40	Asigna velocidad a motor izquierdo	Float: velocidad en m/s
0x41	Asigna velocidad a motor derecho	Float: velocidad en m/s

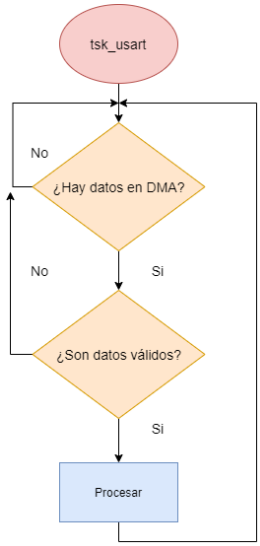

Fig. 24. Esquema para thread tsk_uart

Finalmente, se muestra el esquema de funcionamiento del UART en la ***Figura 24***.

D. Software

Llegamos finalmente al software. Se eligió utilizar una Raspberry PI3 debido a su bajo consumo, simplicidad y una excelente comunidad que brinda mucha información sobre dicho dispositivo. A su vez, cumple con los requisitos de RAM y Memoria de disco para correr perfectamente ROS. Se puede observar en las siguientes figuras la forma física de la placa, así como también la asignación de pines de su bornera.

Fig. 25. Raspberry PI3

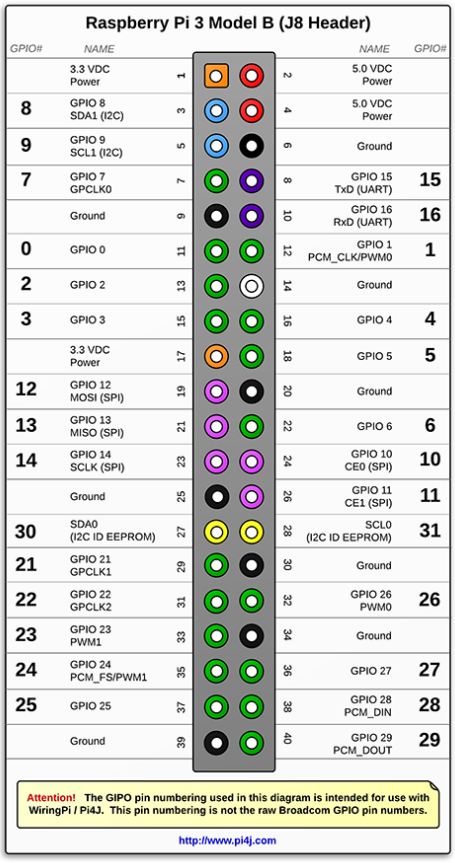

Fig. 26. Pinout

Para poder teleoperar este robot, la Raspberry se conectará a una red WiFi. Esta conexión nos permitirá acceder a su terminal de Linux utilizando SSH [8]. Una vez dentro del terminal, podremos acceder a ROS. Dicho software cuenta con paquetes especializados en manejo de robots utilizando ***differential drive*** [9]. ROS permite conectar dichos paquetes de la manera que se desee para lograr la funcionalidad deseada. El trabajo de desarrollo se centró en crear el proyecto y diseñar un paquete que sirva como interfaz de comunicación entre el ***firmware*** y el ***software***.

El paquete antes mencionado se comunicará con otro denominado ***diff_tf*** que se encarga de convertir la cuenta de pasos de cada encoder en valores de odometría. Estos valores consisten en:

- Datos de posición: X, Y y rotación sobre Z
- Datos de velocidad: velocidad lineal, angular y datos de covarianza.

Estos datos serán utilizados en el futuro para realizar navegación en el robot, tal como lo hace un robot comercial profesional.

Se debe proveer alguna forma de teleoperar el robot. Es aquí donde entra el paquete ***teleop_twist_keyboard***. Permite recibir datos desde el teclado y enviarlos al robot para su ejecución. De esta manera, mediante una consola y un teclado, es posible controlar los movimientos del robot. La interfaz mostrada en pantalla se muestra en la siguiente figura [10].

```
Reading from the keyboard  and Publishing to Twist!
---------------------------
Moving around:
   u    i    o
   j    k    l
   m    ,    .

q/z : increase/decrease max speeds by 10%
w/x : increase/decrease only linear speed by 10%
e/c : increase/decrease only angular speed by 10%
anything else : stop

CTRL-C to quit
```

Fig. 27. Interfaz de consola para controlar el robot

La información que proviene de ***teleop_twist_keyboard*** es una velocidad final y una dirección en la que el robot se debe mover, pero esta información debe convertirse en velocidades individuales para cada motor. Para ello se utilizó paquete ***twist_to_motors***. Este paquete recibe una

información dirección y velocidad (twist) y la convierte en velocidades individuales utilizando las ecuaciones (***1)*** y ***(2)***.

VII. Resultados

En las siguientes imágenes se observa el robot terminado.

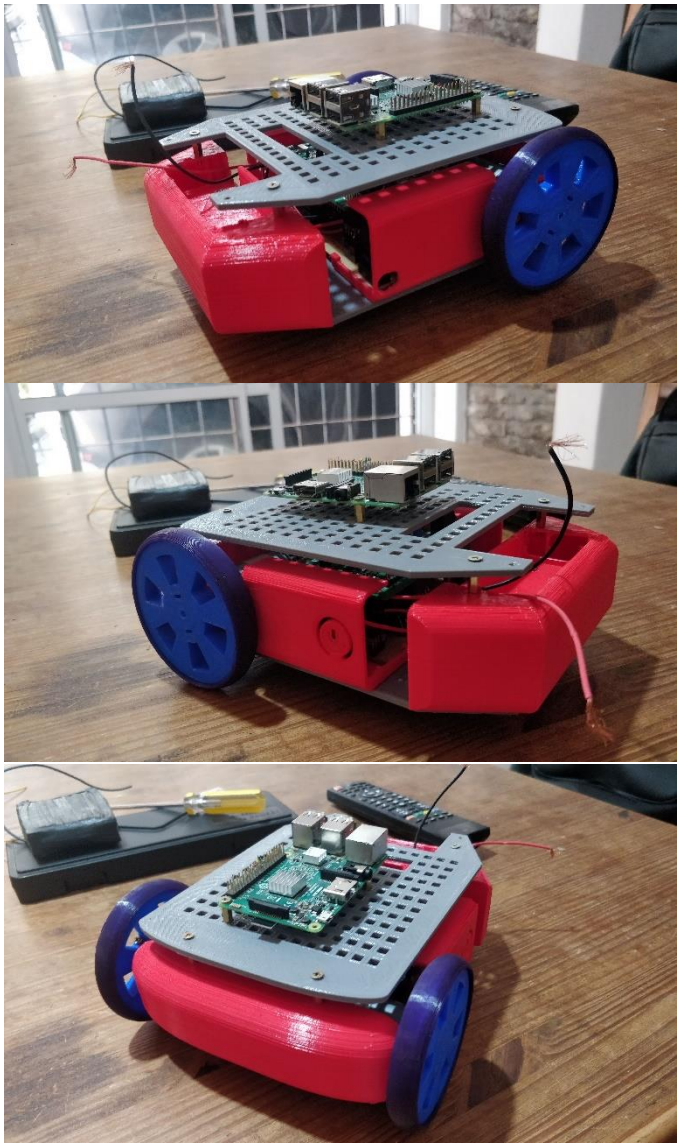

Fig. 28. Robot terminado

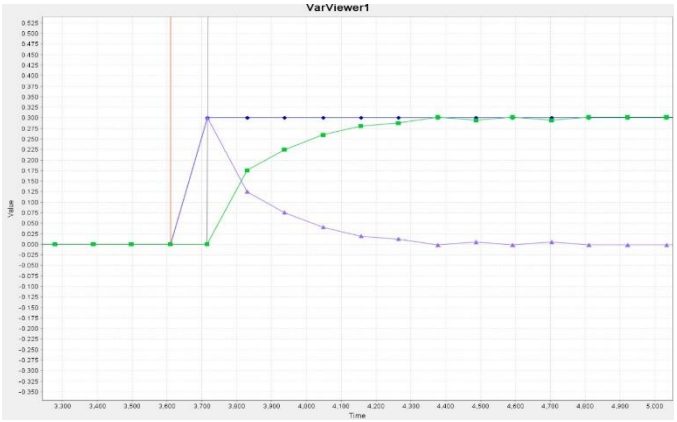

Fig. 29. Prueba PID

La ***Figura 30*** muestra el PID de los motores bajo prueba. La línea violeta es el setpoint y la línea verde es la curva de la velocidad actual. Los valores corresponden a un Kp: 150, Ki: 2000 y un Kd: 1.0

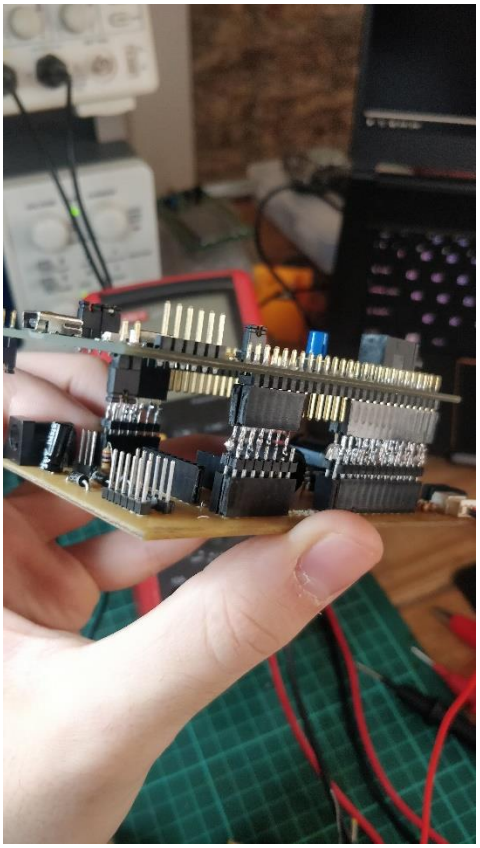

Fig. 30. PCB Terminado y conectado a placa de desarrollo

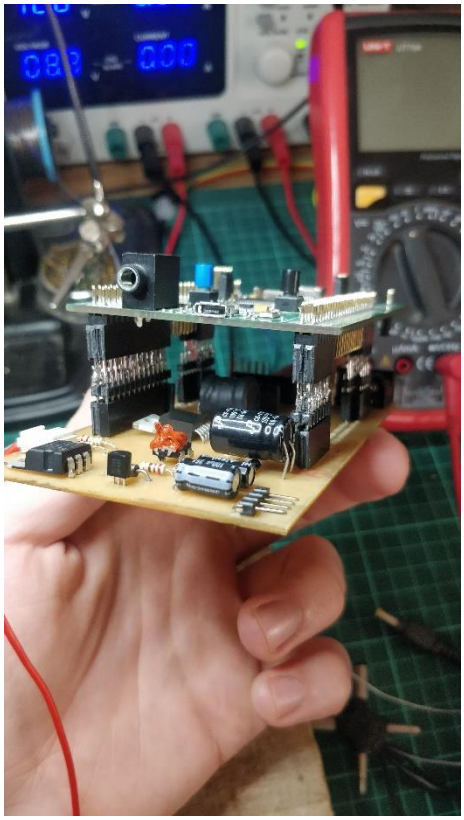

Fig. 31. PCB Terminado

VIII. CONCLUSIÓN

Como conclusión se puede decir que fue un proyecto muy interesante, me permitió adquirir una gran cantidad de conocimientos en distintos temas, tanto mecánica como electrónica y desarrollo de software. Cumplió de manera excelente como proyecto integrador, ya que cada una de las partes que lo componen está íntimamente relacionadas entre sí.

Me permitió mejorar significativamente mis habilidades de gestión, ya que es un proyecto muy demandante que tuve que gestionar mientras desempeñaba mis actividades laborales. Se suma a esto las dificultades por la actual pandemia que dificultó en gran medida la adquisición de componentes y equipamiento para realizar mediciones. El robot debió adaptarse en función de los recursos presentes.

Un valor agregado que quisiera destacar es que permitió diagramar los primeros pasos de mi tesis final.

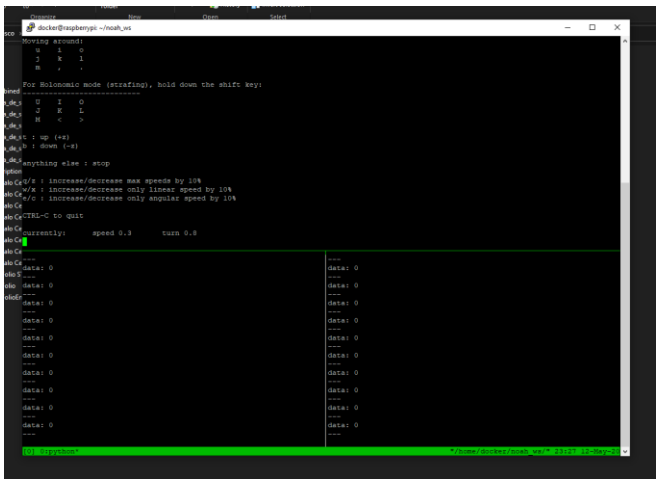

Fig. 32. Consola de control

IX. BIBLIOGRAFÍA

[1] «Robotis,» Dinamixel, [En línea]. Available: http://emanual.robotis.com/docs/en/platform/turtlebot3/overview/. [Último acceso: 10 5 2020].

[2] Robotgroup, «Robotgroup,» [En línea]. Available: https://www.robotgroup.com.ar/. [Último acceso: 10 5 2020].

[3] O. Robotics, «ROS,» Open Robotics, [En línea]. Available: https://www.ros.org/. [Último acceso: 5 10 2020].

[4] D. a. Jenkin, «Computational Principles of Mobile Robotics.,» *CS W4733 NOTES.*

[5] «Jetbot Developper kit,» [En línea]. Available: https://github.com/NVIDIA-AI-IOT/jetbot. [Último acceso: 11 5 2020].

[6] T. instruments, «Datasheet,» Texas instruments, 9 1997. [En línea]. Available: http://www.ti.com/lit/ds/slvs044y/slvs044y.pdf?&ts=1589200641129. [Último acceso: 11 5 2020].

[7] «BatteryUniversity,» 24 04 2018. [En línea]. Available: https://batteryuniversity.com/learn/article/charging_lithium_ion_batteries. [Último acceso: 5 11 2020].

[8] «Secure Shell,» Wikipedia, [En línea]. Available: https://es.wikipedia.org/wiki/Secure_Shell. [Último acceso: 12 5 2020].

[9] J. Stephan, «differential_drive,» Open Robotics, [En línea]. Available: http://wiki.ros.org/differential_drive. [Último acceso: 12 5 2020].

[10] A. Hendrix, «ROS.org,» Open Robotics, [En línea]. Available: http://wiki.ros.org/teleop_twist_keyboard. [Último acceso: 12 5 2020].

[11] D. COLLINS, «Motion Control TIPS,» 5 4 2017. [En línea]. Available: https://www.motioncontroltips.com/faq-difference-between-torque-back-emf-motor-constant/. [Último acceso: 13 5 2020].

[12] «Control tutorials for Matlab & Simulink,» Matlab, [En línea]. Available: http://ctms.engin.umich.edu/CTMS/index.php?example=MotorSpeed&section=ControlDigital. [Último acceso: 13 5 2020].

[13] «Control Tutorials for Matlab & Simulink,» [En línea]. Available: http://ctms.engin.umich.edu/CTMS/index.php?example=MotorSpeed&section=SystemModeling. [Último acceso: 13 5 2020].

[14] «Precision Microdrives,» [En línea]. Available: https://www.precisionmicrodrives.com/content/reading-the-motor-constants-from-typical-performance-characteristics/. [Último acceso: 13 5 2020].

[15] «StackExchange,» Robotics Beta, [En línea]. Available: https://robotics.stackexchange.com/questions/9935/how-to-find-friction-or-viscous-force-b-nmsec-in-dc-motor. [Último acceso: 13 5 2020].

[16] MASSACHUSETTS INSTITUTE OF TECHNOLOGY, [En línea]. Available: https://web.mit.edu/2.14/www/Handouts/PoleZero.pdf. [Último acceso: 5 13 2020].