



FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS

UNIVERSIDAD DE CHILE

GRUPO 6 - ROBOT OMNIDIRECCIONAL

EI2001-5 TALLER DE PROYECTO DE ROBÓTICA Y MECATRÓNICA

INFORME DE DOCUMENTACIÓN DEL PROYECTO DE ROBOT OMNIDIRECCIONAL PARA EL TALLER DE ROBÓTICA Y MECATRÓNICA

PROYECTO DE ROBOT OMNIDIRECCIONAL

Integrantes:	Nicolas Lagos L. Camilo Ramírez C. Joaquín Silva C.
Profesor:	Javier Ruiz del Solar
Auxiliar:	Isao Parra T.
Ayudantes de lab.:	Leonardo Garrido A. Cristopher Gómez Giovanni Pais L. Nicolas Marticorena Vidal
Fecha de entrega:	03 de septiembre de 2018
	Santiago, Chile

Índice de Contenidos

1. Introducción	1
2. Antecedentes generales	2
2.1. Ruedas omnidireccionales y mecanum	2
2.1.1. Ruedas omnidireccionales	2
2.1.2. Ruedas Mecanum	3
2.2. Motores CC con caja de engranajes	4
2.3. Encoder	4
2.4. Placa Arduino DUE	5
2.5. Puentes H	6
2.6. Controlador PID	7
2.7. Módulo Bluetooth HC-06	8
2.8. Programación de aplicaciones para Android	8
3. Descripción del proyecto	9
3.1. Sistema mecánico	9
3.1.1. Diseño y manufactura del Armazón de soporte de componentes	9
3.1.2. Diseño y manufactura de los rodillos	10
3.1.3. Diseño y manufactura de las ruedas A y B	11
3.1.4. Diseño y manufactura de los discos perforados	14
3.1.5. Ensamble de partes y rueda final	15
3.2. Sistema eléctrico	15
3.2.1. Subsistema de alimentación	16
3.2.2. Subsistema de control	16
3.2.3. Subsistema de comunicación	18
3.3. Software de control	19
3.3.1. Principios del movimiento	19
3.3.2. Funciones de movimiento	21
3.3.3. Control PID	22
3.3.4. Comunicación Bluetooth	26
3.3.5. Cohesión de las funciones implementadas en Arduino	29
3.3.6. Aplicación de Android de control	30
4. Resultados	33
5. Conclusiones	34
Referencias	35

Lista de Figuras

1. Ejemplos de ruedas omnidireccionales comerciales	2
2. Ejemplo de distribución geométrica útil para ruedas omnidireccionales	3

3.	Ejemplo de rueda Mecanum comercial	3
4.	Ejemplo de robot con ruedas Mecanum instaladas	4
5.	Mecanismos de un encoder óptico	5
6.	Modelos diferentes de Arduinos	6
7.	Puente H	6
8.	Módulo Bluetooth HC-06	8
9.	Modelo 3D del sistema mecánico	9
10.	Planos del Armazón	10
11.	Plano de los rodillos	11
12.	Proceso de manufactura de las ruedas (1)	12
13.	Proceso de manufactura de las ruedas (2)	12
14.	Proceso de manufactura de las ruedas (3)	13
15.	Proceso de manufactura de las ruedas (4)	13
16.	Proceso de manufactura de las ruedas (5)	14
17.	Planos de los discos perforados	14
18.	Rueda Final	15
19.	Esquema de conexión de un foto-interruptor	16
20.	Esquema de conexión de un motor CC	17
21.	Relación entre los pines del puente H del esquema anterior y el utilizado en el robot	18
22.	Esquema de conexión del módulo bluetooth	18
23.	Diagrama de composición de movimiento para una disposición de ruedas triangular	20
24.	Diagrama de funcionamiento de las ISR	23
25.	Diagrama de funcionamiento de la función <code>RecepciónSerial</code>	28
26.	Diagrama de funcionamiento del <code>loop()</code> de <code>Arduino</code>	30
27.	Capturas de pantalla de las ventanas de trabajo de la aplicación en Android	31
28.	Fotografías del robot terminado	33

Lista de Códigos

1.	Implementación de la función <code>MovimientoXYRotRobot</code>	21
2.	Implementación de la función <code>Debounced</code>	23
3.	Implementación de la función <code>RutinaEncoder</code>	23
4.	Implementación de la función <code>Avanzar</code>	24
5.	Implementación de la función <code>Retroceder</code>	25
6.	Implementación de la función <code>Frenar</code>	25
7.	Implementación de la función <code>Liberar</code>	25
8.	Implementación de la función <code>VerificarDetenciones</code>	26
9.	Implementación de la función <code>RecepcionSerial</code>	28
10.	Implementación de la función <code>ParseBuffer</code>	29
11.	Contrato de la función <code>EjecutarComandos</code>	29
12.	Implementación del <code>loop()</code> de <code>Arduino</code>	30

1. Introducción

El proyecto se motiva bajo la realización de algún proyecto de robótica y mecatrónica libre para la asignatura homónima en la Facultad de Ciencias Físicas y Matemáticas de la Universidad de Chile.

El propósito del proyecto es la construcción de un robot capaz de desplazarse de manera omnidireccional, es decir, que tenga la capacidad de desplazarse lateralmente, frontalmente, diagonalmente y que pueda rotar sobre si mismo.

Como objetivos secundarios, se tienen la implementación de un controlador PID, que teóricamente ajustará la precisión de los movimientos del robot, esto mediante la utilización de un encoder. Por otro lado, también se propone implementar un sistema de comunicación inalámbrico para controlar los movimientos del robot desde la distancia. En principio esto podría realizarse mediante comunicación infrarroja, como con un control remoto, o bluetooth, donde podría utilizarse un teléfono celular. Además, se propone la implementación de un mecanismo de seguridad basado en sensores de proximidad, que restrinja el movimiento del robot cuando esté cerca de chocar.

La realización de este proyecto puede ser útil para entender y experimentar con vehículos omnidireccionales, que en comparación a los vehículos comunes, presentan una movilidad mucho más intuitiva a la hora de controlar, no tanto así en vehículos como un automóvil, que requiere maniobras a veces complejas para, por ejemplo, estacionarse. El proceso de manufactura del robot también aportará experiencia con el tipo de maquinaria que vaya a utilizar, así como con los materiales que se vayan a escoger, en la implementación de circuitos eléctricos y en la programación de un controlador capaz de llevar a cabo las tareas requeridas de la forma apropiada.

2. Antecedentes generales

Previo a realizar una descripción del proyecto en si mismo, es necesario introducir ciertos componentes, principios y sistemas que son utilizados para la realización de este. Estos se encuentran presentes en las siguientes subsecciones.

2.1. Ruedas omnidireccionales y mecanum

Las ruedas omnidireccionales y mecanum están diseñadas para resolver la necesidad de que un aparato realice un movimiento omnidireccional.

El movimiento omnidireccional consiste en la capacidad para moverse en un plano, generalmente horizontal, con tres grados de libertad. Estos corresponden al movimiento lateral (en un eje x), frontal-trasero (en un el eje y , perpendicular a x) y rotacional, generalmente respecto a su centro de masas. La mayoría de vehículos es incapaz de realizar este movimiento. Por ejemplo, un automóvil no puede simplemente desplazarse de forma lateral.

Para la construcción de un aparato capaz de moverse de forma omnidireccional, es necesario plantear un diseño para las ruedas que sea compatible con esta funcionalidad. Los diseños de ruedas utilizados en su mayoría en proyectos de robótica y mecatrónica corresponden a las ruedas omnidireccionales y la ruedas mecanum.

2.1.1. Ruedas omnidireccionales

El diseño de las ruedas omnidireccionales consiste en la colocación de rodillos en la periferia de la rueda. Estos rodillos poseen libertad de rotación, y esta rotación es independiente de la rotación de la rueda, ya que se ubican de forma tal que rotan en un plano perpendicular al plano de rotación de la rueda, esto permite que efectivamente la rueda se pueda desplazar perpendicularmente a su plano de rotación.



Figura 1: Ejemplos de ruedas omnidireccionales comerciales

Sin embargo, una rueda por si sola no tiene la capacidad de desplazarse en la dirección perpendicular de forma autónoma. Por lo que generalmente esto se logra con la disposición geométrica de más ruedas, de forma tal que la composición de los movimientos provocados por las otras desplacen la rueda.



Figura 2: Ejemplo de distribución geométrica útil para ruedas omnidireccionales

2.1.2. Ruedas Mecanum

Similarmente al diseño omnidireccional, la rueda Mecanum posee rodillos ubicados en la periferia, pero estos rodillos no rotan perpendicularmente a la rotación de la rueda, sino que están colocados de forma oblicua, generalmente su plano de rotación está a 45° del plano de rotación de la rueda. Esta disposición de los rodillos provoca que una parte de la fuerza de la rueda sea aplicada en la dirección perpendicular al movimiento que tendría una rueda común.



Figura 3: Ejemplo de rueda Mecanum comercial

Esto permite que una rueda por si sola pueda desplazarse en una dirección oblicua. Con una disposición geométrica adecuada de las ruedas en el sistema, estos desplazamientos pueden componerse para lograr un movimiento completamente omnidireccional.



Figura 4: Ejemplo de robot con ruedas Mecanum instaladas

2.2. Motores CC con caja de engranajes

Un motor de corriente continua (motor CC o DC) es un tipo de motor caracterizado por convertir la energía eléctrica en energía mecánica, provocando un movimiento rotatorio por medio de un campo magnético. Usualmente son utilizados para generar movimientos constantes en sistemas que no requieran tanta precisión. Existen diversos tipos de motores, para este proyecto se trabajará con motores equipados con caja de engranajes, que son capaces de generar un torque superior.

Mediante la aplicación de distintos voltajes y cambios de polaridad, es posible controlar la dirección en la que rota el eje del motor y la rapidez angular con la que lo hace. El control de los motores CC suele llevarse a cabo mediante el uso de puentes H. Si es necesario un control más preciso de la velocidad angular del motor, se puede utilizar un dispositivo llamado encoder.

2.3. Encoder

Un encoder, o codificador, es un dispositivo que permite codificar el movimiento como una señal eléctrica, es decir, mide el movimiento del sistema y lo transforma en una señal utilizable por el controlador. Existen diversos tipos de encoder, entre los que se encuentran los lineales y los rotatorios, los ópticos, magnéticos, etc. Para efectos de este informe, es necesario hablar del encoder rotatorio óptico.

Un encoder rotatorio puede medir la velocidad angular de un giro. Es utilizado en sistemas que requieran precisión en este aspecto. Consiste, en su forma más básica, en un disco marcado que gira solidario a la rueda o motor y un sensor capaz de distinguir las marcas del disco. En el caso de este proyecto, las marcas son perforaciones, y el sensor es un foto-interruptor.

Un foto-interruptor es un sensor óptico compuesto de un emisor y un receptor de luz infrarroja. Si el receptor puede detectar la señal del emisor, es decir, si no hay un objeto opaco entre el emisor y el receptor, el foto-interruptor envía una señal al controlador. En caso contrario, envía otra señal. Así, el controlador puede calcular la diferencia de tiempo entre las interrupciones detectadas. Si las interrupciones están dadas por las perforaciones de un disco que gira solidario al sistema, entonces el controlador tiene acceso al tiempo que se demora el sistema en llegar de un estado a otro, y con

eso, se obtiene la velocidad angular.

Los encoder además de medir rapidez de giro, pueden medir sentido de giro y el ángulo de rotación con respecto a una referencia, esto se realiza haciendo marcas de distintos tamaños.



(a) Disco perforado para encoder óptico capaz de medir posición angular



(b) Fotointerruptor básico

Figura 5: Mecanismos de un encoder óptico

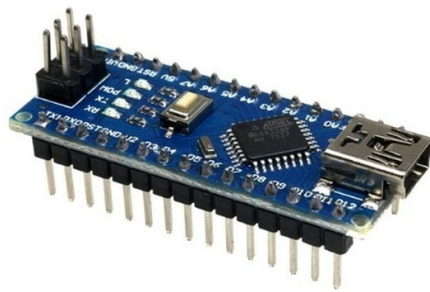
2.4. Placa Arduino DUE

Arduino corresponde al nombre de un proyecto de diseño y manufacturación de placas de desarrollo de hardware open source y open hardware. El objetivo de este proyecto es la confección de placas de desarrollo manipulables según interfaces sencillas. Existen distintos tipos de placas o módulos según las necesidades de un proyecto. Una de ellas es la utilizada por este proyecto, Arduino DUE.

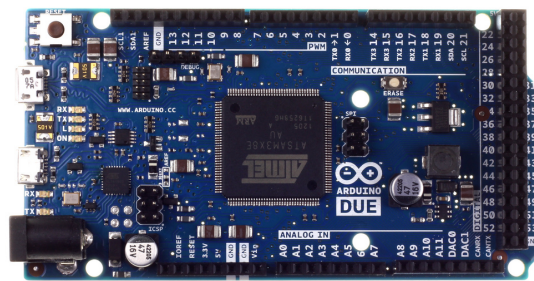
Arduino DUE es una placa con un microcontrolador de 32 bits. Cuenta con 54 pines digitales de entrada y salida (pines desde el 0 al 53), y de los cuales 12 pueden ser utilizados como salidas PWM (pines desde el 2 al 13). Posee 4 puertos seriales (pines 0, 1 y desde 14 hasta 19) y 12 entradas análogas (pines desde A0 hasta A11). Puede ser alimentado con un mínimo de 7V y un máximo de 12V (recomendados) y opera a 3.3V, por lo que aplicar un voltaje mayor en un pin puede dañar la placa.

El software que se carga en la placa de Arduino está basado en el lenguaje de programación C, donde se distinguen las funciones `setup()`, que se ejecuta al encender la placa, `loop()`, que se ejecuta indefinidamente, y las interrupciones, que se detallan a continuación.

Una interrupción o ISR, simplificación de Interrupted Service Routines, corresponde a un tipo de funciones especiales de Arduino que se ejecutan cuando algún pin, definido en el programa como pin de interrupción, sufre un cambio de voltaje, se le aplica, o se le deja de aplicar, interrumpiendo el programa que se está ejecutando en la función `loop()`, para ejecutar una función especial que es posible programar. Todos los pines digitales del Arduino DUE son compatibles con este tipo de funciones, no así en otros modelos como el Arduino Nano, UNO o Mega, que poseen 2, 2 y 3 pines compatibles, respectivamente.



(a) Módulo Arduino Nano



(b) Placa Arduino DUE

Figura 6: Modelos diferentes de Arduinos

2.5. Puentes H

Un puente H es un dispositivo electrónico que permite controlar la rapidez y el sentido de giro de un motor de corriente continua. El puente H es capaz de controlar un motor que opere con un voltaje o corriente mayor a los que pueda soportar un microcontrolador.

Los puentes H utilizados en la construcción de este robot tienen la siguiente disposición de pines y entradas:

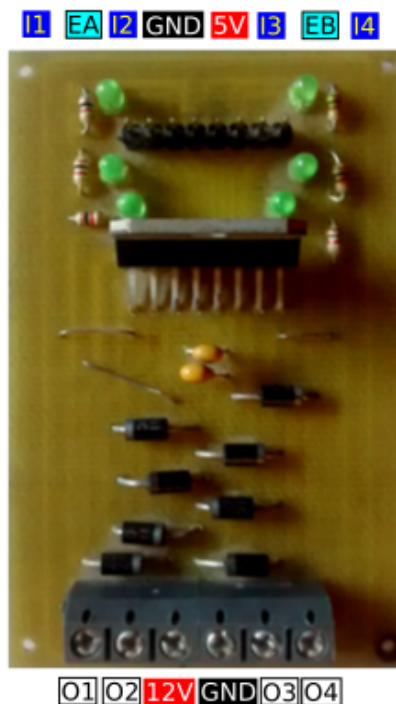


Figura 7: Puente H

Donde los pines 5V y su respectivo GND van conectados al Arduino para alimentación. Los

pinos I1, I2, I3 e I4 se conectan a un pin digital del Arduino y los pines EA y EB se conectan a una salida PWM.

Las salidas O1 y O3 van conectadas a las entradas positivas de los motores a controlar, mientras que las O2 y O4 van a las negativas. Las entradas 12V y GND se conectan a alguna fuente de poder que suministrará la energía a los motores.

Las entradas I1, EA e I2 controlan al motor conectado a los terminales O1 y O2, mientras que las entradas I3, EB e I4 controlan al conectado a los terminales O3 y O4. El control de el primer motor está dado por la siguiente tabla:

Estado de los pines	I1 HIGH	I1 LOW
I2 HIGH	Frenado	Retrocede
I2 LOW	Avanza	Liberado

El control del segundo motor es análogo a este, con las entradas I3 e I4.

2.6. Controlador PID

Un controlador PID (Proporcional-Integral-Derivativo) es un mecanismo de control por retroalimentación que permite reducir el error que presenta un valor medido en comparación al valor deseado mediante una corrección basada en términos proporcionales, derivativos e integrales.

El término proporcional actúa considerando solo el error en el momento de la medición, de forma tal que si este error es grande (o pequeño), le corresponde un ajuste proporcionalmente grande (o pequeño), donde la constante de proporcionalidad es K_p . Matemáticamente, este término es de la forma $P(t) = K_p e(t)$, donde $e(t)$ corresponde al error.

El término integral actúa considerando los valores anteriores del error. Por ejemplo, si existe un error residual después de la aplicación del control proporcional, el término integral busca eliminarlo añadiendo un ajuste relacionado con la acumulación del error en el tiempo. Matemáticamente, este termino corresponde a la integral $I(t) = K_i \int_0^t e(t') dt'$.

El término derivativo esta relacionado con una estimación del comportamiento que presentará el error en el futuro, basada en su tasa de cambio al momento de la medición. Mientras más rápido cambie el error, más grande será el ajuste correspondiente. Matemáticamente, corresponde con el término $D(t) = K_d \frac{de(t)}{dt}$.

Finalmente, el ajuste total se expresa como la suma de todos los términos:

$$u(t) = K_p e(t) + K_i \int_0^t e(t') dt' + K_d \frac{de(t)}{dt}$$

Las constantes K_p , K_i y K_d son positivas y deben ser ajustadas acorde a las respuestas que presente el mecanismo frente al controlador. Al ajustar una de estas constantes como igual a cero, es decir, al deshacerse de uno de los términos anteriormente descritos, se obtiene un controlador más simple, que puede ser PI (Proporcional-Integral), PD (Proporcional-Derivativo), P (Proporcional) o

I (Integral). Dependiendo del caso, puede que sea preferible aplicar alguno de estos antes que el PID.

2.7. Módulo Bluetooth HC-06

El módulo Bluetooth HC-06 corresponde a un tipo de módulo estándar ampliamente comercializado para trabajar con Arduino mediante otro dispositivo que se pueda conectar a él mediante Bluetooth. Está diseñado para trabajar con un voltaje de 5 V. En su versión básica consta de 4 pines, dos para la alimentación, uno de transmisión de datos (TX), que se conecta a un pin de recepción del Arduino (RX) y otro de recepción de datos (RX) que se conecta a un pin de transmisión de datos del Arduino (TX). También es posible encontrar módulos con dos pines extra que sirven para configurar el mismo, dentro de las configuraciones posibles está cambiar su nombre, su pin-contraseña, extraer su dirección MAC, cambiar entre estado maestro-esclavo (Permite una conexión activa o pasiva (única) con otros dispositivos).

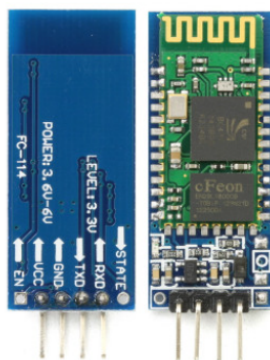


Figura 8: Módulo Bluetooth HC-06

2.8. Programación de aplicaciones para Android

Este proyecto incluye una aplicación Android, las cuales son posible crear de diversas maneras. En internet existen plataformas de programación por bloques como MIT App Inventor que simplifican el proceso de creación de una aplicación.

El método de programación usado en este proyecto fue utilizando el programa de entorno de desarrollo integrado (IDE) de Android Studio, en donde la programación se realiza utilizando los lenguajes Java y el meta-lenguaje XML. La programación en XML se utiliza para almacenar datos de la aplicación, como sus colores, su texto, la disposición de elementos en las ventanas de trabajo, entre otros. La programación en Java corresponde a la implementación de clases que contienen los procesos que ejecutan cada ventana de trabajo (llamadas *activities*), la interacción de los elementos de pantalla así como la transmisión y recepción de datos.

3. Descripción del proyecto

Se dividirá la descripción del proyecto en sus sistemas mecánicos, eléctricos y de software, estos se complementan para lograr los objetivos planteados, ya que el software está programado según la geometría y características electrónicas que se encuentran presentes en el robot.

3.1. Sistema mecánico

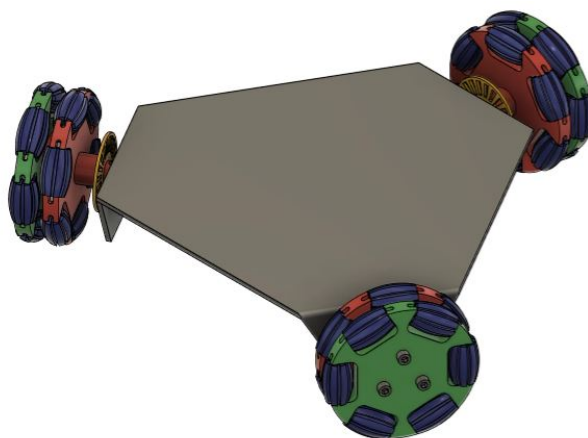


Figura 9: Modelo 3D del sistema mecánico

El sistema mecánico consiste en dos partes principales: el armazón de soportes mecánicos, sobre el cual se fijan los motores y los componentes eléctricos, y las ruedas omnidireccionales. En este caso utilizamos tres ruedas separadas 120 grados una de la otra, dispuestas de forma triangular equidistantes al centro. Las ruedas están diseñadas para que tengan un diámetro de 10 cm en total de un extremo a otro. Cada una está compuesta por tres partes principales: Rueda A, Rueda B y rodillos. Además de un disco perforado para el encoder, los ejes para adosar los rodillos a las ruedas y tres pernos para fijar las ruedas A y B. La división de la rueda en dos partes es debido a la complejidad de la geometría total de la rueda.

3.1.1. Diseño y manufactura del Armazón de soporte de componentes

Debido a que utilizamos tres motores, el armazón debía estar diseñado para que cada motor estuviese a 120 grados con respecto a los otros, para ello fue diseñado con forma de un hexágono irregular, con lados de aproximadamente 11 cm y 17.5 cm intercalados. Esta forma fue obtenida a partir de una circunferencia de radio 13 cm, centrada en el centro de la estructura. La forma hexagonal está inscrita en esta circunferencia, que fue utilizada para lograr construir el armazón de la forma más simétrica posible. En los lados de menor longitud, la lámina presenta un cara hacia abajo con forma de trapecio isocetes, cada una de estas caras tiene cuatro agujeros para fijar un motor (tres agujeros para los pernos y uno para el eje). Se fabrico a partir de una lámina de aluminio de 3 mm de grosor, 50 cm de ancho y 50 cm de largo. se corto un hexagono irregular con lados de 27cm y 5cm de manera intercalada, en cada borde de los lados menores se hizo un doblez de 90 grados a 4cm desde lado hacia el centro, para así lograr el diseño deseado, posteriormente se

hicieron los cuatro agujeros en cada doblez.

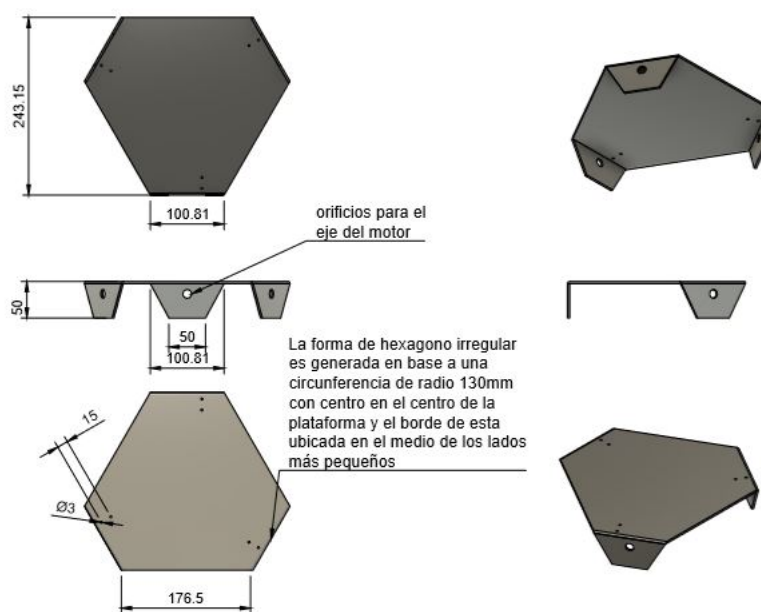


Figura 10: Planos del Armazón

El armazón presenta en su cara principal un agujero para colocar un interruptor, cuatro agujeros para fijar el Arduino DUE, cuatro agujeros para fijar la batería en la parte inferior mediante el uso de dos abrazaderas, un agujero para pasar cables desde la parte inferior a la parte superior y dos agujeros cerca de cada doblez, para fijar las estructuras que cargan los foto-interruptores, estas estructuras son muy simples, se optó por hacerlas con aluminio en forma de 's' con un grosor de 1 mm, hecho de tal modo que el foto-interruptor quede sobre las perforaciones

3.1.2. Diseño y manufactura de los rodillos

Un punto importante es que el diseño de las ruedas se hizo en base a la cantidad y medida de los rodillos, este fue el punto inicial. Según el diseño realizado, se debían hacer 36 rodillos (12 para cada rueda), por esta razón y por los pocos recursos disponibles, estos se modelaron en 3D y se imprimieron con las impresoras 3D Makerbot Replicator 2 en el Fablab de la facultad.

Los rodillos tienen como base un arco de una circunferencia de diámetro 100 mm la cual se describe en toda su parte lateral, poseen un largo total de 30 mm y un eje central de 3 mm de diámetro por donde se unen a las ruedas, su parte más ancha describe una circunferencia de 19 mm de diámetro y los extremos describen una circunferencia de 14.4 mm de diámetro. Debido a que el plástico utilizado en la impresión (PLA) opone muy poca resistencia contra las superficies, se diseñaron rodillos con surcos para así generar más resistencia, los surcos tienen como base un arco de circunferencia de diámetro 96 mm la cual se describe en su parte lateral, la parte más ancha describe una circunferencia de 15 mm de diámetro y los extremos describen una circunferencia de 10.2 mm de diámetro. Cada surco posee 9 grados con respecto al eje central, seguido de 9 grados

de lateral sin surco, esto se repite ocho veces en el rodillo.

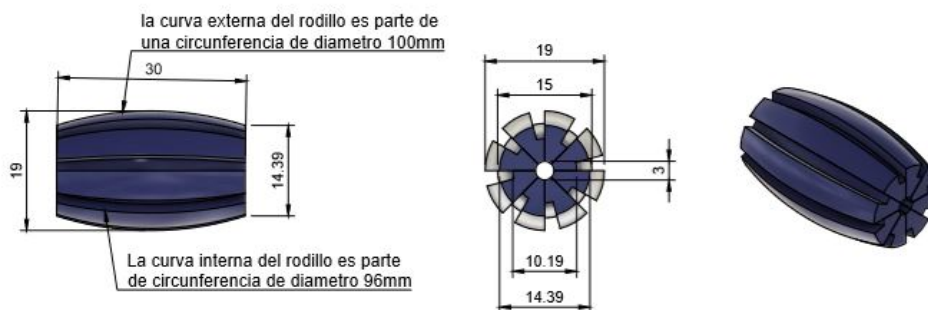


Figura 11: Plano de los rodillos

Por motivos de la impresión el plástico se expande, por lo que al momento de diseñar los rodillos para ser impresos, las medidas se tuvieron que reducir.

3.1.3. Diseño y manufactura de las ruedas A y B

Cabe destacar que todo este proceso se llevó a cabo en el taller metal-mecánico de Vladimir del DIE.

En esencia las partes A y B son idénticas en cuanto a conectar estas con los rodillos, pero difieren en que la rueda A está diseñada por una cara para adosarse al eje del motor y fijar el disco perforado a esta, mientras que la otra cara tiene una saliente circular; por otra parte la rueda B tiene un sacado circular para encajar ahí la saliente de la Rueda A.

Se inició con una barra cilíndrica de duraluminio de aproximadamente 10 cm de diámetro y 20 cm de largo, de aquí se sacaron 3 piezas de 15 mm de largo (para ruedas A) y 3 piezas de 51 mm de largo (para ruedas B). Así, tenemos 3 pares de cada pieza A y B; cada par de piezas se trabajó de la misma manera.

Se utilizó un torno para dar las formas deseadas. La rueda A se torneó hasta lograr una forma (todo esto explicado viendo a lo largo de la barra y cada parte adyacente a la anterior) de 9 mm de largo con 15 mm de diámetro, 16 mm de largo con 20 mm diámetro, 10 mm de largo con 96 mm de diámetro y 12 mm de largo con 40 mm de diámetro, en total el largo es de 47mm, además de una perforación de 10 mm de largo con 6 mm de diámetro en la primera parte del extremo. La rueda B se torneó hasta lograr una pieza de 10 mm de largo con 96 mm de diámetro, además de una perforación de 3 mm de largo con 40 mm de diámetro.

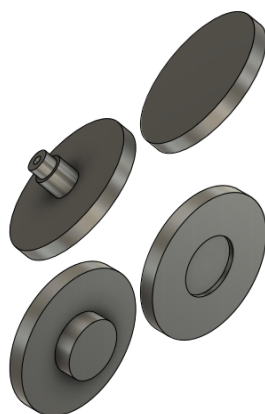
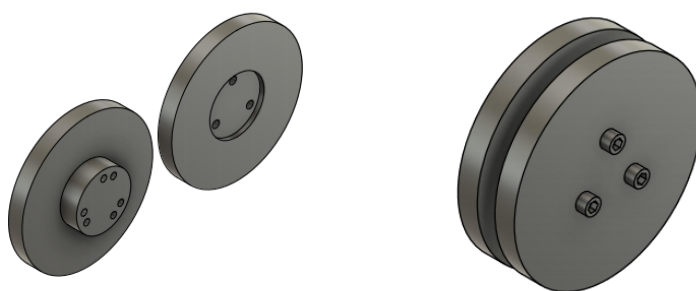


Figura 12: Proceso de manufactura de las ruedas (1)

Luego se hicieron tres perforaciones de 13 mm de largo con 4,3 mm de diámetro en la rueda A (en la parte de 12mm largo y 20mm de diámetro) y en la rueda B tres perforaciones de mismo largo pero de 5 mm de diámetro, las perforaciones deben estar dispuestas en forma triangular y cada una a una distancia de 14,4 mm del centro, en la rueda A se hacen tres perforaciones más con las mismas medidas y con la misma disposición triangular, pero con un desfase de 30° con respecto a las perforaciones previamente hechas, por último en cada orificio de esta rueda se hace un hilo, todo esto con el fin de unir las partes A y B por medio de pernos de 5 mm de diámetro (en este punto no importa que trío de perforaciones se utilicen para unir las), de aquí se obtiene la base de la rueda total con dos discos de 10mm de largo y 96 mm de diámetro, ambas dispuestas de forma paralela y a una distancia de 9 mm.



(a) Ruedas A (izquierda) y B (derecha) separadas.

(b) Ruedas unidas mediante pernos.

Figura 13: Proceso de manufactura de las ruedas (2)

Posteriormente pasó a utilizar una fresadora para hacer seis sacados en los discos antes mencionados, cada sacado es de 14.5 mm de profundidad desde la circunferencia hacia el centro del disco con un ancho de 30 mm, luego el sacado se profundiza 4.5 mm hacia el centro del disco con un ancho de 21 mm, de aquí quedan sobrantes dos esquinas (debido a la diferencia de anchos) que se sacaron con una fresa de punta circular, con lo cual cada borde interno del sacado queda de forma circular, con una base de longitud 21 mm y con una profundidad total de 19 mm y visto desde el

centro hacia afuera a una distancia de 29 mm (se aclara esto último debido a que luego de empezar el sacado ya no se tiene el punto de referencia del borde de la circunferencia, por lo cual ahora los 19 mm son respecto a un borde “imaginario”).

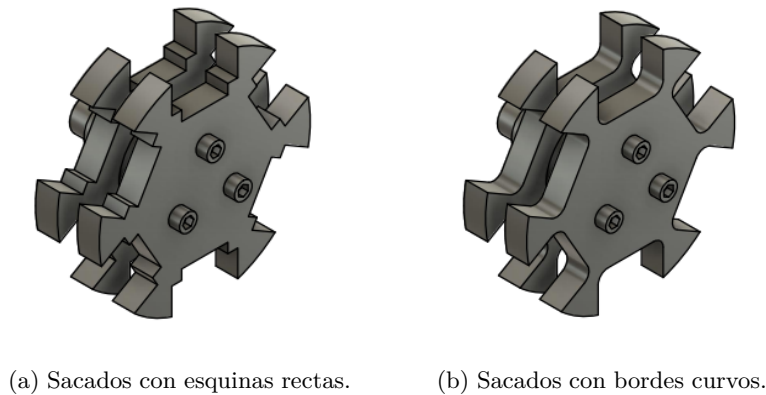


Figura 14: Proceso de manufactura de las ruedas (3)

Con lo anterior nos quedan seis sacados y seis salientes en cada disco, en cada saliente se deben hacer dos perforaciones (una en cada pared del saliente), la perforación debe ser realizada con una fresa de 3 mm, se perfora desde la mitad del saliente con una profundidad de 6,6 mm hacia dentro del disco y 3,5 mm hacia dentro del saliente, con esto quedan 12 sacados en total en cada disco, estos sacados serán para adosar los eje de los rodillos.

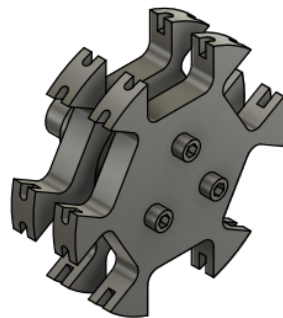


Figura 15: Proceso de manufactura de las ruedas (4)

Por último, se separan las ruedas A y B para hacer un desfase entre ellas, para esto se sacan los pernos, se desfasan 30° las ruedas hasta que las perforaciones de la rueda B calcen con el otro trío de perforaciones de la rueda A, y se vuelven a unir las partes con los pernos.

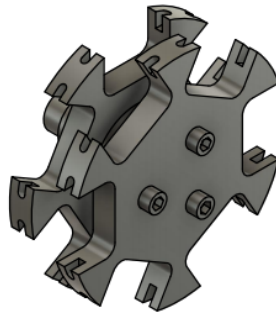


Figura 16: Proceso de manufactura de las ruedas (5)

3.1.4. Diseño y manufactura de los discos perforados

Los discos perforados corresponden a una placa de circuito impreso (PCB) con una geometría particular. Esta se realizó con un programa de modelado 3D sobre un plano en donde se definió una circunferencia de 50 mm de diámetro, luego otras tres concéntricas de 15 mm, 26 mm y 46 mm de diámetro, el círculo interior a la primera circunferencia se le retiró material quedando una perforación de 15 mm de diámetro, luego, entre las circunferencias de 26 y 46 mm se retiraron secciones circulares de 9° de forma intercalada con el objetivo de dejar una figura con 20 perforaciones además de la central.

La geometría realizada en Fusion 360 se sincronizó con Eagle, generando una PCB que fue enviada a cortar con el coordinador de laboratorios del DIE, este material es lo suficientemente opaco para que las foto-interrupciones se realicen de forma efectiva y tiene un espesor de 2 mm.

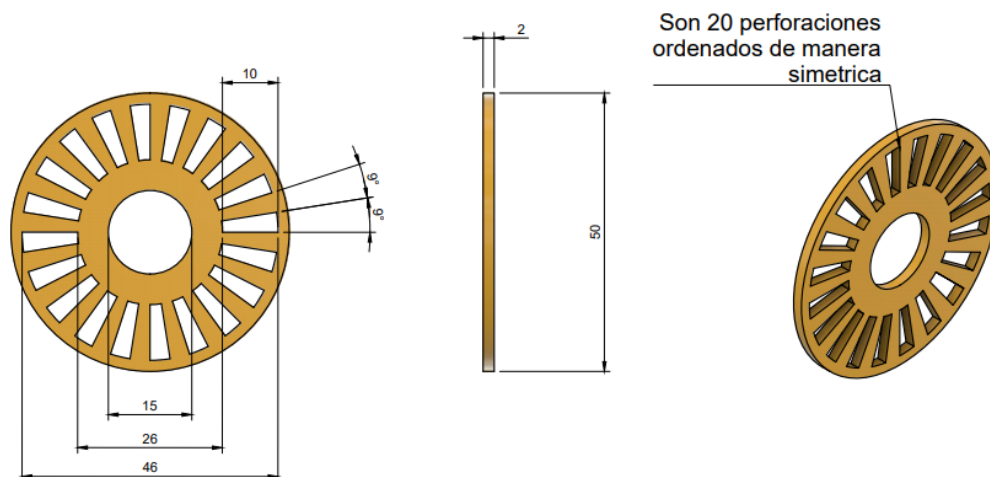


Figura 17: Planos de los discos perforados

3.1.5. Ensamble de partes y rueda final

Finalmente cada rodillo se ensambla en los lugares correspondientes por medio de un eje, en este caso barras de hierro de 36mm de largo y 3mm de diametro, este eje pasa por el eje central del rodillo (el rodillo debe resbalar en el barra) y en cada extremo sobran 3mm de barra los cuales se adosan en los agujeros de los salientes de las ruedas (en nuestro caso los pegamos). Por ultimo, el disco perforado se encaja en un diámetro correspondiente de la rueda A y se pega. Eso es todo.

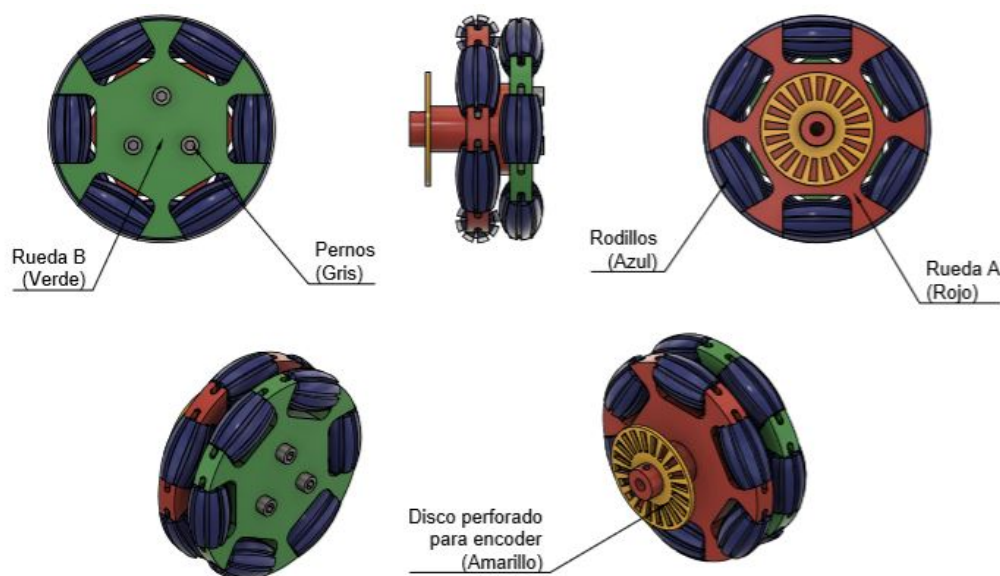


Figura 18: Rueda Final

3.2. Sistema eléctrico

El sistema eléctrico del robot se puede dividir en tres grandes subsistemas, el de alimentación, el de control y el de comunicación. Se detallará cada uno de ellos por separado. En los anexos se incluyen diagramas de conexión de cada subsistema y del cableado general del robot. Se recomienda consultarlos para comprender las conexiones específicas del proyecto.

Los componentes eléctricos y electrónicos utilizados para la ejecución este proyecto corresponden a:

- Batería de ácido y plomo de 12 V y 1,5 Ah de capacidad
- Interruptor de 3 puntas
- 3 motores CC con caja de engranajes de 60 RPM máximas
- 3 foto-interruptores montados en PCB de 10 mm
- 2 puentes H pertenecientes al laboratorio de mecatrónica
- Placa Arduino DUE

- Módulo Bluetooth HC-06

3.2.1. Subsistema de alimentación

La alimentación del robot se realiza con una batería de ácido y plomo, a través de un circuito que se puede abrir y cerrar desde un interruptor instalado en el armazón del robot (del cual solo se utilizó un lado, por ende funciona como un interruptor simple). De este interruptor salen en paralelo tres pares de cables 12V-Tierra, uno va al Arduino, soldado a un conector de 5,5 mm - 2,1 mm diámetro exterior-interior que se conecta al Arduino, los otros dos pares energizan los puentes H.

Todos los componentes que requieren 5 V para funcionar se energizan desde el Arduino, utilizando los pines 5 V y GND. También es importante considerar que todos los cables por los que exista un voltaje de 12 V deben ser mas gruesos que el estándar que se usa en Arduino, ya que los motores consumen una corriente alta para su funcionamiento. La corriente eléctrica del robot encendido es de $0,07 \pm 0,01$ A. Cuando está en funcionamiento completo con todas los motores al máximo de su capacidad, la corriente observada está en el rango de $0,6 \pm 0,1$ A, mientras que los peaks de corriente observado eran del orden de 1,5 A, estos ocurren cuando todas las ruedas cambian de sentido de giro simultáneamente. Estas mediciones se realizaron conectando el robot a una fuente de poder con regulador de corriente.

3.2.2. Subsistema de control

Corresponde a todo el cableado que conecta elementos con el fin de controlar los tres motores del robot. El control, como se ha mencionado, se realiza mediante una placa Arduino DUE, este sistema es posible subdividirlo en otros dos subsistemas.

- Obtención de datos: en este caso corresponde a la recolección de interrupciones por los foto interruptores. Cada foto-interruptor tiene 3 pines, dos de alimentación que se conectan al 5 V y GND del Arduino y otro de señal que entrega voltaje cuando hay interrupción y deja de entregar cuando no la hay, este se conecta a uno de los pines digitales que se configura como interruptor.

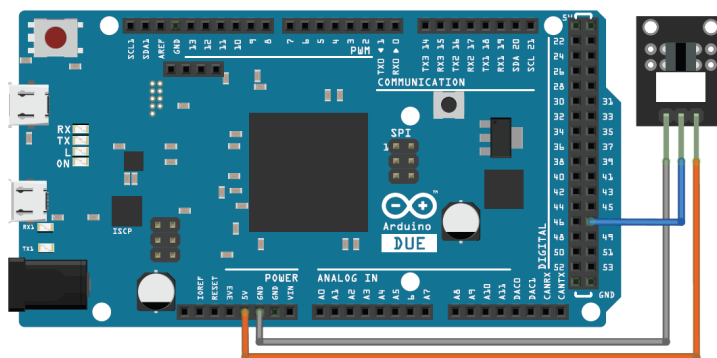


Figura 19: Esquema de conexión de un foto-interruptor

- Control de motores y aplicaciones de velocidad de giro: para hacer esto los motores se deben conectar a un puente H. Este puente tiene dos entradas de alimentación de corriente, un par

5 V - GND que se conecta al Arduino que alimenta los circuitos de control y otro par 12 V - GND que se conecta al sistema de alimentación y redistribuye la corriente hacia los motores según indique la instrucción de control.

Debido a que en este proyecto se utilizan tres motores CC, y cada puente H disponible puede controlar dos motores, se utilizaron 2 puentes H.

Para cada motor conectado al puente H, aparte de la alimentación, llegan tres cables de control (usando 5 V), donde dos de ellos entregan el estado lógico al puente H (avanzar, retroceder, frenar y liberar motores), que se conectan a pines de Arduino que se configuran de salida y el tercero entrega una señal PWM que determina la rapidez con la que se moverá el motor. Aparte de los cables de control, salen dos cables hacia los motores (usando voltajes del rango 0-12 V) que entregan energía según las instrucciones dadas por el microcontrolador.

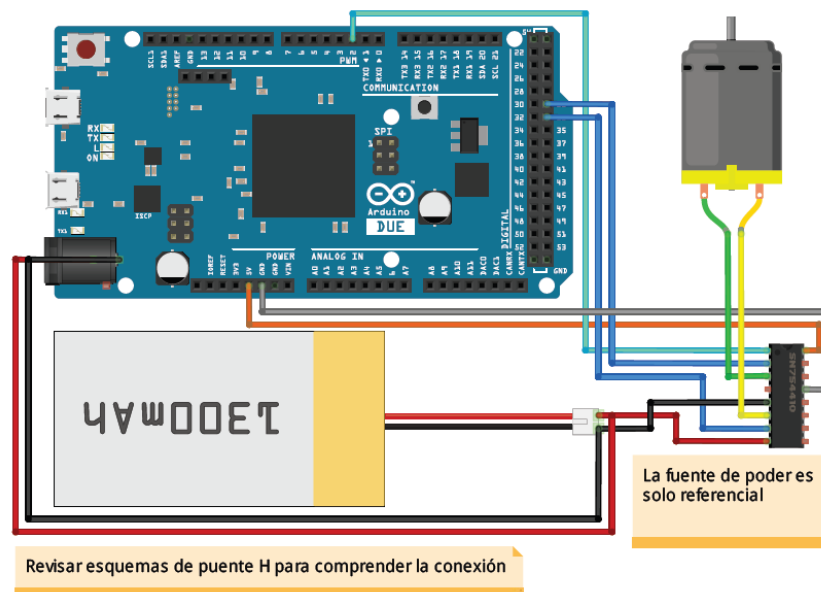


Figura 20: Esquema de conexión de un motor CC

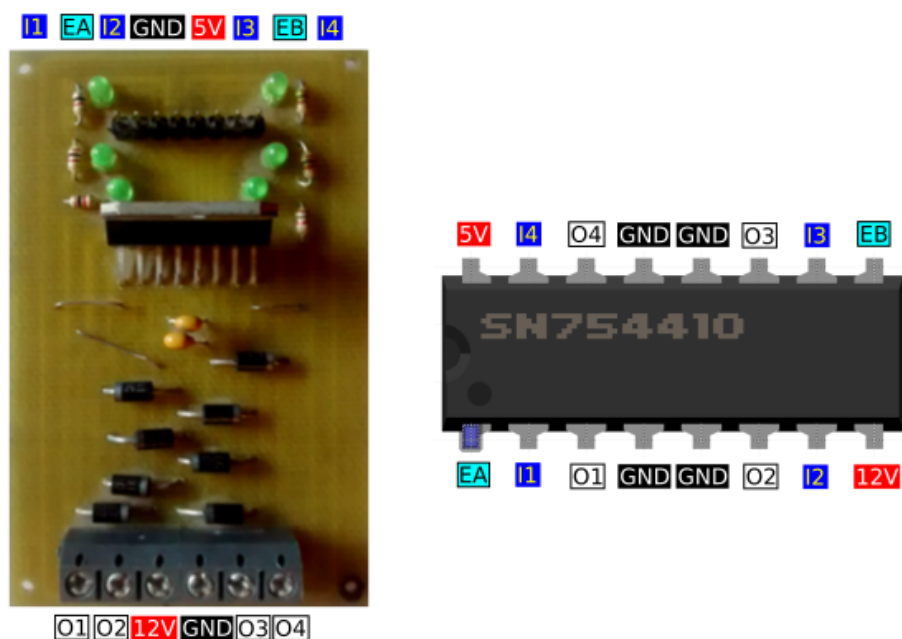


Figura 21: Relación entre los pines del puente H del esquema anterior y el utilizado en el robot

3.2.3. Subsistema de comunicación

El cableado del subsistema de comunicación es mas simple que es de los otros sistemas, ya que solo requiere la conexión del módulo Bluetooth al Arduino, siendo este el único componente. Este requiere cuatro cables para su conexión, dos de ellos son el par 5 V - GND y los otros dos son los de recepción y transmisión que se conectan como fue detallado en los antecedentes generales.

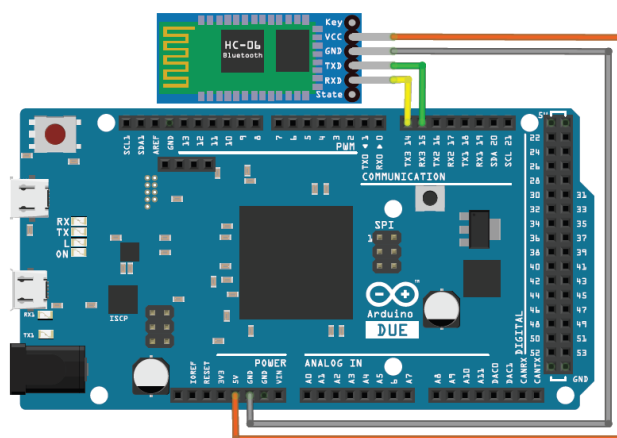


Figura 22: Esquema de conexión del módulo bluetooth

3.3. Software de control

El control por software del robot se realiza mediante Arduino, el que hace todos los procesos necesarios para hacer funcionar los motores. Las instrucciones a realizar por Arduino son enviadas a través del puerto Serial. Dado que Arduino DUE tiene cuatro puertos seriales, se conectó al cuarto puerto (**Serial3**) un módulo Bluetooth HC-06 para entregarle instrucciones de movimiento a través de él. Este módulo puede recibir instrucciones de cualquier dispositivo que sea capaz de conectarse a él y emitir instrucciones en un protocolo que se detallará en las secciones subsiguientes. En particular se desarrolló una aplicación para Android que consta de dos interfaces para enviar distintos tipos de instrucciones al robot.

3.3.1. Principios del movimiento

El funcionamiento y programación de software del robot se basa en ciertos supuestos respecto al mismo para modelar su funcionamiento y computar de manera factible los elementos necesarios para el mismo. A continuación se presenta una lista de los supuestos ideales al momento de programar.

- Las ruedas son perfectamente circulares. Debido a la estructura compleja de las ruedas esto es complejo de lograr, aún así el diseño de los rodillos y su disposición están pensados en lograr que los puntos de contacto de la rueda con el piso se aproximen de manera muy cercana a una circunferencia.
- Las ruedas y rodillos ruedan sin resbalar y la superficie no presenta grandes irregularidades (es plana). Si hay resbalamiento, el movimiento de resultante de las ecuaciones al componerlo no va a resultar en lo deseado, para solventar esto, se hicieron los rodillos con surcos, con el objetivo de aumentar el roce con el suelo.
- Las ruedas tienen un radio de 50 mm. Durante la construcción se intentó seguir esta medida de la forma más precisa posible, pero aún así es difícil de medir, dado que la figura resultante en las ruedas no es cilíndrica.
- Las ruedas están en una disposición triangular equilátera cuyo ortocentro coincide en el centro de masas del robot y la distancia de cada rueda a este es de 170 mm. Este punto es importante, ya que de esto depende que las ecuaciones de movimiento funcionen de forma correcta. Para asegurar esto se hizo la construcción de la forma mas simétrica posible, así como la ubicación de los elementos.

Para realizar un modelamiento del movimiento se consideró un esquema de movimiento omnidireccional:

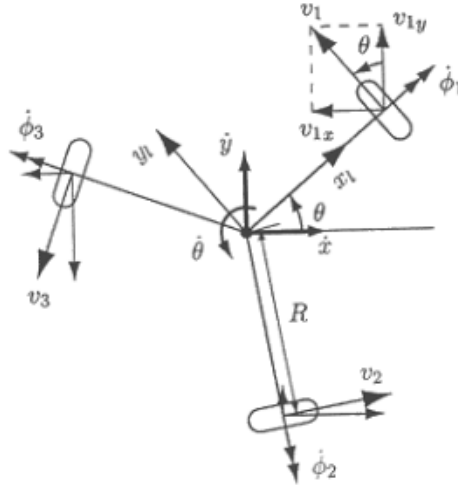


Figura 23: Diagrama de composición de movimiento para una disposición de ruedas triangular

Realizando la deducción de fórmulas de movimiento, que está detallada en la referencia [3] se llega a que la solución está dada por el sistema matricial:

$$\begin{pmatrix} \dot{\phi}_1 \\ \dot{\phi}_2 \\ \dot{\phi}_3 \end{pmatrix} = \frac{1}{r} \begin{pmatrix} -\sin(\theta + \alpha_1) & \cos(\theta + \alpha_1) & R \\ -\sin(\theta + \alpha_2) & \cos(\theta + \alpha_2) & R \\ -\sin(\theta + \alpha_3) & \cos(\theta + \alpha_3) & R \end{pmatrix} \begin{pmatrix} \cos(\theta) & 0 & 0 \\ 0 & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \dot{x}_L \\ \dot{y}_L \\ \dot{\theta} \end{pmatrix}$$

Donde, acorde a la disposición del diagrama, \dot{x}_L corresponde a la rapidez en el eje x , \dot{y}_L la rapidez en el eje y y $\dot{\theta}$ a la rapidez angular en torno a un eje vertical que pasa por su centro de masas, por otro lado $\dot{\phi}_1$, $\dot{\phi}_2$ y $\dot{\phi}_3$ corresponden a las velocidades angulares que se deben aplicar sobre cada rueda para lograr el movimiento.

Finalmente para efectos del proyecto, las funciones resultaron:

$$\dot{\phi}_1 = \frac{(\cos(\theta) \cos(\theta_1) - \sin(\theta) \sin(\theta_1)) v_x - (\sin(\theta) \cos(\theta_1) - \cos(\theta) \sin(\theta_1)) v_y + R\dot{\theta}}{r}$$

$$\dot{\phi}_2 = \frac{(\cos(\theta) \cos(\theta_2) - \sin(\theta) \sin(\theta_2)) v_x - (\sin(\theta) \cos(\theta_2) - \cos(\theta) \sin(\theta_2)) v_y + R\dot{\theta}}{r}$$

$$\dot{\phi}_3 = \frac{(\cos(\theta) \cos(\theta_3) - \sin(\theta) \sin(\theta_3)) v_x - (\sin(\theta) \cos(\theta_3) - \cos(\theta) \sin(\theta_3)) v_y + R\dot{\theta}}{r}$$

En donde $\theta_1 = \theta + \alpha_1$, $\theta_2 = \theta + \alpha_2$, $\theta_3 = \theta + \alpha_3$ y $\theta = \text{atan2}(v_x, v_y)$, la función $\text{atan2}(a, b)$ es aquella que extrae el ángulo que forma (a, b) según la dirección positiva del eje x .

Es importante contemplar que para efectos del movimiento, se considerará la rueda 1 a la frontal, aquella contraria al lado donde se ubica el Arduino DUE, así la 2 y 3 se numeran siguiendo el sentido antihorario respecto al centro del robot. Cada rueda de un número, tendrá asociados el motor y el sistema de encoder del mismo número.

3.3.2. Funciones de movimiento

Las funciones básicas de movimiento del robot corresponden a `Liberar(NMotor)`, `Frenar(NMotor)`, `Avanzar(NMotor,RPM)` y `Retroceder(NMotor,RPM)`, en donde `NMotor` corresponde al número del motor con el que se trabaje y `RPM` a la rapidez angular en RPM con que se desea que se mueva el motor. Notar que cada función corresponde a cada estado lógico distinto de los puentes H, el detalle de estas funciones se tratará con posterioridad.

Funciones de composición de movimiento del robot corresponden a: `MovimientoXYRotRobot(...)`, `MoverXYRotRobot(...)`, `MoverMotores(...)`, `LimitarMov(...)`, `MoverPolarRotRobot(...)` y `FrenarRobot()`.

La función `MovimientoXYRotRobot(...)` corresponde a aquella que realiza todos los cálculos trigonométricos expuestos en la sección anterior, recibe tres parámetros, el primero es la velocidad en el eje x medida en $\frac{\text{mm}}{\text{s}}$, el segundo es la velocidad en el eje y medida en $\frac{\text{mm}}{\text{s}}$ y el tercero una rapidez angular medida en $\frac{\text{grados}}{\text{s}}$. Posterior a realizar los cálculos ya expuestos, almacena los resultados de RPM deseados para cada motor en las variables `RPMMov[0]`, `RPMMov[1]` y `RPMMov[2]`, el código de implementación de esta función corresponde a:

Código 1: Implementación de la función `MovimientoXYRotRobot`

```

1 // MovimientoXYRot, función que dada una velocidad en el eje x en mm/s, una en el eje y en mm/s
  y una rapidez angular en grados/s, modifica la lista RPMMov, con los valores
2 // correspondientes a las RPM que debe ejecutar cada motor según su posición en la lista para
  realizar el movimiento descrito
3 void MovimientoXYRot(double Vx, double Vy, double Rot) {
4     double theta;
5     if (Vy != 0) {
6         theta = atan2(Vx, Vy);
7     }
8     else if (Vx > 0) {
9         theta = PI / 2;
10    }
11    else {
12        theta = -PI / 2;
13    }
14    double RotRads = GradsARads(Rot);
15    double AngParaCal1 = theta + AngRadRuedas[0];
16    double AngParaCal2 = theta + AngRadRuedas[1];
17    double AngParaCal3 = theta + AngRadRuedas[2];
18    double RapAng1_p1 = (cos(theta) * cos(AngParaCal1) - sin(theta) * sin(AngParaCal1)) * Vx;
19    double RapAng2_p1 = (cos(theta) * cos(AngParaCal2) - sin(theta) * sin(AngParaCal2)) * Vx;
20    double RapAng3_p1 = (cos(theta) * cos(AngParaCal3) - sin(theta) * sin(AngParaCal3)) * Vx;
21    double RapAng1_p2 = -(sin(theta) * cos(AngParaCal1) + cos(theta) * sin(AngParaCal1)) * Vy;
22    double RapAng2_p2 = -(sin(theta) * cos(AngParaCal2) + cos(theta) * sin(AngParaCal2)) * Vy;
23    double RapAng3_p2 = -(sin(theta) * cos(AngParaCal3) + cos(theta) * sin(AngParaCal3)) * Vy;
24    double RapAng1_p3 = DistanciaAlCM * RotRads;
25    double RapAng2_p3 = DistanciaAlCM * RotRads;
26    double RapAng3_p3 = DistanciaAlCM * RotRads;
27    double RapAng1 = (RapAng1_p1 + RapAng1_p2 + RapAng1_p3) / RadioRuedas;
28    double RapAng2 = (RapAng2_p1 + RapAng2_p2 + RapAng2_p3) / RadioRuedas;

```



```

29  double RapAng3 = (RapAng3_p1 + RapAng3_p2 + RapAng3_p3) / RadioRuedas;
30  double RPM1 = RadsARPM(RapAng1);
31  double RPM2 = RadsARPM(RapAng2);
32  double RPM3 = RadsARPM(RapAng3);
33  RPMMov[0] = RPM1;
34  RPMMov[1] = RPM2;
35  RPMMov[2] = RPM3;
36  Serial3.println("RPM 1 resultante: "+String(RPM1));
37  Serial3.println("RPM 2 resultante: "+String(RPM2));
38  Serial3.println("RPM 3 resultante: "+String(RPM3));
39 }

```

La función **LimitarMov** toma como argumento tres valores de RPM y verifica que no excedan el límite máximo impuesto de funcionamiento, que para este proyecto fue 50 RPM, a pesar que los motores sean capaces de moverse a 60 RPM, debido a las fluctuaciones de máximo que pueden haber cuando los motores deben hacer un torque significativo. En caso de que uno de los valores exceda el límite de 50 RPM, todos se disminuyen de manera proporcional para mantener la geometría de la composición del movimiento. Los resultados de los cálculos realizados por esta función son almacenados en las variables **RPMLimMov[0]**, **RPMLimMov[1]** y **RPMLimMov[2]**.

La función **MoverMotores** toma como argumento tres valores de RPM que corresponde a las velocidades a las que se desea que se muevan los motores. Lo que hace esta función es limitar estos valores llamando a **LimitarMov**, y con los valores limitados llamar a las funciones de movimiento básicas según sea necesario.

La función **MoverXYRotRobot** toma los mismos parámetros que la función **MovimientoXY-RotRobot** y su función es sencillamente llamar a esta última, para entregar los valores almacenados en las variables **RPMMov** a **MoverMotores**.

La función **MoverPolarRotRobot** toma tres parámetros, el primero corresponde a la rapidez deseada del robot en $\frac{\text{mm}}{\text{s}}$, el segundo al ángulo de dirección del robot, este crece en sentido antihorario desde la dirección positiva del eje x y finalmente una rotación angular en $\frac{\text{grados}}{\text{s}}$.

Finalmente la función **FrenarRobot**, lo que hace es llamar a la función **Frenar** para cada rueda.

3.3.3. Control PID

En principio las funciones **Avanzar** y **Retroceder** deben entregar valores PWM a los motores para que estos se muevan a una velocidad deseada. En realidad eso no ocurre así, ya que para un valor de PWM fijo, la rueda puede girar a distintas velocidades angulares debido a características del piso y el peso del robot. Debido a esto se decidió implementar un control PID en este.

En control PID se implementó desde la librería PID de Brett Beauregard, notar que para la implementación del PID las interrupciones son muy importantes, dado que estas por un lado permiten calcular la rapidez angular de la rueda y entregar un input al PID pero también sirven como instancia para computar y aplicar el PID. A continuación se encuentra un diagrama de funcionamiento de las rutinas realizadas al haber una interrupción.

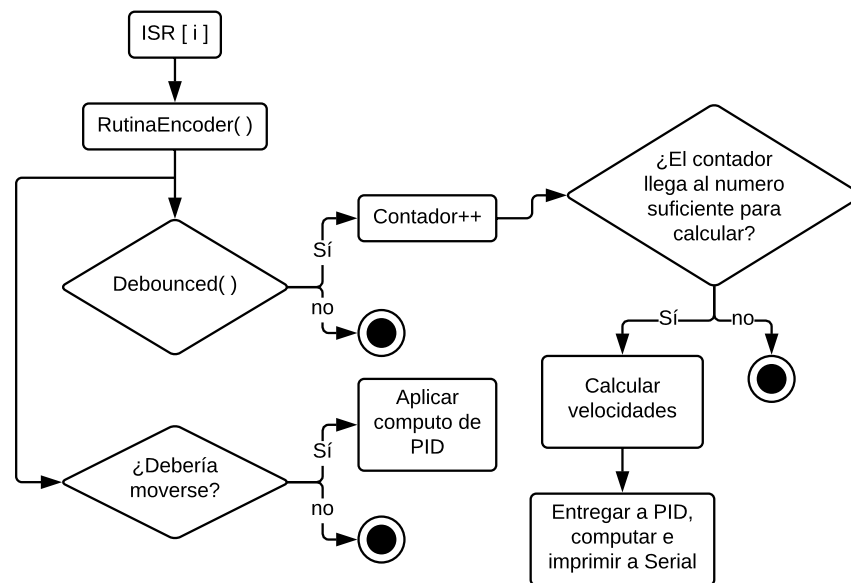


Figura 24: Diagrama de funcionamiento de las ISR

Uno de los problemas encontrados al trabajar con las interrupciones, fue que la señal enviada por el foto-interruptor no era una onda cuadrada perfecta, sino que realizaba oscilaciones en los cambios de estado, lo que provocaba que cada pin fuera interrumpido más de una vez. Para evitar esto, se agregó una función llamada `Debounced()` que sencillamente descarta todas las interrupciones que se den en intervalos de tiempo muy bajos. Aún así en cada interrupción se aplica el computo del PID. A continuación se encuentran las implementaciones de las funciones `Debounced()` y `RutinaEncoder()`.

Código 2: Implementación de la función `Debounced`

```

1 // Debounced, función destinada a ejecutarse en una interrupción, retorna true o false dependiendo
  // si esta misma no se ha ejecutado en un tiempo menor a UltT.
2 // El objetivo es limpiar la lectura de interrupciones erráticas que ocurren en un tiempo muy
  // pequeño.
3 bool Debounced(volatile long UltT) {
4   long Diferencia = micros() - UltT;
5   return Diferencia >= TiempoParaInterrupcion;
6 }

```

Código 3: Implementación de la función `RutinaEncoder`

```

1 // RutinaEncoder, función que ejecuta la acción que debe realizar cada encoder tomando como par
  // ámetro el número de este, cada ISR llama a esta función con el respectivo número del
2 // encoder que ejecutó la interrupción.
3 void RutinaEncoder(byte NEncoder) {
4   byte Num = NEncoder - 1;
5   if (Debounced(UltimoT[Num])) {
6     Contador[Num]++;
7     if (Contador[Num] == Nverificador) {
8       Tiempo_Fin[Num] = millis();
9       RapidezAngMed[Num] = double(Contador[Num]) / (Tiempo_Fin[Num] - Tiempo_Ini[Num]);

```

```

10     RPMMed[Num] = RapidezAngMed[Num] * 60000.0 / TotalPerf;
11     RapidezInPID[Num] = RPMMed[Num] * 255.0 / double(MaxRPM);
12     Tiempo_Ini[Num] = millis();
13     Contador[Num] = 0;
14     PIDs[Num].Compute();
15     if (NEncoder == 2){
16         Serial.println("RPM medidas (" + String(NEncoder) + "): " + String(RPMMed[Num]));
17     }
18 }
19 UltimoT[Num] = micros();
20 }
21 if (Avance[Num] != 1) {
22     analogWrite(MotorPWM[Num], byte(RapidezOutPID[Num]));
23 }
24 }

```

Las variables que se controlaron por PID corresponden a la rapidez de giro de cada rueda, medida según un valor análogo al de PWM entre 0 y 255, donde el input se entrega en la línea 11 del código anterior según las mediciones de velocidad y una conversión lineal de esta a la escala 0-255 según si a rapidez va en 0-60 RPM. El setpoint se calcula de manera similar y se fija en las funciones **Avanzar()** y **Retroceder**, mientras que los cálculos y el output se realizan durante cada interrupción.

La librería PID requiere que se realicen los cálculos en algún momento, estos se implementarán durante cada interrupción. De esto surgen dos problemas, ambos nacen del hecho que el disco perforado, al tener 20 perforaciones, es decir un total de 40 cambios de estado por revolución, posee una resolución muy baja, por ende:

- Al comenzar un movimiento, este lo realiza muy lento, dado que parte con una velocidad inicial 0.
- Al detenerse alguna rueda por algún motivo, y partir desde una velocidad inicial 0, no ocurren interrupciones, por ende el PID nunca realiza cálculos y la rueda no puede comenzar el movimiento.

Para solucionar el primer punto, se implementó en las funciones **Avanzar** y **Retroceder** una instrucción de movimiento inicial que se da a las ruedas independiente del PID según el setpoint calculado, esto es posible verlo en los códigos que se adjuntarán de ambas funciones.

Código 4: Implementación de la función **Avanzar**

```

1 // Avanzar, funcion que dado el numero de un motor y una rapidez angular en RPM, ejecuta las
   acciones necesarias para que avance con las RPM indicadas
2 void Avanzar(byte NMotor, double RPM) {
3     byte indice = NMotor - 1;
4     PIDs[indice].SetMode(AUTOMATIC);
5     RapidezAngDeseada[indice] = RPM;
6     RapidezSetpointPID[indice] = 255 * RPM / MaxRPM;
7     digitalWrite(MotorCtrl[indice][0], HIGH);
8     digitalWrite(MotorCtrl[indice][1], LOW);
9     Avance[indice] = 2;
10    Frenado[indice] = false;
11    Tiempo_Fin[indice] = millis();

```

```
12  PIDs[indice].Compute();
13  analogWrite(MotorPWM[indice], byte(RapidezSetpointPID[indice]));
14  Serial3.println("Motor "+String(NMotor)+" : avanzar "+String(RPM)+" RPM");
15 }
```

Código 5: Implementación de la función Retroceder

```
1 // Retroceder, funcion que dado el numero de un motor y una rapidez angular en RPM, ejecuta las
   acciones necesarias para que retroceda con las RPM indicadas
2 void Retroceder(byte NMotor, double RPM) {
3     byte indice = NMotor - 1;
4     PIDs[indice].SetMode(AUTOMATIC);
5     RapidezAngDeseada[indice] = -RPM;
6     RapidezSetpointPID[indice] = 255 * RPM / MaxRPM;
7     digitalWrite(MotorCtrl[indice][0], LOW);
8     digitalWrite(MotorCtrl[indice][1], HIGH);
9     Avance[indice] = 0;
10    Frenado[indice] = false;
11    Tiempo_Fin[indice] = millis();
12    PIDs[indice].Compute();
13    analogWrite(MotorPWM[indice], byte(RapidezSetpointPID[indice]));
14    Serial3.println("Motor "+String(NMotor)+" : retroceder "+String(RPM)+" RPM");
15 }
```

Código 6: Implementación de la función Frenar

```
1 // Frenar, funcion que dado el numero de un motor ejecuta acciones necesarias para frenarlo
2 void Frenar(byte NMotor) {
3     byte indice = NMotor - 1;
4     PIDs[indice].SetMode(MANUAL);
5     digitalWrite(MotorCtrl[indice][0], HIGH);
6     digitalWrite(MotorCtrl[indice][1], HIGH);
7     RapidezAngDeseada[indice] = 0;
8     RapidezSetpointPID[indice] = 0;
9     Avance[indice] = 1;
10    Frenado[indice] = true;
11    Serial3.println("Motor "+String(NMotor)+" : frenado");
12 }
```

Código 7: Implementación de la función Liberar

```
1 // Liberar, funcion que dado el numero de un motor ejecuta acciones necesarias para que no
   entregue torque y gire libremente
2 void Liberar(byte NMotor) {
3     byte indice = NMotor - 1;
4     PIDs[indice].SetMode(MANUAL);
5     digitalWrite(MotorCtrl[indice][0], LOW);
6     digitalWrite(MotorCtrl[indice][1], LOW);
7     RapidezAngDeseada[indice] = 0;
8     RapidezSetpointPID[indice] = 0;
9     Avance[indice] = 1;
10    Frenado[indice] = false;
```

```

11 Serial3.println("Motor "+String(NMotor)+" : liberado");
12 }

```

Notar que cada función aplica el estado lógico correspondiente para el puente H. Las funciones se implementan distinto dependiendo si se quiere ejecutar un movimiento o no, notar que **Frenar()** y **Liberar()** simplemente apagan el PID, por otro lado las funciones **Avanzar()** y **Retroceder()** encienden el PID y entregan la velocidad inicial dada por el setpoint.

Para solucionar el segundo punto problemático mencionado, se implementó en el `loop()` de Arduino una función que constantemente estuviera revisando si una rueda estaba detenida o no, para hacer esto, se consideró un tiempo mínimo entre interrupciones, en donde si no ocurrían dos consecutivas, la rueda consideraba detenida. Esta función se llamó **VerificarDetenciones()**, en caso de cumplir la condición de detención se le entrega una velocidad medida de 0 y se fuerza al PID a realizar un computo. Su implementación se encuentra a continuación:

Código 8: Implementación de la función **VerificarDetenciones**

```

1 // VerificarDetenciones, función implementada en el loop que verifica el tiempo transcurrido en las
  últimas interrupciones de los encoder, en caso que el motor asociado a este deba
2 // moverse y no hayan interrupciones en un tiempo definido, se ajusta la rapidez del motor a cero y
  se procede a ejecutar el control PID.
3 void VerificarDetenciones(){
4   for (byte i = 0; i < 3; i++){
5     if ((abs(long(Tiempo_Fin[i])-long(millis()))>TiempoDetencion) && (Avance[i]!=1)){
6       RapidezInPID[i] = 0;
7       PIDs[i].Compute();
8       analogWrite(MotorPWM[i], byte(RapidezOutPID[i]));
9       Serial3.println("Rueda "+String(i+1)+" Detenida");
10    }
11  }
12 }

```

Constantes importantes para la aplicación del PID son las K_p , K_i , K_d y su tiempo de muestreo, el tiempo de muestreo es aquel mínimo en donde se consideran los cálculos como aplicables, esto se fija para intentar lograr una regularidad en el funcionamiento del PID. En el caso de este proyecto corresponde a 200 ms. Las constantes K , se fijaron en función de diversas pruebas de movimiento realizadas con el robot ensamblado, donde para su versión final se cargaron constantes $K_p = 0,5$, $K_i = 3$ y $K_d = 0$, en donde se descartó la parte derivativa debido a que causaba una cantidad importante de oscilaciones en la rapidez de la rueda, lo que impedía un movimiento recto del robot o a rapidez constante, resultando finalmente en la operación de un control PI y no un PID.

El resto de las constantes que sea necesario conocer se encuentran en el fichero `Codigo_Omnidireccional.ino` que se encuentra en la documentación anexa.

3.3.4. Comunicación Bluetooth

Dado que, como se mencionó previamente, el Arduino recibe instrucciones desde un puerto Serial, que para efectos de nuestro proyecto está conectado a un módulo Bluetooth, se estableció un protocolo de comunicación entendible por el Arduino. Para recibir instrucciones, este recibe una

cadena de caracteres con el siguiente formato:

<TOK1,TOK2,TOK3,...>

Donde TOK1, TOK2, etc. corresponden a *tokens*, que son en realidad instrucciones de símbolos, letras y números que están separadas por un delimitador, que en este caso corresponde al carácter ",". Notar que los caracteres de inicio y cierre de una instrucción son < y > respectivamente.

Para recibir las instrucciones se trabaja con dos memorias, una corresponde al *buffer* que es un lugar donde se van almacenando las instrucciones válidas, es decir, las que se encuentran entre < y >. En este caso el buffer terminaría almacenando TOK1,TOK2,TOK3,... por otro lado una matriz de caracteres (o lista de strings) que en el caso de este proyecto se llamó simplemente **Param[][]** que almacena los tokens extraídos del buffer, en este caso Param almacenaría TOK1 ; TOK2 ; TOK3 y más en caso de ser necesario.

La función que está a la escucha del puerto serial se implementó como **RecepcionSerial**, y la función que analiza el buffer como **ParseBuffer**, la sintaxis para el caso particular del proyecto es como sigue:

<Función,Argumento 1,Argumento 2,Argumento 3, ... >

Es decir, el primer token corresponde a una función que el Arduino ve si ejecuta o no, que corresponden a comandos de 3 caracteres en mayúsculas, seguidos de los parámetros de la función que se quiere llamar.

A continuación se deja un diagrama de funcionamiento de la función **RecepciónSerial** así como su implementación y la de **ParseBuffer**.

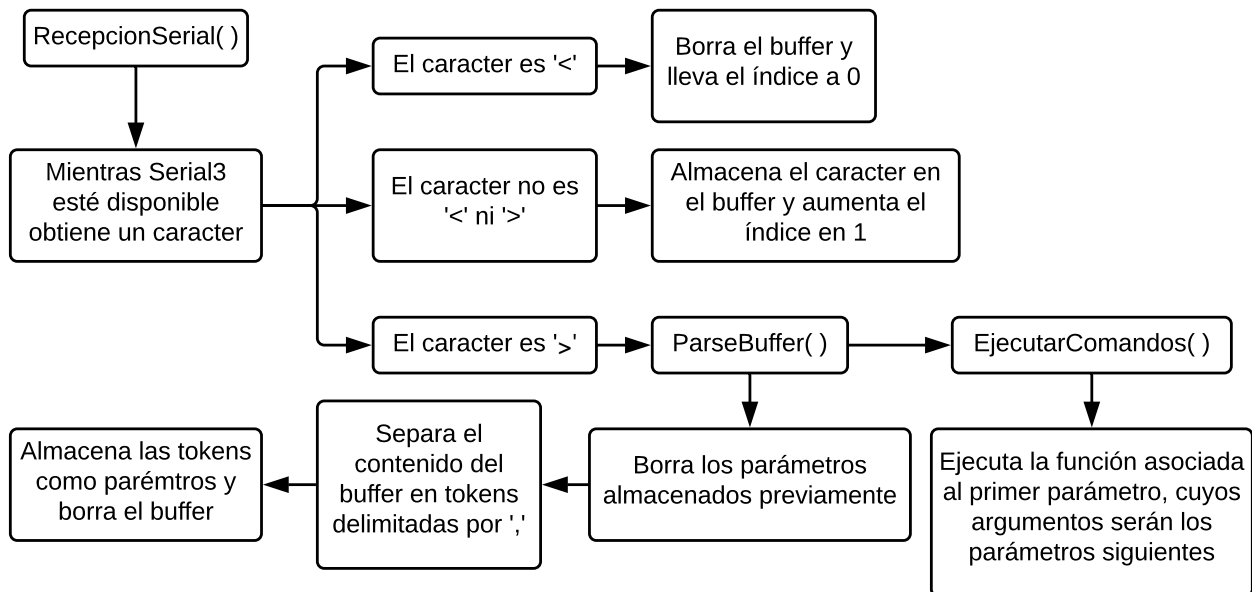


Figura 25: Diagrama de funcionamiento de la función `RecepciónSerial`

Es importante conocer que la función expuesta se encuentra constantemente a la escucha a una velocidad de recepción de 9600 bauds, por ende esa se ubicó en el `loop()` de Arduino. Lo que hace la función `EjecutarComandos` es simplemente comparar el primer token con funciones programadas y entregar los siguientes como argumentos de las funciones, llamándolas.

Código 9: Implementación de la función `RecepcionSerial`

```

1 // RecepcionSerial, función principal que administra la comunicación con el puerto serial ,
  validando los datos recibidos e invocando funciones que los almacenen en Buffer, en caso de
2 // recibir el caracter especial definido como caracter inicial limpia el buffer, el caso de recibir
  el caracter especial definido como caracter final ejecuta el análisis de los datos
3 // recibidos, todo ocurre mientras hayan datos disponibles en el serial, en caso contrario la funci
  ón no ejecuta acción.
4 void RecepcionSerial() {
5   while (Serial3.available() > 0) {
6     Caracter = Serial3.read();
7     if (Caracter == CaractIni) {
8       memset(Buffer, '\0', SizeBuffer);
9       IndiceCaract = 0;
10    }
11    else if (Caracter == CaractFin) {
12      ParseBuffer();
  
```

```

13     EjecutarComandos();
14 }
15 else {
16     Buffer[IndiceCaract] = Caracter;
17     IndiceCaract++;
18 }
19 }
20 }

```

Código 10: Implementación de la función ParseBuffer

```

1 // ParseBuffer, función que al ser ejecutada trabaja el contenido ASCII almacenado en la variable
  // buffer y realiza un análisis gramatical de este para separar el string por el delimitador
2 // definido, almacenando los parámetros resultantes (tokens) en la matriz Param.
3 void ParseBuffer() {
4     memset(Param, '\0', SizeParam);
5     char *Token = strtok(Buffer, Delim);
6     while (Token != NULL) {
7         strcpy(Param[IndiceParam], Token);
8         Token = strtok(NULL, Delim);
9         IndiceParam++;
10    }
11    IndiceParam = 0;
12    memset(Buffer, '\0', SizeBuffer);
13 }

```

También se adjuntará el contrato de la función EjecutarComandos para comprender que funciones llama.

Código 11: Contrato de la función EjecutarComandos

```

1 // EjecutarComandos, función que lee los parámetros contenidos en la matriz Param y los trabaja en
  // función de lo que sea necesario siguiendo la sintaxis <Funcion,Arg1,Arg2,Arg3,...> Es decir, el
  // primer parámetro corresponde al identificador de la función y los siguientes a los argumentos
  // que esta requiera, identificadores a funciones del robot corresponden a:
2 // XYR : MoverXYRotRobot(), recibe 3 argumentos
3 // POL : MoverPolarRotRobot(), recibe 3 argumentos
4 // MMT : MoverMotores(), recibe 3 argumentos
5 // FRN : FrenarRobot(), no recibe argumentos
6 // PID : SetTunings(), recibe 3 argumentos

```

Así finalmente un ejemplo de comando entregado al Arduino corresponde a:

<XYR,200,-100,0>

El cual enviará una instrucción que logrará que el robot se mueva en un vector compuesto por los movimientos de $200 \frac{\text{mm}}{\text{s}}$ hacia la derecha, $100 \frac{\text{mm}}{\text{s}}$ hacia atrás y sin rotación.

3.3.5. Cohesión de las funciones implementadas en Arduino

Finalmente, todas las llamadas importantes de las funciones se realizan en las interrupciones, el `loop()` y el `setup()`, donde previamente se definen todas las funciones expuestas. Más aún, se

declaran diversas variables y arreglos de variables, mientras que en el `setup()` se declaran los pines y se inicializan los controladores PID, asimismo la comunicación Serial. A continuación se deja un diagrama de la implementación del `loop()` así como la implementación misma:

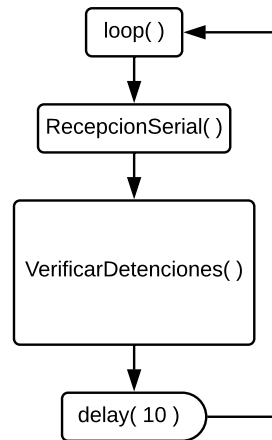


Figura 26: Diagrama de funcionamiento del `loop()` de Arduino

Código 12: Implementación del `loop()` de Arduino

```
1 void loop() {  
2   RecepcionSerial();  
3   VerificarDetenciones();  
4   delay(10);  
5 }
```

3.3.6. Aplicación de Android de control

La aplicación de Android, programada en Android Studio, se basa en el principio de envío de comandos con los protocolos ya expuestos vía Bluetooth al Arduino, esta consta de tres ventanas de trabajo, o como se denomina en la programación de aplicaciones, tres activities. A continuación se deja la captura de pantalla de estas activities.



Figura 27: Capturas de pantalla de las ventanas de trabajo de la aplicación en Android

La ventana (a) corresponde a la ventana de trabajo principal del robot, dispone de una botonera donde cada boton envía una instrucción del tipo $\langle \text{POL}, \text{MOV}, \text{XXX}, \text{ROT} \rangle$, donde para las instrucciones de dirección MOV vale 1000, forzando al robot a ir a su velocidad máxima en esa dirección, XXX representa el ángulo medido desde la dirección hacia adelante de forma antihoraria y ROT vale 0. Por otro lado las funciones de rotación dan instrucciones con MOV y XXX valiendo 0 con ROT valor -1000 o 1000, forzando al robot a rotar a su velocidad máxima en un sentido o el otro. Estos botones arrojan textos de advertencia si se intenta dar la instrucción sin estar conectado al módulo Bluetooth. El botón central rojo es el de freno y envía la instrucción $\langle \text{FRN} \rangle$.

El botón Conectar a Bluetooth ejecuta un método que establece un canal de comunicación con cualquier dispositivo llamado HC-06 ubicado en las cercanías y con quien se encuentre ya enlazado, entregando un texto de advertencia si esto no es posible.

El boton Menú conecta la ventana (a) a la ventana (b). La ventana (b) corresponde al menú de ventanas disponibles en la aplicación que es potencialmente expansible, está implementada con un elemento *RecyclerView* que ordena elementos en este caso en forma de lista y lleva a cada ventana de trabajo al momento de clicar en ellos. Control Remoto corresponde a la venta (a) y Control individual de motores a la ventana (c).

La ventana (c) que cuenta con 3 *SeekBar*, es decir deslizadores, 3 botones y un *ToggleButton* que es un boton estilo switch y los botones de freno y menú. Los botones freno y menú realizan una función análoga a la realizada en la ventana (a).

El boton switch “Enviar constantemente” cumple la función de determinar si las instrucciones que se envían al Arduino son puntuales o son continuas, en el primer caso, lo que se debe hacer es ajustar una velocidad en RPM deseadas para cada motor (cada deslizador controla un motor) y apretar el botón “Enviar RPM” para enviar la instrucción, en el otro caso, cuando el botón está activado, el botón de envío se inhabilita, y se van enviando instrucciones al Arduino a medida que se van modificando los deslizadores.

Se considera que el estado de entrega de la aplicación está en fase beta, ya que se debe probar aún mas y encontrar posible errores (*bugs*). Aún así, la versión es estable, ya que no presenta ningún crasheo al ser probada en un teléfono Xiaomi Redmi Note 4 con Android 7.0 NRD90M instalado.

4. Resultados

Los resultados del proyecto del robot omnidireccional pueden resumirse en los siguientes puntos:

El robot es capaz de desplazarse de forma omnidireccional, en las direcciones y velocidades entregadas mediante comunicación bluetooth con un dispositivo Android. Sin embargo, presenta una pequeña desviación al momento de empezar a desplazarse en las direcciones laterales. Esto es debido a que la relación entre el PWM enviado al puente H y la rapidez angular del motor no es lineal, esto dificulta la precisión en los primeros instantes del movimiento, ya que las ruedas podrían comenzar a rotar con una rapidez angular menor o mayor a la requerida (Usualmente si se requiere una velocidad alta, la rueda comienza girando más lento, y si se requiere una velocidad baja, la rueda comienza girando más rápido).

Este comportamiento no se prolonga en el tiempo, ya que los encoder y el controlador PID funcionan de forma satisfactoria, siendo capaces de ajustar la rapidez angular de las ruedas, de forma que la trayectoria del robot sea la requerida (pasando por alto la desviación inicial). No obstante, al ordenar que una rueda se mueva a una velocidad baja (del orden de 5 RPM), el robot puede presentar dificultades para alcanzarla, y en lugar de ello la rueda podría detenerse y avanzar de forma intermitente. Además, cuando se exige una velocidad del orden de las 40 RPM, el controlador PID podría causar una oscilación en la rapidez angular de la rueda. Este comportamiento no es consistente, es decir, no siempre ocurre, a pesar de darse condiciones iniciales similares.

El robot requiere una superficie rugosa para desplazarse, ya que en superficies más lisas tiende a resbalar, resultando en movimientos no deseados. Esto debido al escaso agarre del plástico que conforma los rodillos con el suelo. En superficies adecuadas, los rodillos rotan con facilidad sobre su eje, sin ofrecer resistencia al movimiento omnidireccional de la rueda.

A continuación se adjuntan imágenes de las ruedas y el robot ensamblados:

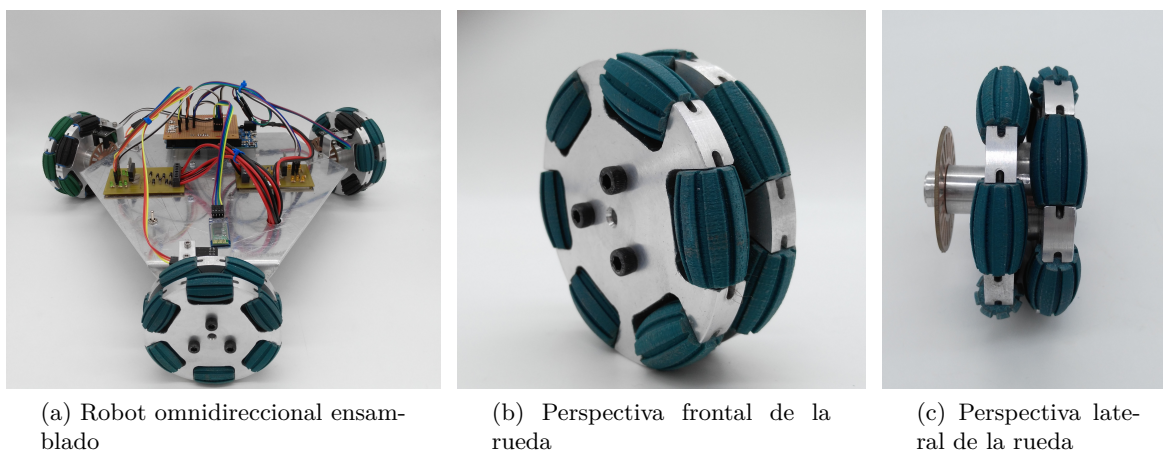


Figura 28: Fotografías del robot terminado

5. Conclusiones

De los objetivos presentados en la introducción, entiendáse, la implementación de un movimiento omnidireccional, de un controlador PID y encoder que ajusten el movimiento de las ruedas del robot, de un sistema de comunicación inalámbrico mediante el cual puedan ingresarse ordenes que el robot seguirá, y de un sistema de seguridad basado en sensores de proximidad para evitar choques, nos damos cuenta de que solo el último objetivo no fue cumplido. Esto probablemente sea debido a un fallo en la estimación del tiempo que iba a tomar la manufactura del robot, en particular la de las ruedas, en las que en total se invirtieron aproximadamente tres semanas de trabajo en el taller metal-mecánico de Vladimir del DIE.

Entre los puntos donde el robot es mejorable destaca el poco agarre que presentan las ruedas al exponerlas a superficies lisas. Esto podría arreglarse mediante la utilización de otro material para su manufactura, o mediante algún recubrimiento con un material que sea capaz de ejercer mayor fricción.

También se consideró la implementación de una interfaz análoga en la aplicación, donde se pudiera fácilmente enviar ordenes que van mas allá de las ocho direcciones básicas implementadas, además de una interfaz más intuitiva para manejar el robot en sus direcciones básicas, donde los botones responderían al mantenerse presionados, en contraste a como lo hacen actualmente. Sin embargo, estas ideas tampoco pudieron implementarse, debido también al tiempo limitado.

Entre los aprendizajes adquiridos podemos notar la utilización de máquinas como el torno, la fresadora y la perforadora, presentes en el taller mecánico, las impresoras 3D disponibles en el Fablab, la implementación de un circuito eléctrico básico y de un programa capaz de ejecutar las instrucciones deseadas, donde se incluye la aplicación de un controlador PID, y una aplicación para Android que facilita la comunicación mediante Bluetooth con el Arduino.

Referencias

- [1] Arduino. Documentación de Arduino DUE. Consultado el 12 de agosto de 2018.
URL: <https://store.arduino.cc/usa/arduino-due> (en ingles)
- [2] 42 Bots. Datasheet del módulo Bluetooth HC-06 y configuración con Arduino. Consultado el 12 de agosto de 2018.
URL: <https://42bots.com/tutorials/hc-06-bluetooth-module-datasheet-and-configuration-with-arduino/> (en ingles)
- [3] T. A. Baede (2006). Motion control of an omnidirectional mobile robot. Singapur: National University of Singapore.
- [4] Ching-Chih Tsai, Li-Bin Jiang, Tai-Yu Wang, Tung-Sheng Wang (2005). Kinematics Control of an Omnidirectional Mobile Robot. Taiwán: National Chung Hsing University.
- [5] Olaf Diegel, Aparna Bradve, Glen Bright, Johan Podgieter, Sylvester Tlale Mechatronics and Robotics Reseach Group (2002). Improved Mecanum Wheel Design for Omnidirectional Robots. Nueva Zelanda: Institute of Technology and Engineering, Massey University.
- [6] Equipo docente EI2001 - Robótica y Mecatrónica 2015-1. Guía Teórica #3. Chile: Universidad de Chile.
- [7] Óscar Torrente Artero (2013). Arduino. Curso Básico de formación. México. Editorial Alfaomega.