TRABAJO FINAL DE GRADO

DISEÑO E IMPLEMENTACIÓN DE UN VEHÍCULO DE CUATRO RUEDAS OMNIDIRECCIONALES

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

Autor: Adrián Campos Siurana

Tutor: Vicente Fermín Casanova Calvo

Valencia, Julio 2018

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Resumen

En el presente proyecto se desarrollará el diseño y la implementación de un vehículo de 4 ruedas omnidireccionales de bajo coste. El robot móvil está pensado para el estudio académico de un vehículo con un tercer grado de libertad añadido, proporcionado por el tipo de ruedas empleado, y el impacto y posibles aplicaciones que se podrían aportar a la industria y a la sociedad este tipo de vehículos.

El diseño propuesto es un vehículo de cuatro ruedas utilizando ruedas omnidireccionales de tipo *Mecanum* en posición cruzadas, es decir, las ruedas izquierda delantera y derecha trasera tienen la misma configuración, distinta a la delantera derecha y trasera izquierda. El bucle de control está gobernado por un microcontrolador del fabricante Arduino, encargado de calcular las acciones de control de los motores y de tomar los datos de la velocidad a partir de los encoders de estos.

Para las distintas pruebas de su funcionalidad se han utilizado dos formas distintas. La primera, a través de un diagrama de Simulink, empleando secuencias de velocidades para enviar las velocidades a los motores, o utilizando los sensores del giroscopio de un Smartphone, comunicados con el ordenador vía Wi-Fi. La segunda, a través de una aplicación Android en el que se diseña un mando de control para los tres grados de libertad.

La comunicación con el microcontrolador ha sido realizada mediante un módulo Bluetooth, ya sea a través del ordenador o del Smartphone.

También se ha recogido la información de la velocidad de giro de las ruedas, por parte del microcontrolador, para calcular su posición en el espacio, y compararla con la posición real con ayuda de una cámara.

Agradecimientos

En primer lugar, agradecer a mi tutor, Vicente Fermín Casanova Calvo, por su ayuda y dedicación, proporcionándome la oportunidad de trabajar en un proyecto tan motivador, divertido y didáctico.

A mi familia y amigos, por el apoyo incondicional, motivación y amor en estos años académicos, que me han ayudado a crecer como persona.

Índice

Índice de Figuras

Introducción

En los últimos años, el uso de robots de forma cotidiana se ha incrementado rápidamente. Hasta la fecha, se han desarrollado distintos tipos de robots móviles con distintas formas de tracción. Dentro de la categoría de robots basados en ruedas como medio de movilidad, se han desarrollado diferentes variantes de acuerdo con el uso y ambiente en donde se lleve a cabo su tarea. La mayoría de las configuraciones de diseño usando ruedas convencionales permiten a los vehículos dos grados de libertad, acotando las trayectorias posibles a realizar, sobre todo en espacios reducidos o con obstáculos, siendo el manejo para evitar colisiones más complejo.

En búsqueda de mejorar los diseños de robots móviles para incrementar la maniobrabilidad se han estudiado otras configuraciones de ruedas y el uso de otro tipo de ruedas especiales. Estas nuevas configuraciones introducen los robots omnidireccionales. Se definen robots omnidireccionales a aquellos capaces de realizar cualquier movilidad en cualquier dirección, desde un punto arbitrario, en un plano, sin tener que rotar previo al comienzo del desplazamiento. Es decir, es capaz de realizar movimientos en cualquiera de las componentes del plano, bien sean traslaciones o rotaciones, a partir de un estado de movilidad. Por lo tanto, un robot omnidireccional cuenta con tres grados de libertad a su disposición. Este grado de libertad añadido proporciona una gran cantidad de aplicaciones para estos robots, como, por ejemplo, vehículos industriales, como un montacargas omnidireccional en un almacén con espacio reducido, juguetes, académicos, entre otros.

En este trabajo se va a estudiar un tipo de específico de ruedas omnidireccionales llamado *Mecanum.* Estas ruedas se caracterizan por tener en su periferia una corona de rodillos colocados en un ángulo de 45º grados respecto a su eje de giro. Estos rodillos brindan una componente pasiva de movilidad en el eje perpendicular al eje de giro de las ruedas. Así, en función de las velocidades de giro de las ruedas del robot, se brinda cualquier movimiento posible.

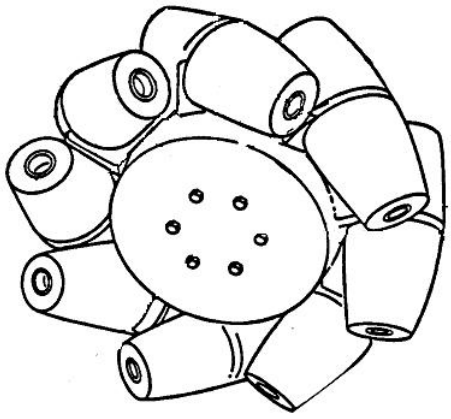


Figura 1: Rueda omnidireccional de tipo Mecanum.

Objetivos

Durante el siguiente trabajo se pretende conseguir completar los siguientes objetivos:

- Comprobar la validez de la cinemática desarrollada para un robot móvil de cuatro ruedas *Mecanum*.
- Diseño de un vehículo de 4 ruedas omnidireccionales funcional para realizar pruebas competentes y fiables.
- Implementar el diseño realizado y realizar las mediciones para comprobar la cinemática desarrollada.
- Realizar un control con Arduino capaz y efectivo.
- Manejar el robot con una aplicación Bluetooth para Smartphone.

Estudio teórico

Existen varias formas de componer las ruedas omnidireccionales en un vehículo en función del número de ruedas que se pretendan usar y donde. Debido a las ruedas *Mecanum*, estas ofrecen un movimiento en las dos direcciones perpendiculares de forma pasiva, ya que los rodillos están colocados en una posición de 45º. Esta disposición de los rodillos, en función de la composición del número de ruedas y de su posición en el robot, en función de la dirección de giro, proporciona distintas direcciones de movimiento según el resultado de la suma de velocidades de cada rueda en cada instante de tiempo, dando como resultado a todas las direcciones posibles en el plano, añadido por ese nuevo grado de libertad proporcionado por las ruedas.

En este proyecto, se ha optado por seguir la implementación como se muestra en la *Figura 2,* un vehículo de 4 ruedas con las ruedas en configuración cruzada, es decir, la rueda delantera izquierda y la trasera derecha tienen los rodillos en sentido izquierdo, mientras que la rueda delantera derecha y trasera izquierda tienen los rodillos en sentido derecho. Controlando individualmente las ruedas del vehículo, mediante la combinación lineal de las fuerzas resultantes y de las velocidades de cada una, se obtiene cualquier dirección y sentido de desplazamiento de forma instantánea.

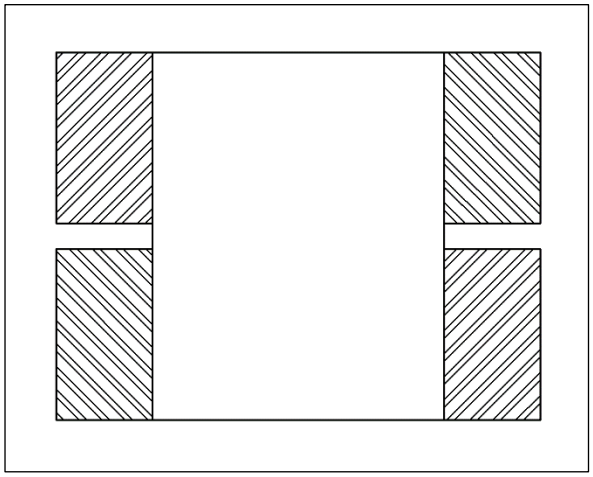


Figura 2: Disposición de ruedas Mecanum en un robot de 4 ruedas

En la siguiente figura se muestra la combinación de las direcciones de giro de cada rueda y los movimientos resultantes que realiza el robot como resultado de la suma de estas:

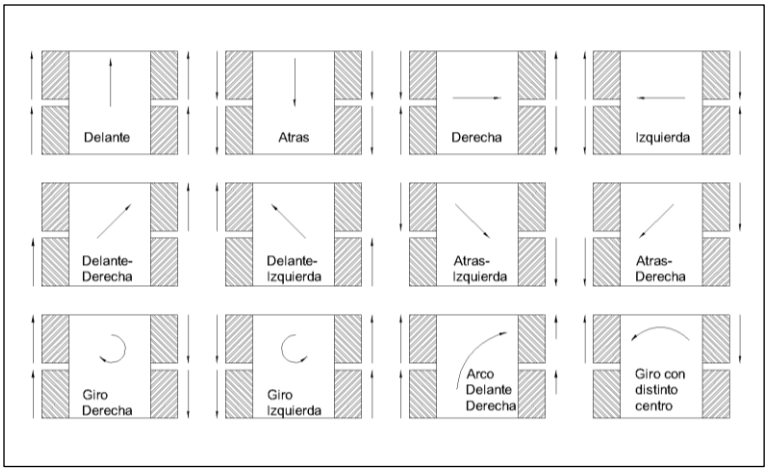


Figura 3: Combinación de velocidades para obtener cualquier dirección de desplazamiento

A continuación, usando la configuración acordada anteriormente, se procede al estudio de las velocidades de todos los componentes del robot para la elaboración de una relación entre las velocidades individuales de cada rueda y las velocidades generales del robot. En la *Figura 4* se puede ver la disposición de las ruedas, vistas las proyecciones de los rodillos en contacto con el suelo, los ejes del vehículo, y las componentes de todas las velocidades influyentes en la movilidad del vehículo. En negro se indica las velocidades totales del vehículo, en verde las velocidades lineales de cada rueda, y en azul, las velocidades de los rodillos en contacto con el suelo.

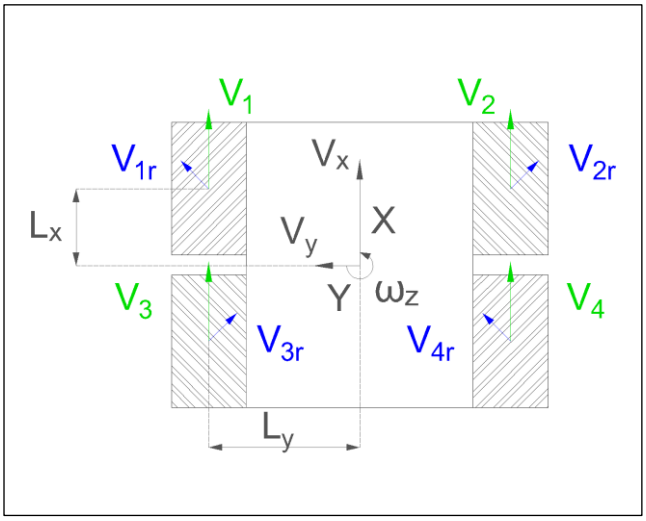


Figura 4: Sistema de coordenadas y velocidades de todos los componentes

V_{ir} (i = 1, 2, 3, 4) es la velocidad tangencial del rodillo que está en contacto con el suelo, V_i (i = 1, 2, 3, 4) e la velocidad lineal correspondiente a las revoluciones de las ruedas, donde $V_i = R \cdot \omega_i$, siendo R el radio de la rueda y ω_i la velocidad angular de la rueda.

V_{irx} y V_{iry} son las componentes de los rodillos que aportan a las velocidades lineales de las ruedas en los ejes X e Y. Estas velocidades son:

$$V_{irx} = V_{ir} \cdot \cos(45º) = V_{ir} \cdot \frac{\sqrt{2}}{2} \quad (1)$$

$$V_{iry} = V_{ir} \cdot \text{sen}(45º) = V_{ir} \cdot \frac{\sqrt{2}}{2} \quad (2)$$

Las velocidades lineales totales de las ruedas, en función del radio y de la velocidad de los rodillos, serán la suma de las velocidades lineales más la aportada por los rodillos:

$$V_{1x} = V_1 + V_{1r} \cdot \frac{\sqrt{2}}{2} = R \cdot \omega_1 + V_{1r} \cdot \frac{\sqrt{2}}{2}, \qquad V_{1y} = V_{1r} \cdot \frac{\sqrt{2}}{2} \quad (3), (4)$$

$$V_{2x} = V_2 + V_{2r} \cdot \frac{\sqrt{2}}{2} = R \cdot \omega_2 + V_{2r} \cdot \frac{\sqrt{2}}{2}, \qquad V_{2y} = V_{2r} \cdot \frac{\sqrt{2}}{2} \quad (5), (6)$$

$$V_{3x} = V_3 + V_{3r} \cdot \frac{\sqrt{2}}{2} = R \cdot \omega_3 + V_{3r} \cdot \frac{\sqrt{2}}{2}, \qquad V_{3y} = V_{3r} \cdot \frac{\sqrt{2}}{2} \quad (7), (8)$$

$$V_{4x} = V_4 + V_{4r} \cdot \frac{\sqrt{2}}{2} = R \cdot \omega_4 + V_{4r} \cdot \frac{\sqrt{2}}{2}, \qquad V_{4y} = V_{4r} \cdot \frac{\sqrt{2}}{2} \qquad (9), (10)$$

Las velocidades de las ruedas también se pueden expresar como la combinación de las velocidades del vehículo:

$$V_{1x} = V_x - L_y \cdot \omega_z, \qquad V_{1y} = V_y + L_x \cdot \omega_z \qquad (11), (12)$$

$$V_{2x} = V_x + L_y \cdot \omega_z, \qquad V_{2y} = V_y + L_x \cdot \omega_z \qquad (13), (14)$$

$$V_{3x} = V_x - L_y \cdot \omega_z, \qquad V_{3y} = V_y - L_x \cdot \omega_z \qquad (15), (16)$$

$$V_{4x} = V_x + L_y \cdot \omega_z, \qquad V_{4y} = V_y - L_x \cdot \omega_z \qquad (17), (18)$$

Combinando las ecuaciones de la (3) a la (18), se obtiene el sistema de ecuaciones:

$$V_1 = V_{1x} - V_{1r} \cdot \frac{\sqrt{2}}{2} = V_{1x} - V_{1y} = V_x - L_y \cdot \omega_z - V_y - L_x \cdot \omega_z = V_x - V_y - \left(L_x + L_y\right) \cdot \omega_z \qquad (19)$$

$$V_2 = V_{2x} - V_{2r} \cdot \frac{\sqrt{2}}{2} = V_{2x} + V_{2y} = V_x + L_y \cdot \omega_z + V_y + L_x \cdot \omega_z = V_x + V_y + \left(L_x + L_y\right) \cdot \omega_z \qquad (20)$$

$$V_3 = V_{3x} - V_{3r} \cdot \frac{\sqrt{2}}{2} = V_{3x} + V_{3y} = V_x - L_y \cdot \omega_z + V_y - L_x \cdot \omega_z = V_x + V_y - \left(L_x + L_y\right) \cdot \omega_z \qquad (21)$$

$$V_4 = V_{4x} - V_{4r} \cdot \frac{\sqrt{2}}{2} = V_{4x} - V_{4y} = V_x + L_y \cdot \omega_z - V_y + L_x \cdot \omega_z = V_x - V_y + \left(L_x + L_y\right) \cdot \omega_z \qquad (22)$$

Representado en forma matricial obtenemos las expresiones de las velocidades de las ruedas en función de las velocidades del vehículo:

$$\begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ V_4 \end{bmatrix} = \begin{bmatrix} 1 & -1 & -\left(L_x + L_y\right) \\ 1 & 1 & \left(L_x + L_y\right) \\ 1 & 1 & -\left(L_x + L_y\right) \\ 1 & -1 & \left(L_x + L_y\right) \end{bmatrix} \cdot \begin{bmatrix} V_x \\ V_y \\ \omega_z \end{bmatrix} \qquad (23)$$

Por otro lado, las velocidades del vehículo pueden ser obtenidas a partir de las velocidades de las ruedas con una matriz pseudo-inversa. Siendo:

$$V_i = \begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ V_4 \end{bmatrix};\ M = \begin{bmatrix} 1 & -1 & -(L_x + L_y) \\ 1 & 1 & (L_x + L_y) \\ 1 & 1 & -(L_x + L_y) \\ 1 & -1 & (L_x + L_y) \end{bmatrix} y\ V_0 = \begin{bmatrix} V_x \\ V_y \\ \omega_z \end{bmatrix} \quad (24)$$

$$V_i = M \cdot V_0 \quad (25)$$

La pseudo-inversa por la izquierda es:

$$M^+ = (M^t \cdot M)^{-1} \cdot M^t \quad (26)$$

Por lo tanto, las ecuaciones en forma matricial de las velocidades del vehículo expresadas según las velocidades de las ruedas son:

$$\begin{bmatrix} V_x \\ V_y \\ \omega_z \end{bmatrix} = \frac{1}{4} \cdot \begin{bmatrix} 1 & 1 & 1 & 1 \\ -1 & 1 & 1 & -1 \\ -\dfrac{1}{(L_x + L_y)(L_x + L_y)} & \dfrac{1}{(L_x + L_y)(L_x + L_y)} & \dfrac{1}{(L_x + L_y)(L_x + L_y)} & -\dfrac{1}{(L_x + L_y)(L_x + L_y)} \end{bmatrix} \cdot \begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ V_4 \end{bmatrix} \quad (27)$$

Y expresadas en función de las velocidades angulares, teniendo todas las ruedas el mismo radio:

$$\begin{bmatrix} V_x \\ V_y \\ \omega_z \end{bmatrix} = \frac{1}{4} \cdot \begin{bmatrix} 1 & 1 & 1 & 1 \\ -1 & 1 & 1 & -1 \\ -\dfrac{1}{(L_x + L_y)(L_x + L_y)} & \dfrac{1}{(L_x + L_y)(L_x + L_y)} & \dfrac{1}{(L_x + L_y)(L_x + L_y)} & -\dfrac{1}{(L_x + L_y)(L_x + L_y)} \end{bmatrix} \cdot R \cdot \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix} \quad (28)$$

Simulación

Primero se ha procedido a realizar una simulación del robot y de su funcionamiento mediante MATLAB y Simulink, utilizando la librería de Simscape, que permite simular movimientos e interacciones de los objetos. Para ello, se ha utilizado el siguiente esquema de Simulink:

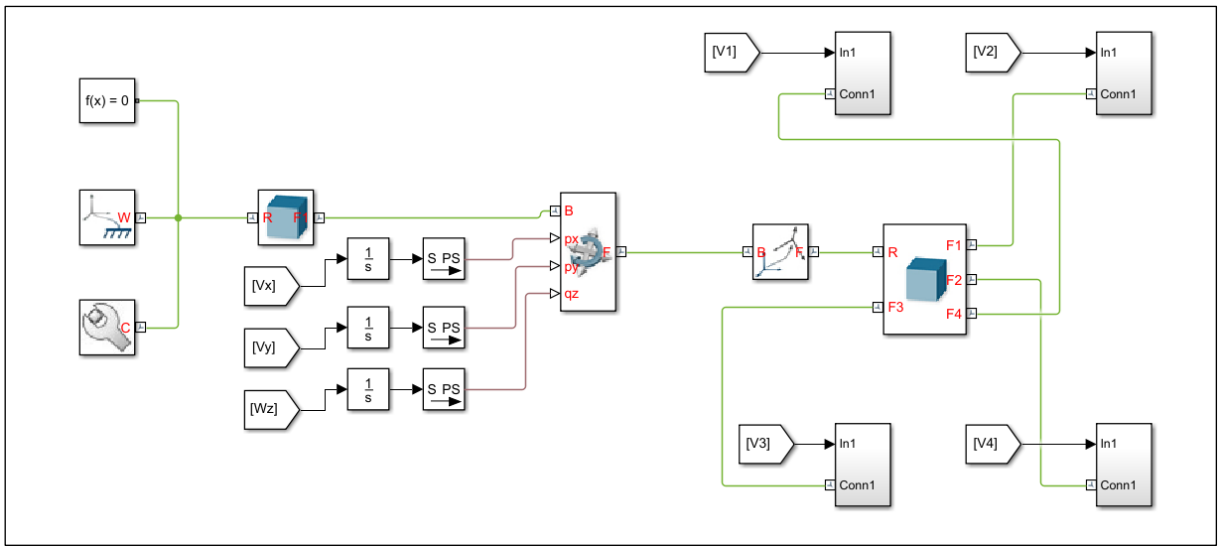


Figura 5: Esquema de Simulink para la simulación del movimiento del robot

Al sistema se introducen las velocidades de las ruedas y las del vehículo calculadas con las matrices del estudio teórico. En cada subsistema de las ruedas, aquellos conectados con las articulaciones F1, F2, F3 y F4, está el siguiente esquema para simular el movimiento de las ruedas:

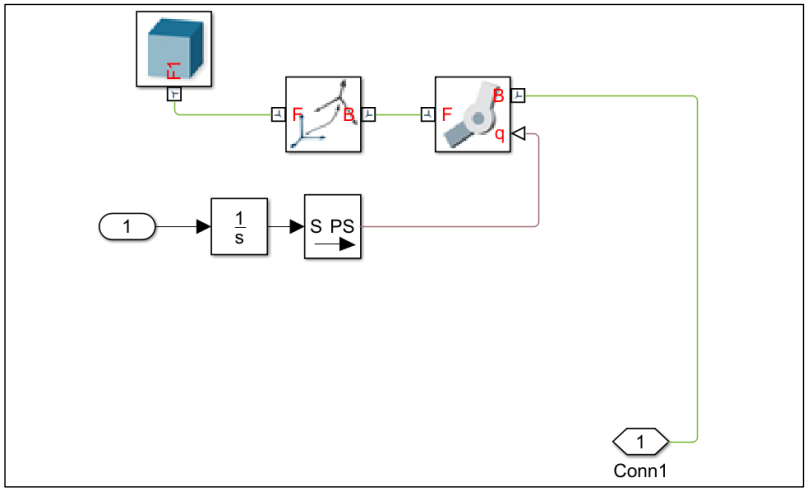


Figura 6: Esquema Simulink de las ruedas del robot.

Se han realizado diversas simulaciones, unas en donde se debía calcular las velocidades del vehículo y otras donde se debía calcular las velocidades de las ruedas. La primera, introduciendo velocidades constantes o conocidas en las ruedas, se calcula la velocidad del vehículo para realizar trayectorias. Se introduce una secuencia para que la trayectoria sea hacia delante, atrás, izquierda, derechas, giro antihorario y giro horario:

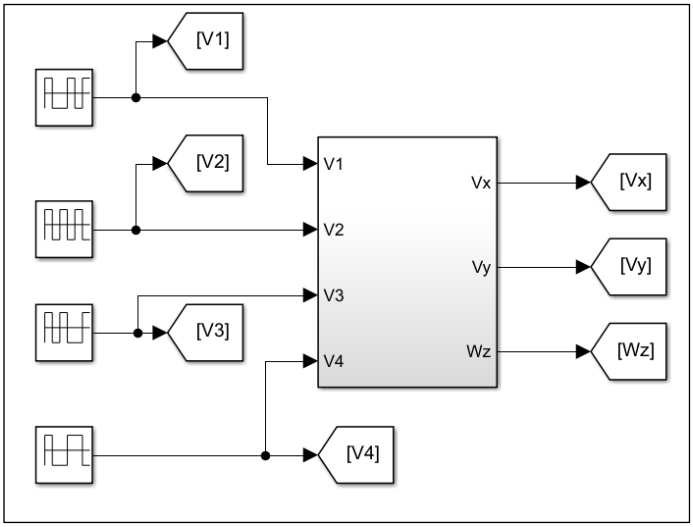


Figura 7: Diagrama Simulink Simulación 1

En la segunda simulación se enviaba a través de un Smartphone, con la información del sensor del giroscopio, a través de una comunicación Wi-Fi con el ordenador, las velocidades del robot. Los valores de los sensores se disminuyen para evitar grandes aceleraciones. Se utiliza el bloque "*Real-Time Sync*" para que la simulación se ejecute en tiempo real. El valor de *Pitch* (o cabeceo) domina la velocidad en el eje X (delante y detrás), el valor de *Roll* (alabeo) controla la velocidad en el eje Y, y el valor de *Yawn* (guiñada) gobierna el movimiento rotativo respecto al eje central del robot.

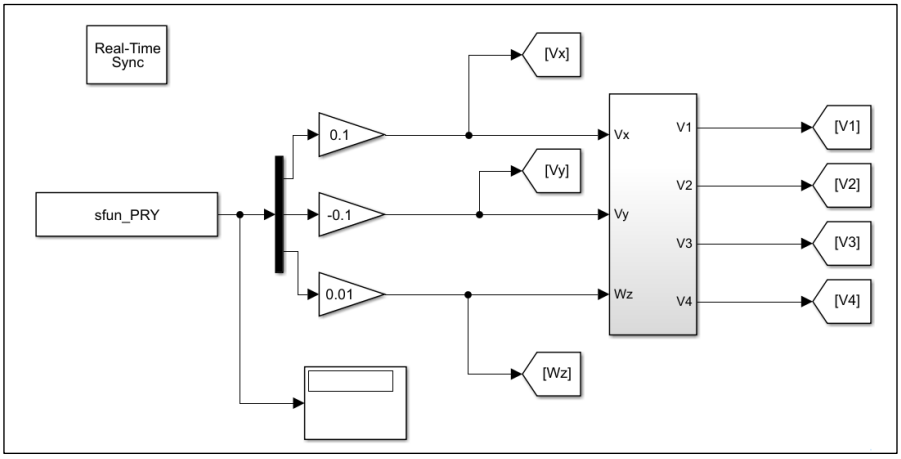


Figura 8: Diagrama Simulink Simulación 2

El resultado de ambas simulaciones se comprueba en la ventana de simulación, obteniendo resultados satisfactorios.

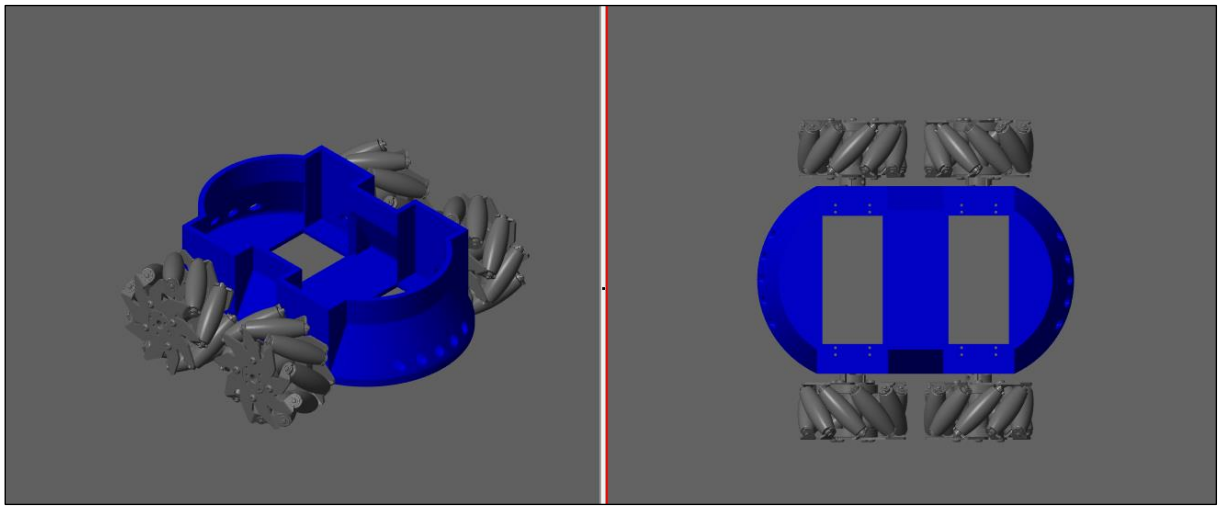


Figura 9: Entorno de la simulación

Diseño del Robot Móvil Omnidireccional usado en las pruebas

Visto el resultado en las simulaciones, se procede al montaje del robot para el estudio del movimiento de este. Este modelo es sólo funcional y su propósito es comprobar el objetivo principal del estudio. No es un prototipo pensado para la industria o comercialización. Se ha tenido en cuenta para el diseño del robot su resistencia, funcionalidad, equilibrio, autonomía, capacidad de evolución, fiabilidad, adaptabilidad y capacidad para responder.

En este apartado se mostrará una imagen representativa del elemento a describir y sus características. Las dimensiones y montajes se encuentran en el apartado *Planos*.

Chasis

La *Figura 10* muestra el diseño del chasis utilizado, donde se incluirá en su interior los motores y demás componentes, y en el exterior se acoplarán las ruedas omnidireccionales. El montaje total del proyecto se encuentra en el *Plano 01,* y el del chasis con sus dimensiones en el *Pla*no 02:

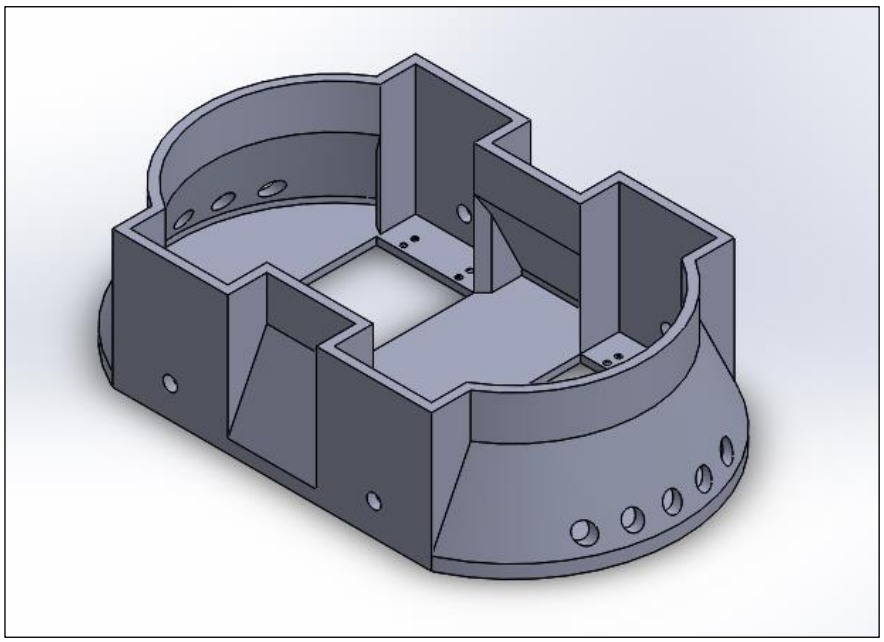


Figura 10: Diseño del chasis

El chasis tiene las siguientes características:

- El material utilizado es plástico, fabricada con una impresora 3D.
- Sus dimensiones son 288.79x171x77.5 mm.

Motores

La *Figura 11* muestra uno de los 4 motores utilizados, los Metal gearmotor 37D de Pololu. Estos motores han sido escogidos debido a que la fuerza que proporcionan es suficiente para mover las ruedas a las velocidades necesarias para el estudio. Además, es necesario acceder al encoder para el bucle de control, pues accediendo a la información de estos se calcula la velocidad de giro y el cálculo de la acción de control necesaria para suplir el error. Los motores se alimentan a una tensión de 12 V DC e integran una reductora de 70:1 Los encoders disponen de una resolución de 64 cuentas por vuelta que, teniendo en cuenta la caja de cambios, se traducen en 4480 cuentas por revolución Es decir, se dispone de una resolución de 360/4480 = 0.08º. Las dimensiones del motor se encuentran en el *Plano 03*.

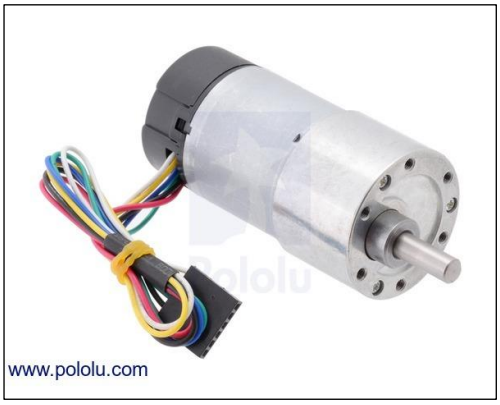


Figura 11: Motor Metal gearmotor 37D

Las características del motor son:

- Dimensiones: 37D x 70L mm.
- Peso: 225 g.
- Tensión nominal de los encoders: 5V.
- Velocidad sin carga: 200 rpm.
- Consumo nominal: 5000 mA.
- Consumo sin carga: 300 mA.
- Fuerza: 17 oz-inch.

Los dos encoders del motor generan una onda cuadrada, de tensión proporcional a la tensión de alimentación a los encoders, desfasadas 90 grados. Cuando el motor gira, el contador se desplaza a través de esta onda cuadrada. El microcontrolador se activa en los cambios del encoder, a nivel bajo y nivel alto, y en función de las posiciones relativas de los encoders, calcula la velocidad de los motores. La siguiente tabla describe las salidas y entradas del motor por color:

CABLE	DESCRIPCIÓN
Rojo	Alimentación del motor (conecta con un terminal)
Negro	Alimentación del motor (conecta con otro terminal)
Verde	GND del encoder
Azul	Alimentación del encoder (3.5 – 20 V)

Amarillo	Salida del encoder A
Blanco	Salida del encoder B

Ruedas

Figura 12: Ruedas Mecanum 100 mm

La *Figura 12* muestra las ruedas Mecanum de Makeblock utilizadas en el montaje. La elección de estas ruedas viene determinada por las características explicadas en el apartado del estudio teórico del tipo *Mecanum*, además, de que el peso que soportan estas ruedas es suficiente para soportar al robot entero. El tamaño es apropiado y cómodo para el estudio de la velocidad, tanto de las ruedas como del vehículo. El material del que están hechos proporciona una rigidez y resistencia apropiada, suficiente para aguantar golpes y el desgaste para asegurar su durabilidad. El material de los rodillos proporciona un buen agarre sobre el suelo, pues este agarre es clave para el buen funcionamiento de las velocidades pasivas que otorgan el tercer grado de libertad, pues si los rodillos patinaran sobre el suelo, las ecuaciones del modelo no servirían y se perdería el carácter omnidireccional. Sus dimensiones son especificadas en el *Plano 04*.

Las especificaciones de las ruedas son:

- 100D x 56.4 mm.
- Material: Aluminio (2 placas de SPCC).
- Peso: 2150 g.
- Rodillos: 9 rodillos de goma.

Arduino DUE

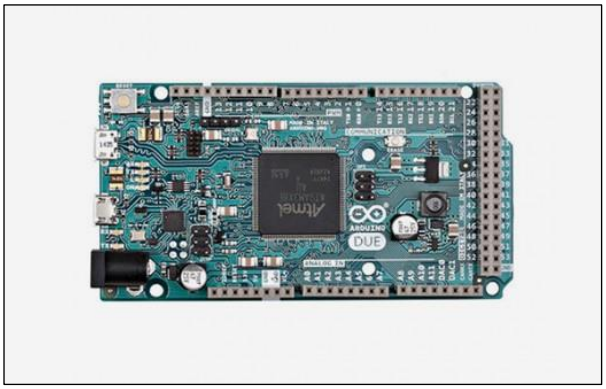


Figura 13: Tarjeta Arduino DUE

La tarjeta Arduino DUE es la primera tarjeta basada en un microcontrolador de 32 bits. Contiene una numerosa cantidad de pines digitales (54) y analógicos (12). Debido a esto, a la gran velocidad de procesamiento y a que todos los pines digitales son aptos para realizar interrupciones, se escoge esta tarjeta para controlar a los motores. El software requerido para poder programarlo es totalmente gratuito y se puede encontrar en la página oficial de Arduino.

Las especificaciones de la tarjeta se encuentran resumidas en la siguiente tabla:

Microcontrolador	**AT91SAM3X8E**
Tensión nominal	3.3 V
Voltaje de entrada (recomendado)	7 – 12 V
Voltaje de entrada (límites)	6 – 16 V
Pines Digitales E/S	54 (de los cuales 12 proveen de salida PWM)
Pines de Entrada Analógico	12
Pines de Salida Analógicos	2 (DAC)
Corriente DC de salida total en todas las líneas de E/S	130 mA
Corriente DC para pines de 3.3V	800 mA
Corriente DC para pines de 5V	800 mA
Memoria Flash	512 KB toda disponible para aplicaciones de usuario
SRAM	96 KB (dos bandas de 64 KB y 32 KB)
Velocidad de reloj	84 MHz
Longitud	101.52 mm

Anchura	53.3 mm
Peso	36 g

Motor Shield V1

La *Figura 14* muestra el componente utilizado para gobernar los 4 motores de DC. El Motor Shield V1 de Adafruit permite controlar 4 motores de continua, además de 2 servos (que no utilizamos). No hay ningún PIN de Arduino conectados directamente a los motores, ya que la gestión de estos se realiza a través del Shift Register (74HC595). El Shield proporciona un control de las corrientes para gobernar los motores más preciso que el Arduino, ya que este no es capaz de proporcionar esas intensidades.

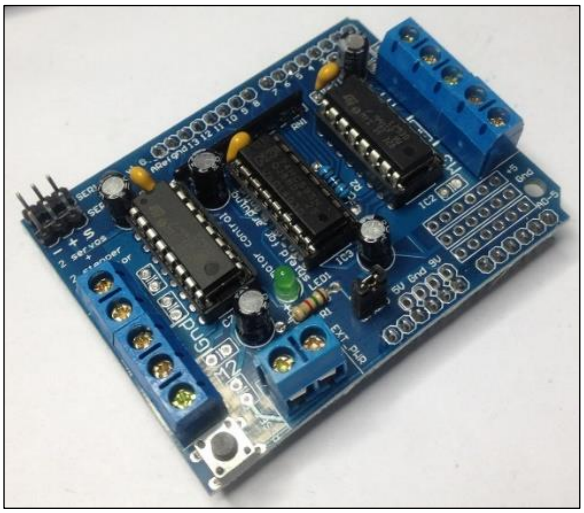


Figura 14: Adafruit Motor Shield V1

Sus especificaciones son:

- 4 H Bridges incluidos con dos L293D chips.
- Hasta 4 motores CC con control bidireccional y selección de velocidad de 8 bits.
- Máxima corriente de 0.6 A.
- Acepta motores alimentados entre 4.5 – 25 V.
- Disponible alimentación separada para motores para evitar ruido e interferencias.

Batería

La batería mostrada en la *Figura 15*, de litio recargable, ha sido la encargada de suplir de energía al procesador y al Shield para mover los motores, además de proporcionar la alimentación de cada uno de los componentes del robot.

Figura 15: Batería de litio de 7.4 V

Especificaciones de la batería:

- Capacidad de Voltaje: 7.4 V.
- Capacidad de corriente: 6000 mAh.
- Peso: 200 g.

Cables y led

Para conectar los motores se han utilizado tanto cables Macho-Macho y Macho-Hembra. Se ha utilizado un led azul de modo indicador de diversas funciones.

Figura 16: Cables Macho-Macho, Macho-Hembra y led Azul

Módulo Bluetooth HC-06

Para la comunicación se ha empleado el módulo HC-06 para comunicar por vía Bluetooth con cualquier terminal compatible.

Figura 17: Módulo Bluetooth HC-06

Las especificaciones del HC-06 son:

- Tensión de alimentación: 5 V.
- Tensión funcionamiento: 3.3 V.

Control

Al igual que en la simulación, se ha procedido a dos tipos de controles. Uno, un control de trayectoria, indicando las velocidades en los dos ejes, y otro utilizando el giroscopio de un Smartphone para controlar las velocidades, tanto lineales como angular del vehículo. Todos los códigos utilizados y Esquemas de Simulink se detallan en el apartado *Anexos*.

Para mejorar el control de las trayectorias del robot se ha diseñado un control PI mediante el método de identificación experimental. Al inicio, con una acción integral nula, se aumentaba el valor de la acción proporcional hasta obtener unas acciones de control cercanas a la saturación. El criterio establecido es llegar alrededor de 180-200 bits de acción de control.

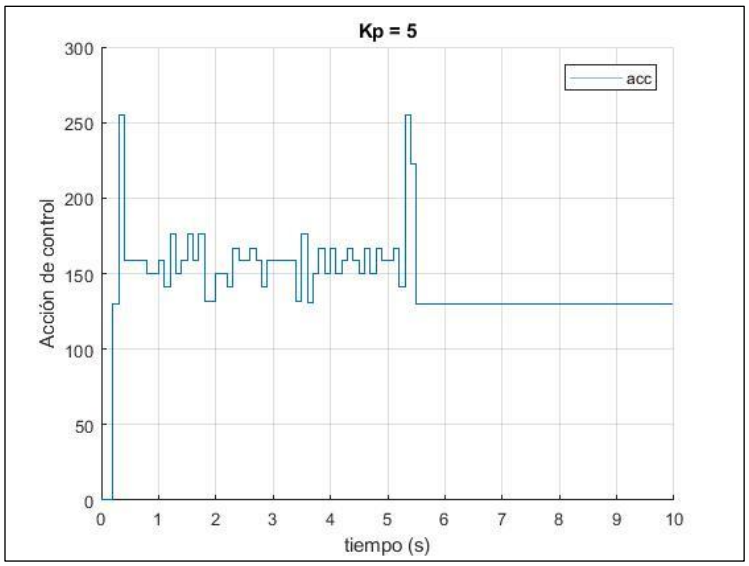


Figura 18: Acciones de control ante escalón con Kp = 5

Se emplea una referencia de tipo escalón que dura 5 segundos para visualizar las distintas acciones de control. En la *Figura 18*, se usa una Kp pequeña y se observa que las acciones de control rondan alrededor de 150 bits. Los dos picos se producen en los cambios de referencia.

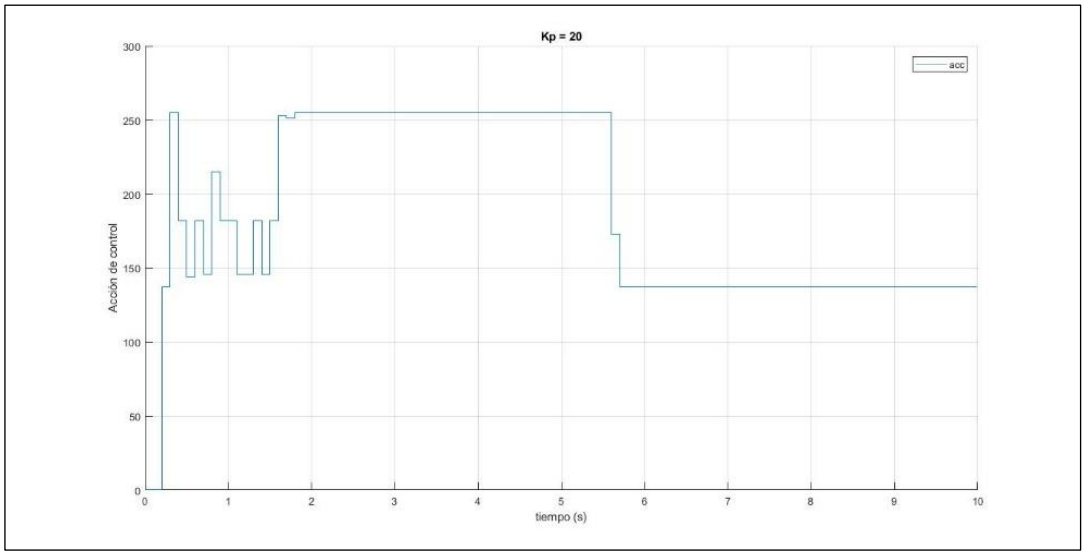


Figura 19: Acciones de control ante escalón con Kp = 20

Aumentando mucho la acción proporcional, las acciones de control saturan al máximo (255), por lo que es necesario reducir la constante proporcional para dejar margen a la acción integral. En la *Figura 19* se muestra la saturación de la acción de control.

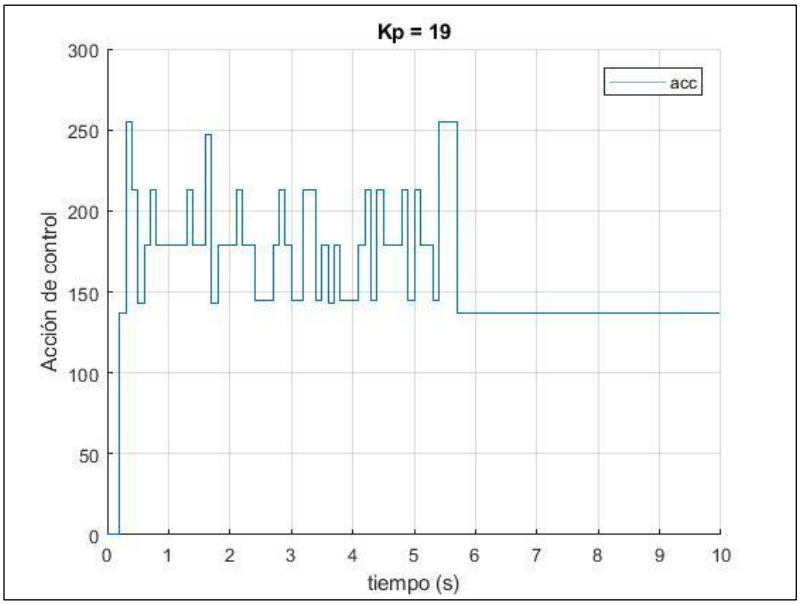


Figura 20: Acciones de control ante escalón con Kp = 19

Con una constante de 19 se consiguen la máxima acción integral sin que sature demasiado, como se observa en la gráfica de la *Figura 20*. Se decide emplear esta constante para el control PI, incluso se puede reducir un poco en caso de que la acción integral sature las acciones de control.

Una vez determinada la acción proporcional, se aumenta la constante integral observando el comportamiento en las velocidades de las ruedas, observando que se anule el error lo más rápido posible sin pasarse (significa que la acción de control es máxima, y por tanto saturada).

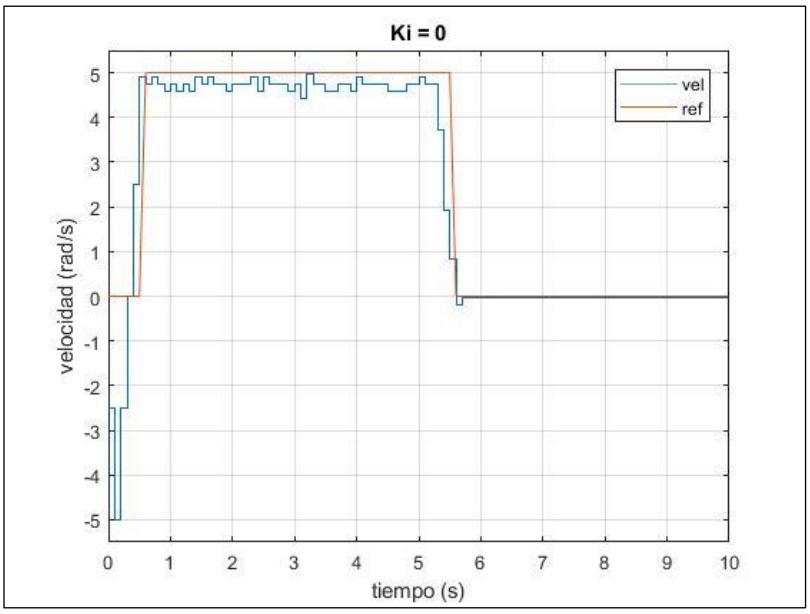


Figura 21: Velocidad de las ruedas ante escalón con Ki = 0

Con una constante de integración nula, el error es muy pequeño, pero se decide anularlo lo máximo posible y que anule perturbaciones, como irregularidades del suelo o rozamientos que influyan en la velocidad de las ruedas.

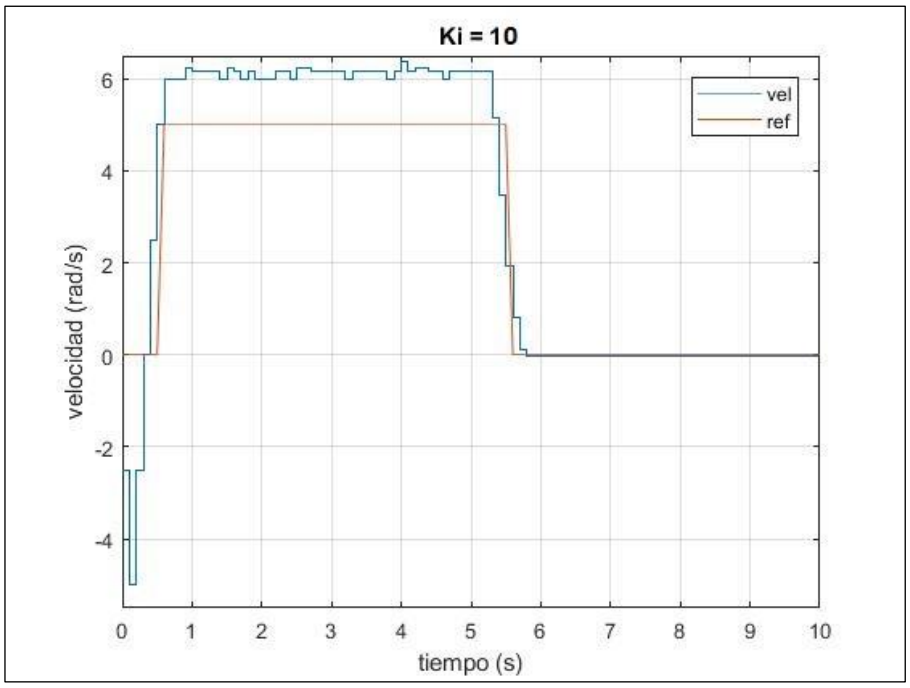


Figura 22: Velocidad de las ruedas ante escalón con Ki = 10

Con una constante de integración muy alta las acciones de control saturan y la velocidad de las ruedas es mayor a la referencia, aumentando el error, como se observa en la gráfica de la *Figura 22*.

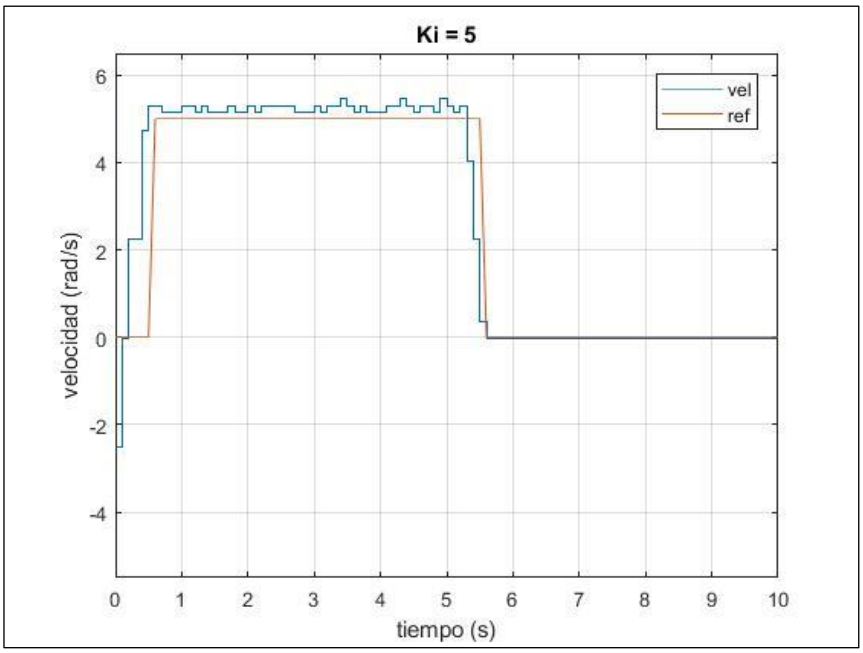


Figura 23: Velocidad de las ruedas ante escalón con Ki = 5

La mejor respuesta conseguida es con una constante de integración de 5, aunque la velocidad sea ligeramente superior a la referencia, pero esto es debido a los redondeos al transformar velocidades y acciones de control a bits y a rad/s. En al *Figura 23* se muestra el seguimiento de la referencia de 5 rad/s.

Aplicación Android

Para el manejo del robot omnidireccional se ha diseñado y creado, con la ayuda de una aplicación Android llamada Bluetooth Electronicks ©. Para el diseño de esta aplicación se plantea dos mandos de control: el de la derecha, con ocho direcciones, se encarga de controlar la velocidad de las ruedas para que se mueva en la dirección indicada, con la orientación fija. El mando de la izquierda solo tiene habilitado los pads de arriba y abajo, encargados de gobernar la orientación del robot. Si se pulsa el botón de arriba gira en el sentido de las agujas del reloj, y si se pulsa el de abajo, en sentido contrario.

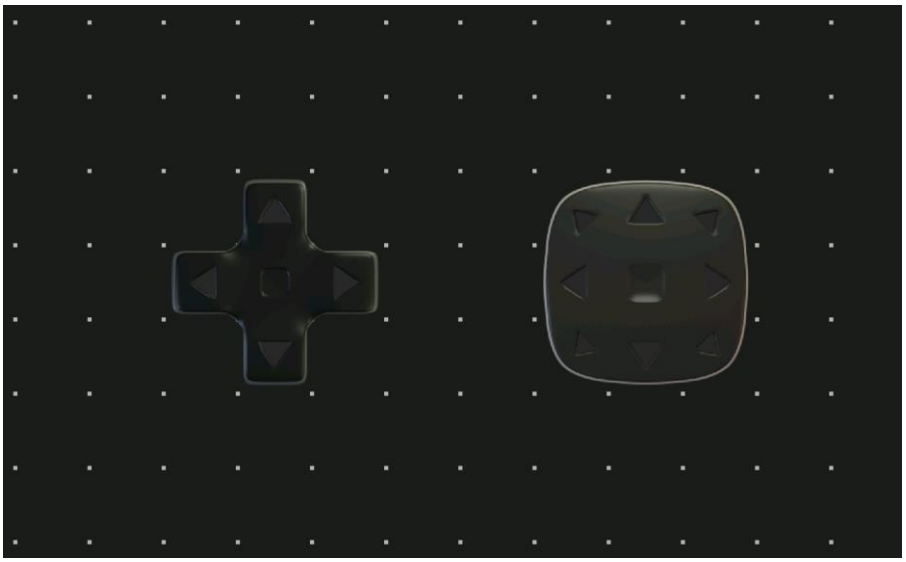


Figura 24: Mando de control de la aplicación Android para el manejo del robot

Resultados

Utilizando los Esquemas de Simulink y códigos de Arduino del apartado *Anexos*, se observa el comportamiento y funcionamiento del robot en el plano. Para la representación de los resultados se ha preestablecido una secuencia de velocidades para trazar diversas trayectorias y ver su movimiento. El objetivo del proyecto no es un control de trayectorias, por lo que no existe un control de estas ni se espera que los resultados sean favorables a un seguimiento de trayectorias.

La primera trayectoria fijada es un recorrido cuadrado donde la orientación es fija.

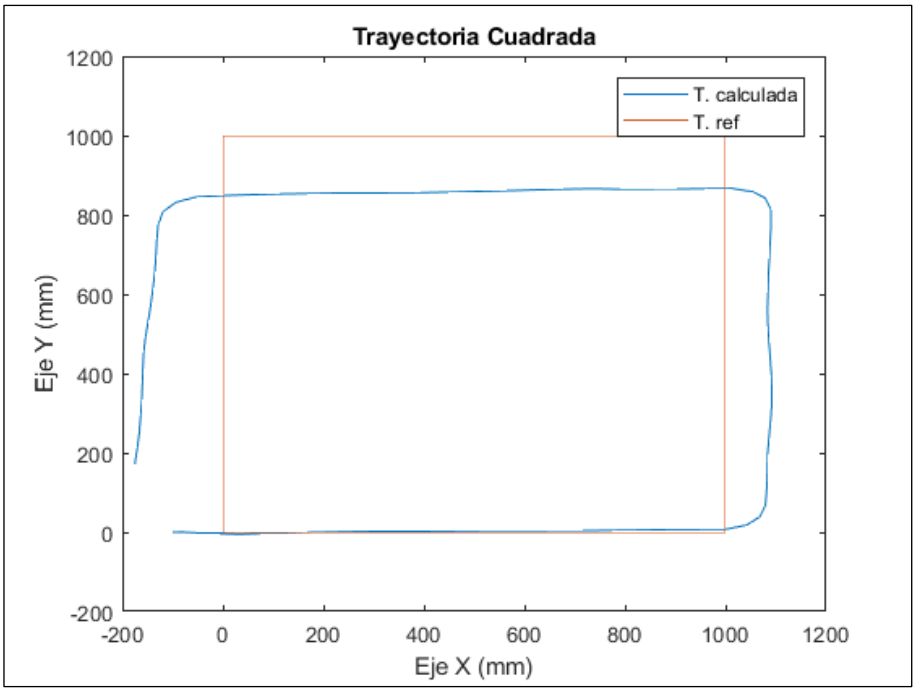


Figura 25: Trayectoria cuadrada

En la *Figura 24* se muestran la trayectoria de referencia en rojo y en azul la trayectoria que el robot piensa que está haciendo, calculada mediante las velocidades de las ruedas. Se observa que el movimiento lateral es algo más corto de lo que debería. En la siguiente imagen se muestra el recorrido del robot grabado con una cámara desde arriba. El cuadrado blanco es la referencia que debe seguir.

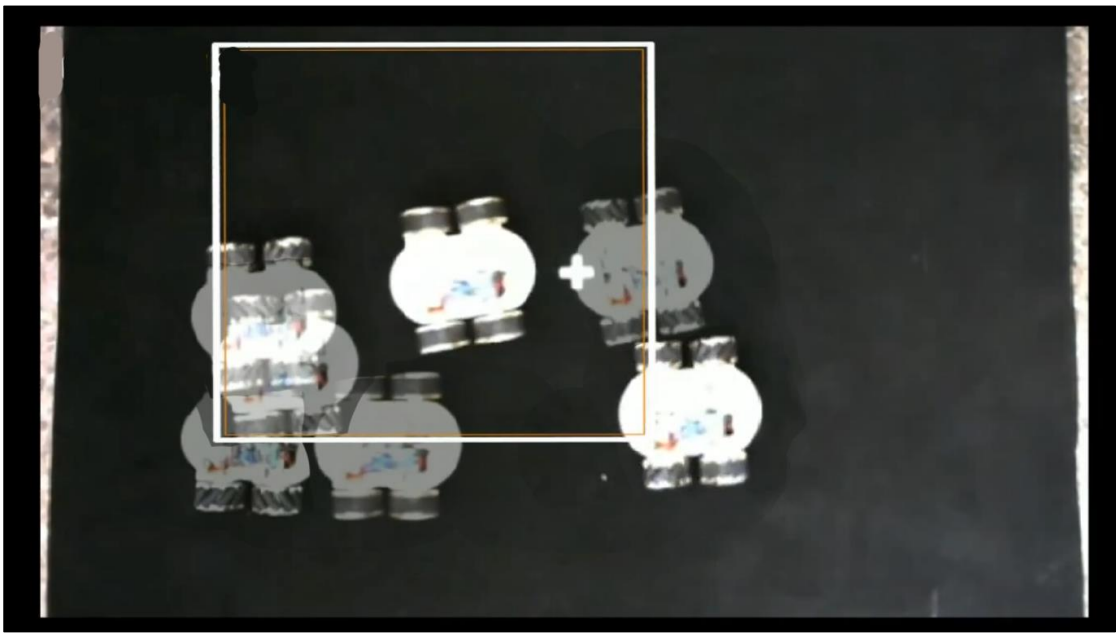


Figura 26: Trayectoria cuadrada real del robot

La siguiente trayectoria es una circular, manteniendo la orientación fija. Esta trayectoria se ha generado con la suma de dos senoidales en los dos ejes con una frecuencia de 1 rad/s, con un periodo de muestreo de 100 ms. Como se observa en la *Figura 26*, el seguimiento es muy pobre. Esto es debido a que los cambios tan rápidos en la referencia no son capaces de comunicarlos a través de la comunicación ni de alcanzar dicha referencia en tan poco tiempo.

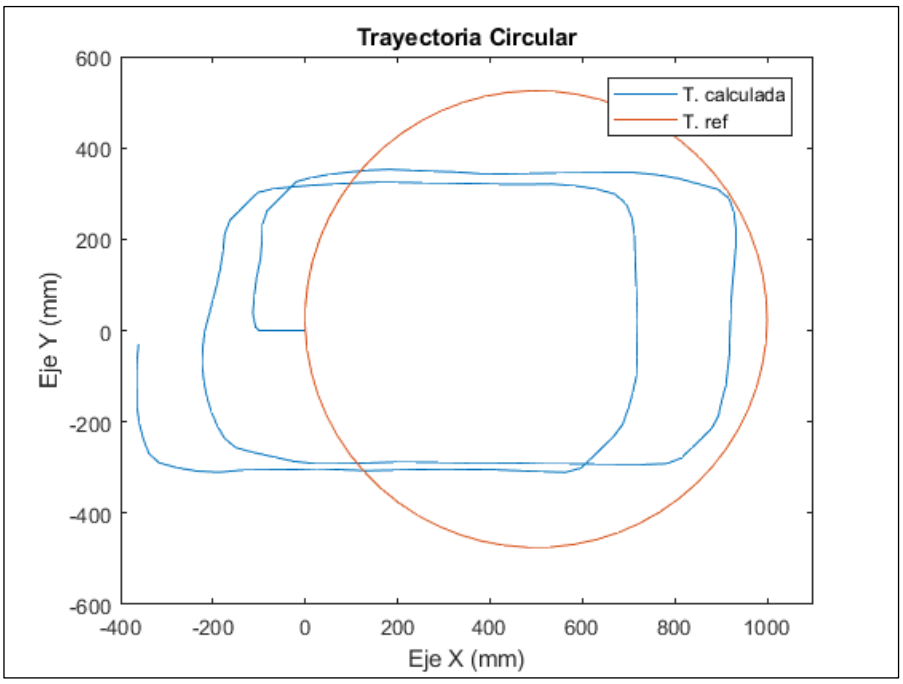


Figura 27: Trayectoria circular

En la siguiente imagen se muestra la trayectoria real realizada por el vehículo. La línea azul indica la referencia que debería hacer y se ha marcado en rojo la que de verdad a seguido.

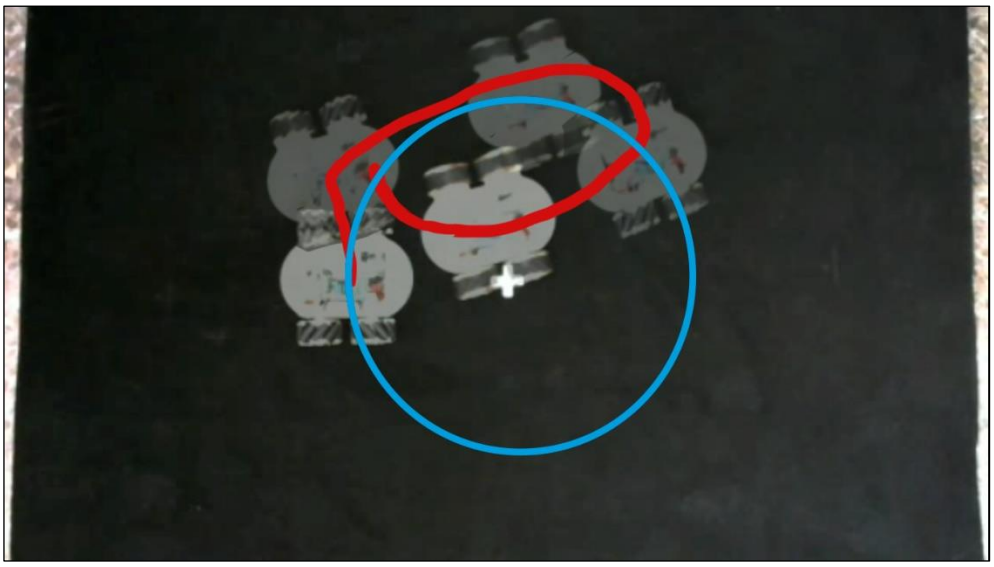


Figura 28: Trayectoria circular real del robot

La última trayectoria realizada es una curva de Lissajous, realizada con dos senoidales cuya relación de frecuencias es 3:1. A igual que en la trayectoria circular, los cambios tan rápidos en la referencia de las senoidales no son alcanzables y la trayectoria no sigue la referencia.

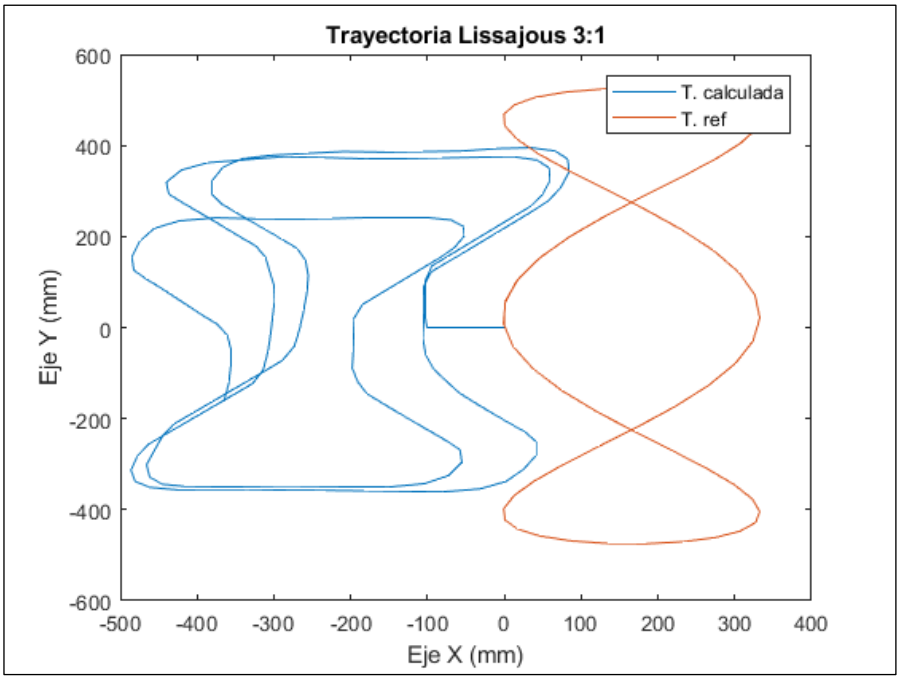


Figura 29: Trayectoria Lissajous 3:1

La *Figura 29* muestra la trayectoria real seguida por el vehículo con la misma referencia. Esta es la trayectoria que más a costado de seguir y peores resultados se observan.

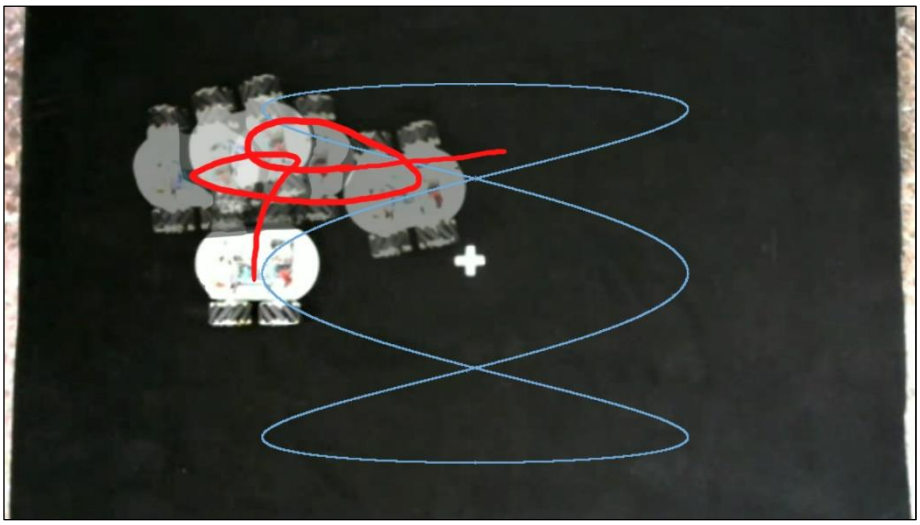


Figura 30: Trayectoria curva Lissajous real

Conclusiones y propuestas de mejora

En cuanto a las conclusiones del trabajo, podemos afirmar que la cinemática estudiada para este robot con ruedas Mecanum es funcional. Se ha comprobado, tanto a nivel teórico como practico, la capacidad omnidireccional del robot aportada por las ruedas utilizadas, ya que se ha podido comprobar individualmente en cada grado de libertad en el plano, de manera independiente, en todas las pruebas.

El diseño utilizado para el robot ha sido considerado que sea lo más funcional posible, compacto y fiable. Este diseño a sido adecuado, pues protege y contiene todos los elementos en su interior para evitar desperfectos, y las llantas se colocan de forma correcta para producir un movimiento uniforme en su desplazamiento. Los problemas encontrados ha sido la falta de peso, pues al construir la carcasa del chasis con plástico, el robot era muy ligero, sumando a las pequeñas imprecisiones a la hora de montar las ruedas y que no estuviesen perfectamente niveladas, el rozamiento con el suelo no era igual en todos los rodillos, provocando diferencias en los desplazamientos frontales y laterales. Se ha intentado suplir este desperfecto añadiendo peso con dos pesas de 2 kg cada una, mejorando ligeramente los resultados, pero notando un descenso notario en la velocidad de desplazamiento.

Respecto al control, el seguimiento de trayectorias con cambios constantes o muy rápidos se hace complicado o es ineficaz, a velocidades constantes. Las velocidades pequeñas son casi imposibles de realizar debido a las limitaciones de los motores, debido a la zona muerta de estos y al ruido de los encoders de baja calidad que provocan cambios bruscos en el error de la referencia, por lo que trayectorias pequeñas son casi imposible de generar. La comunicación utilizada entre el generador de trayectorias, un ordenador, y el controlador del robot (la tarjeta Arduino DUE) se realiza a través del módulo bluetooth que funciona a una velocidad de 100 ms, y para un buen control de la velocidad de los motores, el bucle de control se realiza cada 10 ms. La comunicación se hace 10 veces más lenta que el control de las velocidades, por lo que cambios rápidos en la referencia provocan pérdidas de información y realizan trayectorias no deseadas. Como se ha mencionado, el objetivo del trabajo no es realizar un seguimiento de trayectoria, si no comprobar y estudiar el tercer grado de libertad aportado por las ruedas *Mecanum*.

La utilización de un dispositivo con Bluetooth ha permitido utilizar los sensores del giroscopio y acelerómetro para gobernar las velocidades de las ruedas, sin necesidad de un ordenador o un software complicado. Los problemas de comunicación son los mismos que usando un ordenador, ya que el limitador es la comunicación inalámbrica. Otro factor limitante es la distancia máxima de comunicación, siendo esta, aproximadamente, de 10 metros.

Propuestas de mejora

El robot utilizado es un prototipo válido para el estudio de la validez de la cinemática desarrollada para este tipo de ruedas. Existen aspectos en el diseño no contemplados a detalle, ya que el principal objetivo es el estudio del tipo de ruedas y el añadido grado de libertad que otorga la omnidireccionalidad. En trabajos futuros se deben hacer mejoras en la parte mecánica como en la de control para su desarrollo.

Un sistema de suspensión para las ruedas. El robot utilizado no posee ningún sistema de suspensión. Es conocido que una buena amortiguación en las ruedas permite absorber las irregularidades del terreno por la que se desplaza un vehículo. Aumentando el peso del vehículo y con una buena suspensión, los errores producidos en las pruebas pueden ser reducidos notablemente, siempre y cuando los motores sigan teniendo la suficiente fuerza para mover el vehículo.

En la mayoría de las pruebas la velocidad de las ruedas era constante, por lo que el seguimiento de trayectorias era muy complicado. Un cambio tan rápido o tan continuado puede provocar la saturación de las velocidades, por lo que una de las mejoras que se pueden realizar es un control de trayectorias con velocidad modulable, es decir, cuando la dirección cambie muy bruscamente, que produzca un cambio drástico de las velocidades de las ruedas, esta cambie gradualmente para evitar saturaciones. También hay que considerar el tamaño de la trayectoria, que no sea muy pequeño, y realizar el control por posición, no desde las velocidades directamente. Para un control del seguimiento de trayectorias hay que considerar un bucle de control externo al diseñado, con un sensor externo, mediante una cámara, por ejemplo, capaz de corregir la posición del robot para un seguimiento óptimo.

Otra implementación es considerar un sistema de comunicación diferente a la empleada en este proyecto. Una comunicación utilizando ondas de radio permiten una comunicación más rápida y de mayor alcance, supliendo el problema del tamaño de las trayectorias y de la velocidad de comunicación, para hacer un control a tiempo real de las trayectorias.

Presupuesto

Introducción

A continuación, se presenta el presupuesto estimado para el proyecto realizado. En él, se incluye tanto los costes relacionados con los equipos y software empleados como también los costes de personal e instalaciones.

En primer lugar, se realizará el conteo de los recursos utilizados para la realización del estudio. Se mostrará el desglose de costes, especificando el porcentaje del coste global de los recursos correspondiente. Es decir, se calculará que porcentaje de la vida útil del elemento en cuestión se ha empleado en el trabajo y su valor económico.

Con relación a los costes de personal e instalación, el trabajo se engloba en un proyecto de fin de grado de la Universidad Politécnica de València. Por este motivo, se consideran nulos las aportaciones económicas tanto al gasto de personal como de las instalaciones, pues se ha utilizado la sala multiusos del Departamento de Sistemas y Automática de la UPV.

Conteo de los recursos

Hardware

Descripción	Unidades
ACER E15	1
IMPRESIÓN 3D DE LA CARCASA	1
Metal Gearmotor 37DX70L mm witch 64 CPR encoder	4
Pack 4 ruedas Mecanum 100 mm	1
Arduino Due CPU 32-bits AT91SAM3X8E	1
Motor Shield V1 de Adafruit	1
Bateria Litio 7.4V 2600mAh 15C	1
Modulo Bluetooth HC-06	1

Software

Solo será contabilizado aquellas licencias de pago, así pues, no aparecerán en la lista aquellas aplicaciones de licencia libre o gratuitas.

Descripción	Unidades
Licencia Matlab R2017b	1

Personal

Descripción	Unidades
Estudiante de Grado	1

Instalaciones

Descripción	Unidades
Laboratorio de Investigación	1

Desglose de costes

Hardware

Destacar que para el coste del trabajo se ha considerado el precio total de los elementos utilizados en el montaje del robot. Sin embargo, para el caso del ordenador utilizado, solo se considera la vida útil empleada durante la realización del trabajo. Es decir, 4 meses de 60 meses de vida útil.

Descripción	**Coste Unitario**	**Coste Total (IVA incluido)**	**Coste Proyecto**
ACER E15 AMD A10	376.52 €	455.59 €	30.37 €
IMPRESIÓN 3D DE LA CARCASA	150.21 €	181.76 €	181.76 €
Metal Gearmotor 37DX70L mm witch 64 CPR encoder	38.95 €	188.52 €	188.52 €
Pack 4 ruedas Mecanum 100 mm	80.00 €	96.80 €	96.80 €
Arduino Due CPU 32-bits AT91SAM3X8E	39.75 €	48.10 €	48.10 €
Motor Shield V1 de Adafruit	9.00 €	10.89 €	10.89 €
Bateria Litio 7.4V 2600mAh 15C	16.53 €	20.00 €	20.00 €
Modulo Bluetooth HC-06	6.60 €	7.99 €	7.99 €
	TOTAL	1009.65 €	584.43 €

El coste total correspondiente al *hardware* es de QUINIENTOS OCHENTA Y CUATRO EUROS Y CUARENTA Y TRES CÉNTIMOS.

Licencias de Software

Para el cálculo de las licencias se ha seguido el mismo procedimiento que con el cálculo del presupuesto del ordenador, contabilizando únicamente el coste proporcional a las horas empleadas. Para ello, se ha obtenido el número de horas de trabajo empelados en este proyecto.

Horas de trabajo anuales = 8 horas · 5 días · 4 semanas · 11 meses = 1760 h

Horas de trabajo empleadas = 4 horas · 5 días · 4 semanas · 4 meses = 320 h

Coste del Proyecto = Coste Total · 320/1760

Descripción	**Coste Unitario**	**Coste Total (IVA incluido)**	**Coste Proyecto**
Licencia Matlab R2017b	661.16 €	800.00 €	145.45 €
	TOTAL	800.00 €	145.45 €

El coste total correspondiente al *software* es de CIENTO CUARENTA Y CINCO EUROS Y CUARENTA Y CINCO CÉNTIMOS.

Personal e Instalaciones

Como se ha mencionado, el trabajo se enmarca en un proyecto de final de grado, por lo que no existen retribuciones económicas al personal ni existen gastos correspondientes al uso de instalaciones, ya que se ha trabajado en una de las salas del Departamento de Ingeniería de Sistemas y Automática de la Universitat Politécnica de València.

El coste total correspondiente al personal y a las instalaciones es de CERO EUROS.

Resumen del presupuesto

CONCEPTO	Coste
Costes totales Hardware	584.43 €
Costes totales Software	145.45 €
Costes totales Personal	0.00 €
Costes totales Instalaciones	0.00 €
TOTAL	**729.88 €**

El coste total del proyecto es de SETECIENTOS VEINTINUEVE EUROS Y OCHENTA Y OCHO CÉNTIMOS.

Anexos

En las siguientes páginas se incluyen los Esquemas de Simulink y Códigos de Arduino para las pruebas realizadas, así como las aplicaciones realizadas para Bluetooth Electronics.

Anexo 1: Diagrama de Simulink para el control de trayectorias.

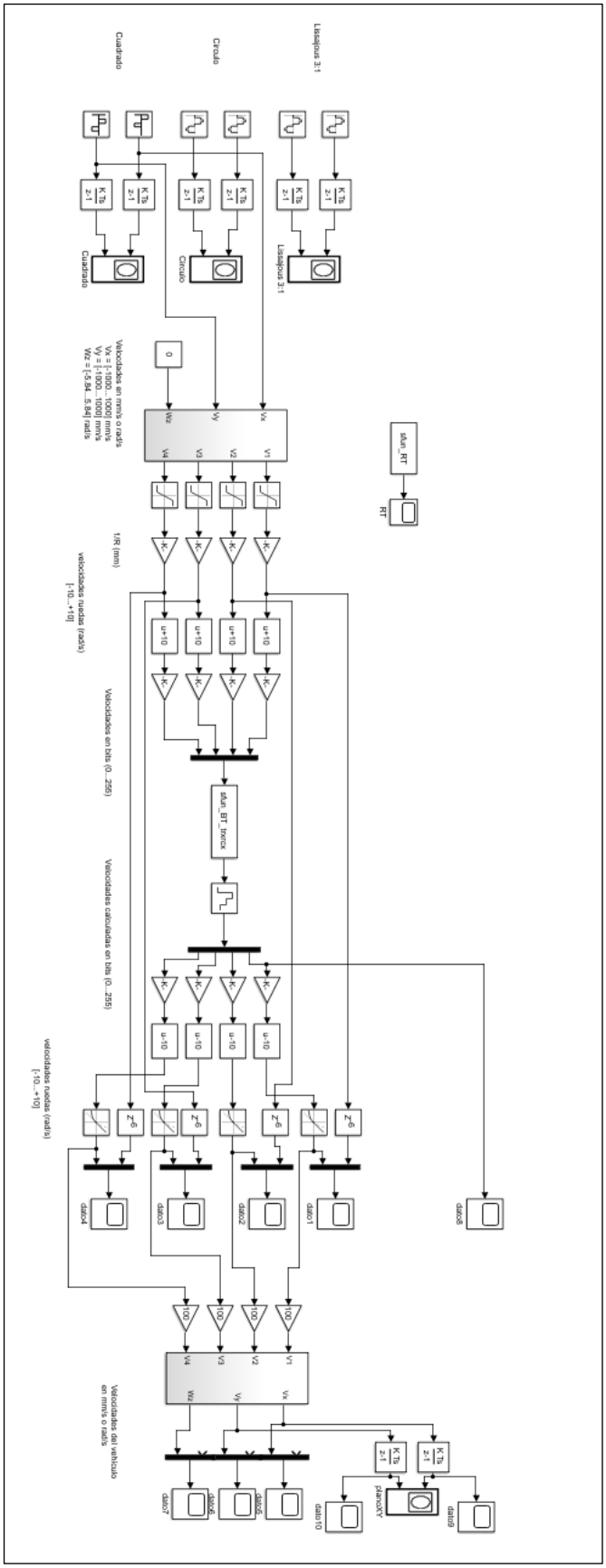


Figura 31: Diagrama de Simulink para las trayectorias

Anexo 2: Diagrama de Simulink para el manejo del robot a través del giroscopio de un Smartphone.

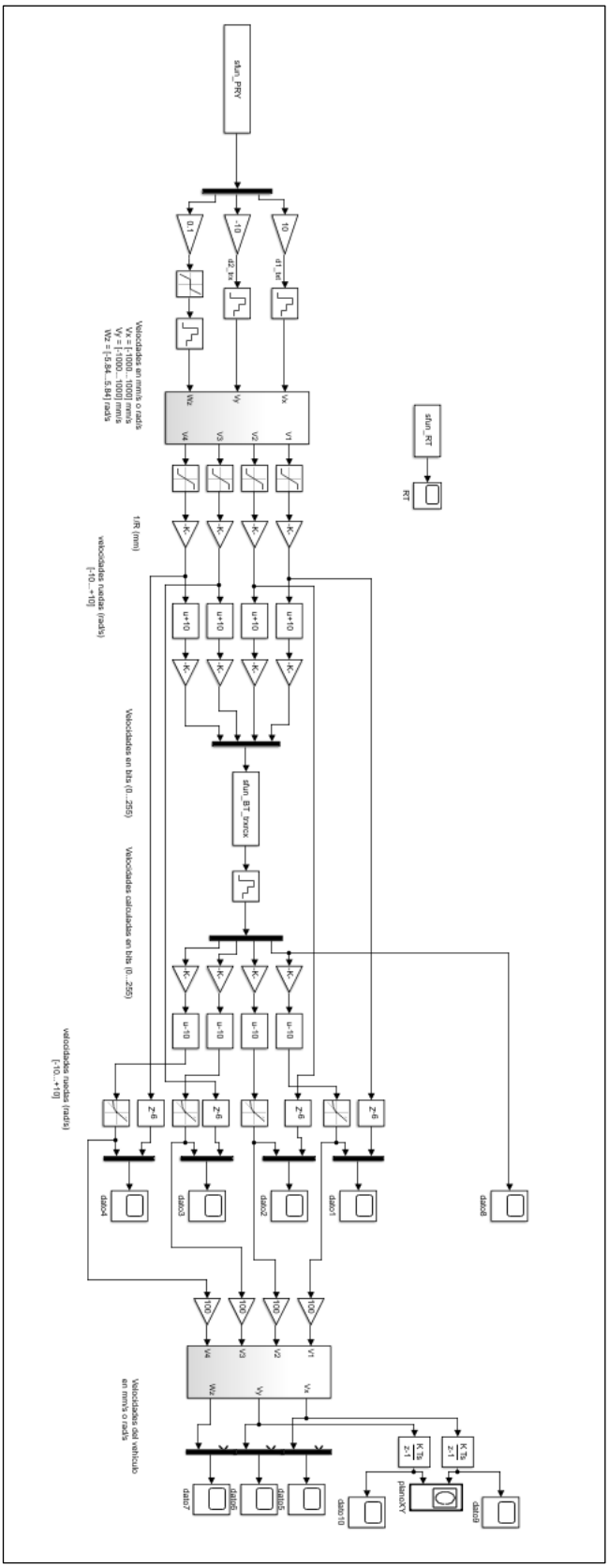


Figura 32: Diagrama de Simulink con la función sfun_PYR para el manejo del robot con el giroscopio

Anexo 3: Función sfun_RT para el control en tiempo real.

```
function [sys,x0,str,ts] = sfun_RT(t,x,u,flag,T)

global t1

switch flag,

  case 0,

    t1=tic;
    sizes = simsizes;
    sizes.NumContStates  = 0;
    sizes.NumDiscStates  = 0;
    sizes.NumOutputs     = 1;
    sizes.NumInputs      = 0;
    sizes.DirFeedthrough = 1;
    sizes.NumSampleTimes = 1;
    sys = simsizes(sizes);
    x0  = [];
    str = [];
    ts  = [0 0];

  case 3,
    t2=toc(t1) ;
    if t2>=T
        sys=[1] ;
    else
        sys=[0] ;
       while t2<T
           t2=toc(t1) ;
       end
    end
    t1=tic ;

  case 9,
    sys = [];

end
```

Anexo 4: Función sfun_BT_trxrcx para la comunicación vía Bluetooth.

```
function [sys,x0,str,ts] = sfun_BT_trxrcx(t,x,u,flag,BT,nn)

global b dd

switch flag,

  case 0,

    instrreset
    b = Bluetooth(BT,1) ;
    b.InputBuffer=nn ;
    b.OutputBuffer=nn ;
    b.TimeOut=0.001 ;
    fopen(b) ;
    dd=zeros(1,nn) ;


    sizes = simsizes;
    sizes.NumContStates  = 0;
    sizes.NumDiscStates  = 0;
    sizes.NumOutputs     = nn;
    sizes.NumInputs      = nn;
    sizes.DirFeedthrough = 1;
    sizes.NumSampleTimes = 1;
    sys = simsizes(sizes);
    x0  = [];
    str = [];
    ts  = [0 0];

  case 2,
    fwrite(b,u,'uint8') ;
    sys=[] ;

  case 3,
    if (b.BytesAvailable>=nn)
        dd=fread(b,nn) ;
    end
    sys=dd ;

  case 9,
    fclose(b) ;
    delete(b) ;
    sys = [];

end
```

Anexo 5: Función sfun_PRY para la comunicación vía Wifi de los sensores del Smartphone con el ordenador.

```
function [sys,x0,str,ts] = sfun_pry(t,x,u,flag)

global mov ori_ant yaw_ini

switch flag,

  case 0,

    sizes = simsizes;
    sizes.NumContStates  = 0;
    sizes.NumDiscStates  = 0;
    sizes.NumOutputs     = 3;
    sizes.NumInputs      = 0;
    sizes.DirFeedthrough = 1;
    sizes.NumSampleTimes = 1;
    sys = simsizes(sizes);
    x0  = [];
    str = [];
    ts  = [0 0];

    mov=mobiledev ;
    mov.OrientationSensorEnabled=1 ;
    mov.Logging=1 ;
    ori_ant=[0 0 0] ;

    ori=mov.orientation ;
    if isempty(ori) ;
        ori=ori_ant ;
    end
    yaw_ini=ori(1) ;


  case 3,
    ori=mov.orientation ;
    if isempty(ori) ;
        ori=ori_ant ;
    else
        ori_ant=ori ;
    end
    pitch=ori(2) ;
    roll=ori(3) ;
    yaw=ori(1)-yaw_ini ;
    sys=[pitch roll yaw] ;

  case 9,
    mov.Logging=0 ;
    mov.OrientationSensorEnabled=0 ;
    delete(mov) ;
    sys = [];

end
```

Anexo 6: Código de Arduino encargado de controlar la comunicación y la velocidad de los motores de las ruedas.

```
////////////////////////////////
////// ESQUEMA COCHE //////
////////////////////////////////
//
//      __                  __
//       | |_____________| |
//   A | |                 | | B
//      |_|                 |_|
//        |                 |
//     _|                  |_
//    | |                 | |
// C | |_____________| | D
//    |_|                 |_|


//Definir LED Real Time
#define RT 53

//Definir número de datos a transmitir por Bluetooth
#define nn 4

//Definir minimo y máximo ponderado de datos
#define MIN -1.0
#define MAX 1.0
#define VMIN -10.0
#define VMAX 10.0

//Definir sentidos de giro
#define CW 0
#define CCW 1

// Arduino pins for the shift register
#define MOTORLATCH 12
#define MOTORCLK 4
#define MOTORENABLE 7
#define MOTORDATA 8

// 8-bit bus after the 74HC595 shift register
// (not Arduino pins)
// These are used to set the direction of the bridge driver.
#define MOTOR1_A 2
#define MOTOR1_B 3
#define MOTOR2_A 1
#define MOTOR2_B 4
#define MOTOR3_A 5
#define MOTOR3_B 7
#define MOTOR4_A 0
#define MOTOR4_B 6

// Arduino pins for the PWM signals.
#define MOTOR1_PWM 11
#define MOTOR2_PWM 3
#define MOTOR3_PWM 6
#define MOTOR4_PWM 5
```

```
55
56 // Codes for the motor function.
57 #define FORWARD 1
58 #define BACKWARD 2
59 #define BRAKE 3
60 #define RELEASE 4
61
62 //Arduino pins para los encoders de los motores
63 #define pinENCA_chA 44
64 #define pinENCA_chB 45
65 #define pinENCB_chA 46
66 #define pinENCB_chB 47
67 #define pinENCC_chA 42
68 #define pinENCC_chB 43
69 #define pinENCD_chA 40
70 #define pinENCD_chB 39
71
72 //Definir máximos y mínimos de acc y potencia
73 #define MINU 0
74 #define MAXU 255
75 #define MINW -15.0
76 #define MAXW 15.0
77
78 //Variables de tiempo
79 long t1 = 0 ;
80 long t2 = 0 ;
81 int T = 10 ; // Periodo de muestreo milisegundos
82 float Tm = T/1000.0; // Periodo de muestreo segundos
83 int N=10 ;
84 int cont=0 ;
85
86 //Variables Bluetooth
87 bool flag = 0;
88
89 //Variables datos serie
90 float dato1 = 0.0 ;
91 float dato2 = 0.0 ;
92 float dato3 = 0.0 ;
93 float dato4 = 0.0 ;
94 byte leodato1 = 127.5;
95 byte leodato2 = 127.5;
96 byte leodato3 = 127.5;
97 byte leodato4 = 127.5;
98
99 float C2D = 360.0/4480.0; // Conversión cuentas a deg
100 float C2RS = (2*3.14/4480.0)/Tm; // Conversión cuentas a rad/seg
101
102 //Variables PI
103 float err1 = 0.0; // error de la señal
104 float err2 = 0.0; // error de la señal
105 float err3 = 0.0; // error de la señal
106 float err4 = 0.0; // error de la señal
107 float Ik1 = 0.0; // Integral del error
108 float Ik2 = 0.0; // Integral del error
109 float Ik3 = 0.0; // Integral del error
110 float Ik4 = 0.0; // Integral del error
```

```
111 float Kp = 18.0; // Constante proporcional
112 float Ki = 5.0; // Constante integral
113
114
115 //Variables
116 float accA = 0.0; //Acción de control motor A
117 float accB = 0.0; //Acción de control motor B
118 float accC = 0.0; //Acción de control motor C
119 float accD = 0.0; //Acción de control motor D
120 float velA = 0.0; //Velocidad motor A
121 float velB = 0.0; //Velocidad motor B
122 float velC = 0.0; //Velocidad motor C
123 float velD = 0.0; //Velocidad motor D
124 byte velAbyte = 0; //Velocidad motor A en bytes
125 byte velBbyte = 0; //Velocidad motor B en bytes
126 byte velCbyte = 0; //Velocidad motor C en bytes
127 byte velDbyte = 0; //Velocidad motor D en bytes
128 float posA = 0.0; //Posición motor A
129 float posB = 0.0; //Posición motor B
130 float posC = 0.0; //Posición motor C
131 float posD = 0.0; //Posición motor D
132 long quadA = 2241; //
133 long quadB = 2241; //
134 long quadC = 2241; //
135 long quadD = 2241; //
136 long enc_antA = quadA; //Encoder anterior A
137 long enc_antB = quadB; //Encoder anterior B
138 long enc_antC = quadC; //Encoder anterior C
139 long enc_antD = quadD; //Encoder anterior D
140 long enc_actA = quadA; //Encoder actual A
141 long enc_actB = quadB; //Encoder actual B
142 long enc_actC = quadC; //Encoder actual C
143 long enc_actD = quadD; //Encoder actual D
144 int enc_difA = 0; //Diferencia encoders A
145 int enc_difB = 0; //Diferencia encoders B
146 int enc_difC = 0; //Diferencia encoders C
147 int enc_difD = 0; //Diferencia encoders D
148 int dirA; //Dirección motor A
149 int dirB; //Dirección motor B
150 int dirC; //Dirección motor C
151 int dirD; //Dirección motor D
152
153 ////////////////////////////////////////
154 /////////////// SETUP ///////////////
155 ////////////////////////////////////////
156
157 void setup()
158 {
159 Serial.begin(115200) ;
160 pinMode(RT,OUTPUT) ;
161
162 //Configirar velocidad serial 1 (Bluetooth)
163 Serial1.begin(115200);
164
165 //Funciones de interrupción de actualización de los encoders
166 attachInterrupt(pinENCA_chA,encoderA_chA,CHANGE);
```

```
attachInterrupt(pinENCA_chB,encoderA_chB,CHANGE);
attachInterrupt(pinENCB_chA,encoderB_chA,CHANGE);
attachInterrupt(pinENCB_chB,encoderB_chB,CHANGE);
attachInterrupt(pinENCC_chA,encoderC_chA,CHANGE);
attachInterrupt(pinENCC_chB,encoderC_chB,CHANGE);
attachInterrupt(pinENCD_chA,encoderD_chA,CHANGE);
attachInterrupt(pinENCD_chB,encoderD_chB,CHANGE);

//Resolución de lectura y escritura
analogWriteResolution(8);
analogReadResolution(8);

delay(1000);
}


////////////////////////////////////
///////////// LOOP ////////////////
////////////////////////////////////

void loop()
{
flag = 1;
if(cont == N)
{
if (Serial1.available() >= nn)
{
flag = 1;

//Lectura del Serial
leodato1 = Serial1.read();
leodato2 = Serial1.read();
leodato3 = Serial1.read();
leodato4 = Serial1.read();

//Escribir datos en el serial
Serial1.write(velAbyte);
Serial1.write(velBbyte);
Serial1.write(velCbyte);
Serial1.write(velDbyte);

cont = 0;
}
}
else
{
dato1 = (leodato1*(VMAX-VMIN)/255.0)+VMIN ; //lectura de las velocidades
(rad/s) y escaladas [-10...+10]
dato2 = (leodato2*(VMAX-VMIN)/255.0)+VMIN ; //lectura de las velocidades
(rad/s) y escaladas [-10...+10]
dato3 = (leodato3*(VMAX-VMIN)/255.0)+VMIN ; //lectura de las velocidades
(rad/s) y escaladas [-10...+10]
dato4 = (leodato4*(VMAX-VMIN)/255.0)+VMIN ; //lectura de las velocidades
(rad/s) y escaladas [-10...+10]

//Cálculo velocidad motores
```

```
enc_actA = quadA;
enc_actB = quadB;
enc_actC = quadC;
enc_actD = quadD;
enc_difA = enc_actA - enc_antA;
enc_difB = enc_actB - enc_antB;
enc_difC = enc_actC - enc_antC;
enc_difD = enc_actD - enc_antD;
velA = (enc_actA - enc_antA)*C2RS;
velB = (enc_actB - enc_antB)*C2RS;
velC = (enc_actC - enc_antC)*C2RS;
velD = (enc_actD - enc_antD)*C2RS;
enc_antA = enc_actA;
enc_antB = enc_actB;
enc_antC = enc_actC;
enc_antD = enc_actD;

err1 = dato1 - velA; //Calculo error motor A
err2 = dato2 - velB; //Calculo error motor B
err3 = dato3 - velC; //Calculo error motor C
err4 = dato4 - velD; //Calculo error motor D

Ik1 = err1*Tm + Ik1; //Calculo integral del error
Ik2 = err2*Tm + Ik2; //Calculo integral del error
Ik3 = err3*Tm + Ik3; //Calculo integral del error
Ik4 = err4*Tm + Ik4; //Calculo integral del error

//Cálculo Acciones de Control
accA = abs(Kp*err1 + Ki*Ik1);
accB = abs(Kp*err2 + Ki*Ik2);
accC = abs(Kp*err3 + Ki*Ik3);
accD = abs(Kp*err4 + Ki*Ik4);

//Saturar acción de control
if(accA > VMAX) {accA = VMAX; digitalWrite(RT,HIGH);} if (accA < VMIN) {accA =
VMIN; digitalWrite(RT,HIGH);}
if(accB > VMAX) {accB = VMAX; digitalWrite(RT,HIGH);} if (accB < VMIN) {accB =
VMIN; digitalWrite(RT,HIGH);}
if(accC > VMAX) {accC = VMAX; digitalWrite(RT,HIGH);} if (accC < VMIN) {accC =
VMIN; digitalWrite(RT,HIGH);}
if(accD > VMAX) {accD = VMAX; digitalWrite(RT,HIGH);} if (accD < VMIN) {accD =
VMIN; digitalWrite(RT,HIGH);}

//Transformar acciones de control de [-10...10] a [0...255]
accA = (accA + VMAX)*255/(VMAX - VMIN);
accB = (accB + VMAX)*255/(VMAX - VMIN);
accC = (accC + VMAX)*255/(VMAX - VMIN);
accD = (accD + VMAX)*255/(VMAX - VMIN);

//Definir sentido de giro
if(int(dato1) > 0) dirA = FORWARD; else if(int(dato1) == 0) dirA = RELEASE ;
else dirA = BACKWARD;
if(int(dato2) > 0) dirB = FORWARD; else if(int(dato2) == 0) dirB = RELEASE ;
else dirB = BACKWARD;
if(int(dato3) > 0) dirC = FORWARD; else if(int(dato3) == 0) dirC = RELEASE ;
else dirC = BACKWARD;
```

```
268 if(int(dato4) > 0) dirD = FORWARD; else if(int(dato4) == 0) dirD = RELEASE ;
else dirD = BACKWARD;
269
270 //Escribir acción de control
271 motor(1,dirA,accA);
272 motor(2,dirB,accB);
273 motor(3,dirC,accC);
274 motor(4,dirD,accD);
275
276 //Transformar velocidad en bytes
277 velAbyte = (velA + VMAX)*255/(VMAX-VMIN); // de [-10...+10] a [0...255]
278 velBbyte = (velB + VMAX)*255/(VMAX-VMIN); // de [-10...+10] a [0...255]
279 velCbyte = (velC + VMAX)*255/(VMAX-VMIN); // de [-10...+10] a [0...255]
280 velDbyte = (velD + VMAX)*255/(VMAX-VMIN); // de [-10...+10] a [0...255]
281
282 cont++;
283
284 Serial.print(cont); Serial.print(" ");
285 Serial.print(velA); Serial.print(" ");
286 Serial.print(err1); Serial.print(" ");
287 Serial.print(accA); Serial.print(" ");
288 //Serial.print(velA); Serial.print(" ");
289 //Serial.print(velB); Serial.print(" ");
290 //Serial.print(velC); Serial.print(" ");
291 //Serial.print(velD); Serial.print(" ");
292 Serial.println("");
293 }
294 Serial1.flush() ;
295
296 if (flag)
297 {
298 for (int ii=1 ; ii<=nn ; ii++){}
299 //Serial1.flush() ;
300 } else Serial.println("No conectado") ;
301
302 //Acabar ciclo
303 t2=millis() ;
304 if (t2-t1>T) {digitalWrite(RT,HIGH) ;}
305 else {digitalWrite(RT,LOW) ;}
306 while (t2-t1<T) {t2=millis() ;}
307 t1=millis() ;
308 }
309
310 //Actualización de los encoders
311 void encoderA_chA(){if(digitalRead(pinENCA_chA)==digitalRead(pinENCA_chB)) quadA--;
else quadA++;}
312 void encoderA_chB(){if(digitalRead(pinENCA_chA)==digitalRead(pinENCA_chB)) quadA++;
else quadA--;}
313
314 void encoderB_chA(){if(digitalRead(pinENCB_chA)==digitalRead(pinENCB_chB)) quadB--;
else quadB++;}
315 void encoderB_chB(){if(digitalRead(pinENCB_chA)==digitalRead(pinENCB_chB)) quadB++;
else quadB--;}
316
317 void encoderC_chA(){if(digitalRead(pinENCC_chA)==digitalRead(pinENCC_chB)) quadC--;
else quadC++;}
```

```
318 void encoderC_chB(){if(digitalRead(pinENCC_chA)==digitalRead(pinENCC_chB)) quadC++;
else quadC--;}
319
320 void encoderD_chA(){if(digitalRead(pinENCD_chA)==digitalRead(pinENCD_chB)) quadD--;
else quadD++;}
321 void encoderD_chB(){if(digitalRead(pinENCD_chA)==digitalRead(pinENCD_chB)) quadD++;
else quadD--;}
322
323
324 // --------------------------------
325 // motor
326 //
327 // Select the motor (1-4), the command,
328 // and the speed (0-255).
329 // The commands are: FORWARD, BACKWARD, BRAKE, RELEASE.
330 //
331 void motor(int nMotor, int command, int speed)
332 {
333 int motorA, motorB;
334
335 if (nMotor >= 1 && nMotor <= 4)
336 {
337 switch (nMotor)
338 {
339 case 1:
340 motorA = MOTOR1_A;
341 motorB = MOTOR1_B;
342 break;
343 case 2:
344 motorA = MOTOR2_A;
345 motorB = MOTOR2_B;
346 break;
347 case 3:
348 motorA = MOTOR3_A;
349 motorB = MOTOR3_B;
350 break;
351 case 4:
352 motorA = MOTOR4_A;
353 motorB = MOTOR4_B;
354 break;
355 default:
356 break;
357 }
358
359 switch (command)
360 {
361 case FORWARD:
362 motor_output (motorA, HIGH, speed);
363 motor_output (motorB, LOW, -1); // -1: no PWM set
364 break;
365 case BACKWARD:
366 motor_output (motorA, LOW, speed);
367 motor_output (motorB, HIGH, -1); // -1: no PWM set
368 break;
369 case BRAKE:
370 // The AdaFruit library didn't implement a brake.
```

```
371 // The L293D motor driver ic doesn't have a good
372 // brake anyway.
373 // It uses transistors inside, and not mosfets.
374 // Some use a software break, by using a short
375 // reverse voltage.
376 // This brake will try to brake, by enabling
377 // the output and by pulling both outputs to ground.
378 // But it isn't a good break.
379 motor_output (motorA, LOW, 255); // 255: fully on.
380 motor_output (motorB, LOW, -1); // -1: no PWM set
381 break;
382 case RELEASE:
383 motor_output (motorA, LOW, 0); // 0: output floating.
384 motor_output (motorB, LOW, -1); // -1: no PWM set
385 break;
386 default:
387 break;
388 }
389 }
390 }
391
392 // --------------------------------
393 // motor_output
394 //
395 // The function motor_ouput uses the motor driver to
396 // drive normal outputs like lights, relays, solenoids,
397 // DC motors (but not in reverse).
398 //
399 // It is also used as an internal helper function
400 // for the motor() function.
401 //
402 // The high_low variable should be set 'HIGH'
403 // to drive lights, etc.
404 // It can be set 'LOW', to switch it off,
405 // but also a 'speed' of 0 will switch it off.
406 //
407 // The 'speed' sets the PWM for 0...255, and is for
408 // both pins of the motor output.
409 // For example, if motor 3 side 'A' is used to for a
410 // dimmed light at 50% (speed is 128), also the
411 // motor 3 side 'B' output will be dimmed for 50%.
412 // Set to 0 for completelty off (high impedance).
413 // Set to 255 for fully on.
414 // Special settings for the PWM speed:
415 // Set to -1 for not setting the PWM at all.
416 //
417 void motor_output (int output, int high_low, int speed)
418 {
419 int motorPWM;
420
421 switch (output)
422 {
423 case MOTOR1_A:
424 case MOTOR1_B:
425 motorPWM = MOTOR1_PWM;
426 break;
```

```
427 case MOTOR2_A:
428 case MOTOR2_B:
429 motorPWM = MOTOR2_PWM;
430 break;
431 case MOTOR3_A:
432 case MOTOR3_B:
433 motorPWM = MOTOR3_PWM;
434 break;
435 case MOTOR4_A:
436 case MOTOR4_B:
437 motorPWM = MOTOR4_PWM;
438 break;
439 default:
440 // Use speed as error flag, -3333 = invalid output.
441 speed = -3333;
442 break;
443 }
444
445 if (speed != -3333)
446 {
447 // Set the direction with the shift register
448 // on the MotorShield, even if the speed = -1.
449 // In that case the direction will be set, but
450 // not the PWM.
451 shiftWrite(output, high_low);
452
453 // set PWM only if it is valid
454 if (speed >= 0 && speed <= 255)
455 {
456 analogWrite(motorPWM, speed);
457 }
458 }
459 }
460
461 // --------------------------------
462 // shiftWrite
463 //
464 // The parameters are just like digitalWrite().
465 //
466 // The output is the pin 0...7 (the pin behind
467 // the shift register).
468 // The second parameter is HIGH or LOW.
469 //
470 // There is no initialization function.
471 // Initialization is automatically done at the first
472 // time it is used.
473 //
474 void shiftWrite(int output, int high_low)
475 {
476 static int latch_copy;
477 static int shift_register_initialized = false;
478
479 // Do the initialization on the fly,
480 // at the first time it is used.
481 if (!shift_register_initialized)
482 {
```

```
483 // Set pins for shift register to output
484 pinMode(MOTORLATCH, OUTPUT);
485 pinMode(MOTORENABLE, OUTPUT);
486 pinMode(MOTORDATA, OUTPUT);
487 pinMode(MOTORCLK, OUTPUT);
488
489 // Set pins for shift register to default value (low);
490 digitalWrite(MOTORDATA, LOW);
491 digitalWrite(MOTORLATCH, LOW);
492 digitalWrite(MOTORCLK, LOW);
493 // Enable the shift register, set Enable pin Low.
494 digitalWrite(MOTORENABLE, LOW);
495
496 // start with all outputs (of the shift register) low
497 latch_copy = 0;
498
499 shift_register_initialized = true;
500 }
501
502 // The defines HIGH and LOW are 1 and 0.
503 // So this is valid.
504 bitWrite(latch_copy, output, high_low);
505
506 // Use the default Arduino 'shiftOut()' function to
507 // shift the bits with the MOTORCLK as clock pulse.
508 // The 74HC595 shiftregister wants the MSB first.
509 // After that, generate a latch pulse with MOTORLATCH.
510 shiftOut(MOTORDATA, MOTORCLK, MSBFIRST, latch_copy);
511 delayMicroseconds(5); // For safety, not really needed.
512 digitalWrite(MOTORLATCH, HIGH);
513 delayMicroseconds(5); // For safety, not really needed.
514 digitalWrite(MOTORLATCH, LOW);
515 }
516
```

Anexo 7: Diagrama de Conexiones

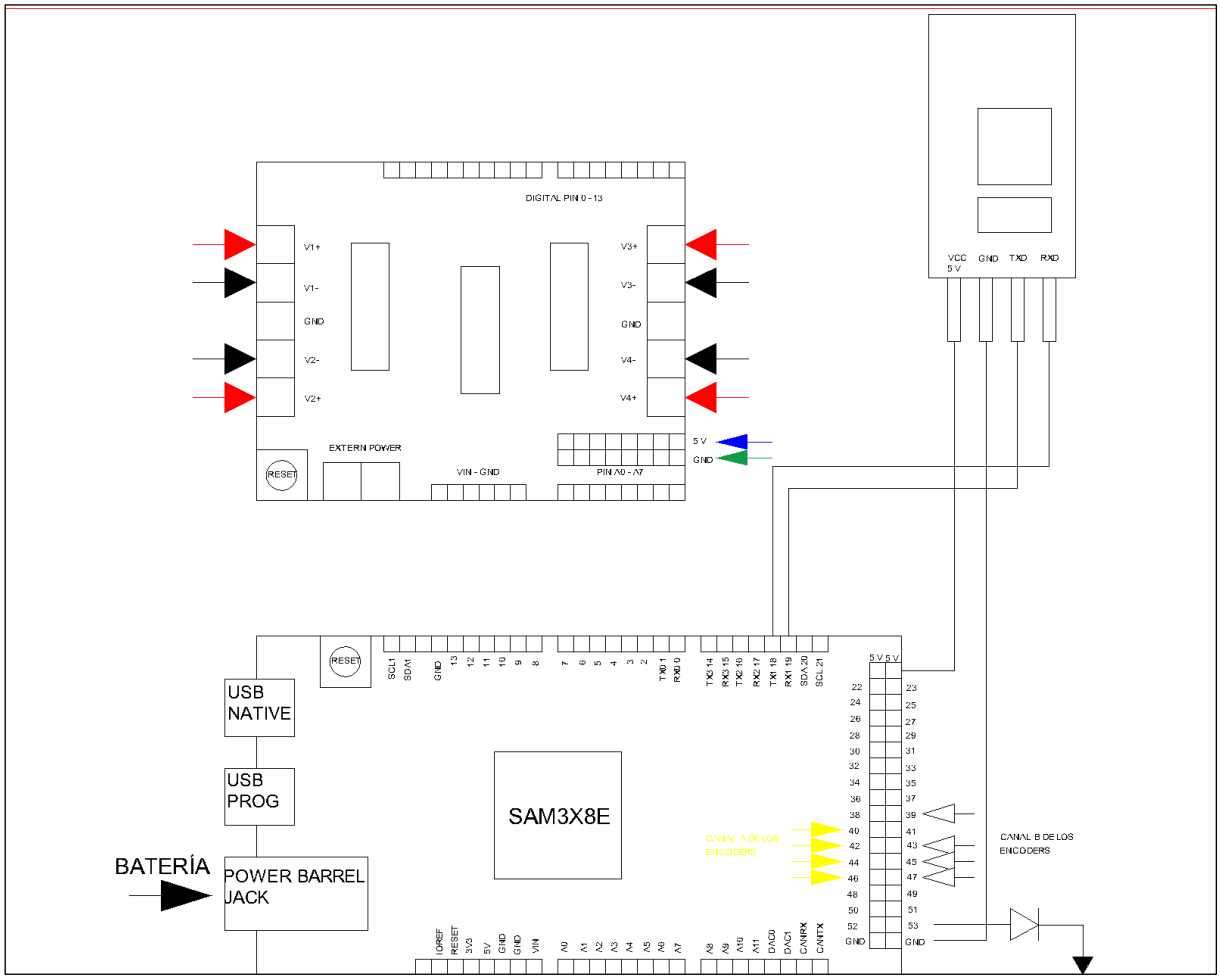


Figura 32: Diagrama de conexiones entre Arduino DUE, Shield V1 y HC-06

Anexo 8: Código Arduino modificado para la aplicación Android para el manejo del robot a través de Bluetooth.

```
////////////////////////////////
////// ESQUEMA COCHE //////
////////////////////////////////
//
//      __                  __
//       | |_____________| |
//   A | |                 | | B
//      |_|                 |_|
//        |                 |
//     _|                  |_
//    | |                 | |
// C | |_____________| | D
//    |_|                 |_|


//Definir LED Real Time
#define RT 53

//Definir número de datos a transmitir por Bluetooth
#define nn 2

//Definir minimo y máximo ponderado de datos
#define MIN -1.0
#define MAX 1.0
#define VMIN -10.0
#define VMAX 10.0

//Definir sentidos de giro
#define CW 0
#define CCW 1

// Arduino pins for the shift register
#define MOTORLATCH 12
#define MOTORCLK 4
#define MOTORENABLE 7
#define MOTORDATA 8

// 8-bit bus after the 74HC595 shift register
// (not Arduino pins)
// These are used to set the direction of the bridge driver.
#define MOTOR1_A 2
#define MOTOR1_B 3
#define MOTOR2_A 1
#define MOTOR2_B 4
#define MOTOR3_A 5
#define MOTOR3_B 7
#define MOTOR4_A 0
#define MOTOR4_B 6

// Arduino pins for the PWM signals.
#define MOTOR1_PWM 11
#define MOTOR2_PWM 3
#define MOTOR3_PWM 6
#define MOTOR4_PWM 5
```

```
55
56 // Codes for the motor function.
57 #define FORWARD 1
58 #define BACKWARD 2
59 #define BRAKE 3
60 #define RELEASE 4
61
62 //Arduino pins para los encoders de los motores
63 #define pinENCA_chA 44
64 #define pinENCA_chB 45
65 #define pinENCB_chA 46
66 #define pinENCB_chB 47
67 #define pinENCC_chA 42
68 #define pinENCC_chB 43
69 #define pinENCD_chA 40
70 #define pinENCD_chB 39
71
72 //Definir máximos y mínimos de acc y potencia
73 #define MINU 0
74 #define MAXU 255
75 #define MINW -15.0
76 #define MAXW 15.0
77
78 //Variables de tiempo
79 long t1 = 0 ;
80 long t2 = 0 ;
81 int T = 10 ; // Periodo de muestreo milisegundos
82 float Tm = T/1000.0; // Periodo de muestreo segundos
83 int N=10 ;
84 int cont=0 ;
85
86 //Variables Bluetooth
87 bool flag = 0;
88
89 //Variables datos serie
90 float dato1;
91 float dato2;
92 float dato3;
93 float dato4;
94 char leodato;
95
96 float C2D = 360.0/4480.0; // Conversión cuentas a deg
97 float C2RS = (2*3.14/4480.0)/Tm; // Conversión cuentas a rad/seg
98
99 //Variables PI
100 float err1 = 0.0; // error de la señal
101 float err2 = 0.0; // error de la señal
102 float err3 = 0.0; // error de la señal
103 float err4 = 0.0; // error de la señal
104 float Ik1 = 0.0; // Integral del error
105 float Ik2 = 0.0; // Integral del error
106 float Ik3 = 0.0; // Integral del error
107 float Ik4 = 0.0; // Integral del error
108 float Kp = 18.0; // Constante proporcional
109 float Ki = 5.0; // Constante integral
110
```

```
111
112 //Variables
113 float accA = 0.0; //Acción de control motor A
114 float accB = 0.0; //Acción de control motor B
115 float accC = 0.0; //Acción de control motor C
116 float accD = 0.0; //Acción de control motor D
117 float velA = 0.0; //Velocidad motor A
118 float velB = 0.0; //Velocidad motor B
119 float velC = 0.0; //Velocidad motor C
120 float velD = 0.0; //Velocidad motor D
121 byte velAbyte = 0; //Velocidad motor A en bytes
122 byte velBbyte = 0; //Velocidad motor B en bytes
123 byte velCbyte = 0; //Velocidad motor C en bytes
124 byte velDbyte = 0; //Velocidad motor D en bytes
125 float posA = 0.0; //Posición motor A
126 float posB = 0.0; //Posición motor B
127 float posC = 0.0; //Posición motor C
128 float posD = 0.0; //Posición motor D
129 long quadA = 2241; //
130 long quadB = 2241; //
131 long quadC = 2241; //
132 long quadD = 2241; //
133 long enc_antA = quadA; //Encoder anterior A
134 long enc_antB = quadB; //Encoder anterior B
135 long enc_antC = quadC; //Encoder anterior C
136 long enc_antD = quadD; //Encoder anterior D
137 long enc_actA = quadA; //Encoder actual A
138 long enc_actB = quadB; //Encoder actual B
139 long enc_actC = quadC; //Encoder actual C
140 long enc_actD = quadD; //Encoder actual D
141 int enc_difA = 0; //Diferencia encoders A
142 int enc_difB = 0; //Diferencia encoders B
143 int enc_difC = 0; //Diferencia encoders C
144 int enc_difD = 0; //Diferencia encoders D
145 int dirA; //Dirección motor A
146 int dirB; //Dirección motor B
147 int dirC; //Dirección motor C
148 int dirD; //Dirección motor D
149
150 ////////////////////////////////////////
151 ////////////// SETUP ///////////////
152 ////////////////////////////////////////
153
154 void setup()
155 {
156 Serial.begin(115200) ;
157 pinMode(RT,OUTPUT) ;
158
159 //Configirar velocidad serial 1 (Bluetooth)
160 Serial1.begin(115200);
161
162 //Funciones de interrupción de actualización de los encoders
163 attachInterrupt(pinENCA_chA,encoderA_chA,CHANGE);
164 attachInterrupt(pinENCA_chB,encoderA_chB,CHANGE);
165 attachInterrupt(pinENCB_chA,encoderB_chA,CHANGE);
166 attachInterrupt(pinENCB_chB,encoderB_chB,CHANGE);
```

```
167 attachInterrupt(pinENCC_chA,encoderC_chA,CHANGE);
168 attachInterrupt(pinENCC_chB,encoderC_chB,CHANGE);
169 attachInterrupt(pinENCD_chA,encoderD_chA,CHANGE);
170 attachInterrupt(pinENCD_chB,encoderD_chB,CHANGE);
171
172 //Resolución de lectura y escritura
173 analogWriteResolution(8);
174 analogReadResolution(8);
175
176 delay(1000);
177 }
178
179
180 ////////////////////////////////////
181 ////////////// LOOP ////////////////
182 ////////////////////////////////////
183
184 void loop()
185 {
186 flag = 1;
187 //Realizar las lecturas del Serial del Bluetooth cada 10 ciclos (100 ms)
188 if(cont == N)
189 {
190 if (Serial1.available())
191 {
192 flag = 1;
193
194 //Lectura del Serial
195 leodato = Serial1.read();
196
197 //Velocidades de las ruedas en rad/s
198
199 if (leodato == '0') //RELEASE
200 {
201 dato1 = 0;
202 dato2 = 0;
203 dato3 = 0;
204 dato4 = 0;
205 }
206 if (leodato == '1') //Forward
207 {
208 dato1 = 5;
209 dato2 = 5;
210 dato3 = 5;
211 dato4 = 5;
212 }
213 if (leodato == '2') //Right
214 {
215 dato1 = 5;
216 dato2 = -5;
217 dato3 = -5;
218 dato4 = 5;
219 }
220 if (leodato == '3') //Backward
221 {
222 dato1 = -5;
```

```
223 dato2 = -5;
224 dato3 = -5;
225 dato4 = -5;
226 }
227 if (leodato == '4') //Left
228 {
229 dato1 = -5;
230 dato2 = 5;
231 dato3 = 5;
232 dato4 = -5;
233 }
234 if (leodato == '5') //Forward Right
235 {
236 dato1 = 5;
237 dato2 = 0;
238 dato3 = 0;
239 dato4 = 5;
240 }
241 if (leodato == '6') //Backward Right
242 {
243 dato1 = 0;
244 dato2 = -5;
245 dato3 = -5;
246 dato4 = 0;
247 }
248 if (leodato == '7') //Backward Left
249 {
250 dato1 = -5;
251 dato2 = 0;
252 dato3 = 0;
253 dato4 = -5;
254 }
255 if (leodato == '8') //Forward Left
256 {
257 dato1 = 0;
258 dato2 = 5;
259 dato3 = 5;
260 dato4 = 0;
261 }
262 if (leodato == 'A') //Giro reloj
263 {
264 dato1 = 5;
265 dato2 = -5;
266 dato3 = 5;
267 dato4 = -5;
268 }
269 if (leodato == 'B') //Giro anti reloj
270 {
271 dato1 = -5;
272 dato2 = 5;
273 dato3 = -5;
274 dato4 = 5;
275 }
276
277 cont = 0;
278 }
```

```
279 }
280 else
281 {
282 //Cálculo velocidad motores
283 enc_actA = quadA;
284 enc_actB = quadB;
285 enc_actC = quadC;
286 enc_actD = quadD;
287 enc_difA = enc_actA - enc_antA;
288 enc_difB = enc_actB - enc_antB;
289 enc_difC = enc_actC - enc_antC;
290 enc_difD = enc_actD - enc_antD;
291 velA = (enc_actA - enc_antA)*C2RS;
292 velB = (enc_actB - enc_antB)*C2RS;
293 velC = (enc_actC - enc_antC)*C2RS;
294 velD = (enc_actD - enc_antD)*C2RS;
295 enc_antA = enc_actA;
296 enc_antB = enc_actB;
297 enc_antC = enc_actC;
298 enc_antD = enc_actD;
299
300 err1 = dato1 - velA; //Calculo error motor A
301 err2 = dato2 - velB; //Calculo error motor B
302 err3 = dato3 - velC; //Calculo error motor C
303 err4 = dato4 - velD; //Calculo error motor D
304
305 Ik1 = err1*Tm + Ik1; //Calculo integral del error
306 Ik2 = err2*Tm + Ik2; //Calculo integral del error
307 Ik3 = err3*Tm + Ik3; //Calculo integral del error
308 Ik4 = err4*Tm + Ik4; //Calculo integral del error
309
310 //Cálculo Acciones de Control
311 accA = abs(Kp*err1 + Ki*Ik1);
312 accB = abs(Kp*err2 + Ki*Ik2);
313 accC = abs(Kp*err3 + Ki*Ik3);
314 accD = abs(Kp*err4 + Ki*Ik4);
315
316 //Saturar acción de control
317 if(accA > VMAX) {accA = VMAX; digitalWrite(RT,HIGH);} if (accA < VMIN) {accA =
VMIN; digitalWrite(RT,HIGH);}
318 if(accB > VMAX) {accB = VMAX; digitalWrite(RT,HIGH);} if (accB < VMIN) {accB =
VMIN; digitalWrite(RT,HIGH);}
319 if(accC > VMAX) {accC = VMAX; digitalWrite(RT,HIGH);} if (accC < VMIN) {accC =
VMIN; digitalWrite(RT,HIGH);}
320 if(accD > VMAX) {accD = VMAX; digitalWrite(RT,HIGH);} if (accD < VMIN) {accD =
VMIN; digitalWrite(RT,HIGH);}
321
322 //Transformar acciones de control de [-10...10] a [0...255]
323 accA = (accA + VMAX)*255/(VMAX - VMIN);
324 accB = (accB + VMAX)*255/(VMAX - VMIN);
325 accC = (accC + VMAX)*255/(VMAX - VMIN);
326 accD = (accD + VMAX)*255/(VMAX - VMIN);
327
328 //Definir sentido de giro
329 if(int(dato1) > 0) dirA = FORWARD; else if(int(dato1) == 0) dirA = RELEASE ;
else dirA = BACKWARD;
```

```
330 if(int(dato2) > 0) dirB = FORWARD; else if(int(dato2) == 0) dirB = RELEASE ;
else dirB = BACKWARD;
331 if(int(dato3) > 0) dirC = FORWARD; else if(int(dato3) == 0) dirC = RELEASE ;
else dirC = BACKWARD;
332 if(int(dato4) > 0) dirD = FORWARD; else if(int(dato4) == 0) dirD = RELEASE ;
else dirD = BACKWARD;
333
334 //Escribir acción de control
335 motor(1,dirA,accA);
336 motor(2,dirB,accB);
337 motor(3,dirC,accC);
338 motor(4,dirD,accD);
339
340 //Transformar velocidad en bytes
341 velAbyte = (velA + VMAX)*255/(VMAX-VMIN); // de [-10...+10] a [0...255]
342 velBbyte = (velB + VMAX)*255/(VMAX-VMIN); // de [-10...+10] a [0...255]
343 velCbyte = (velC + VMAX)*255/(VMAX-VMIN); // de [-10...+10] a [0...255]
344 velDbyte = (velD + VMAX)*255/(VMAX-VMIN); // de [-10...+10] a [0...255]
345
346 cont++;
347 }
348 Serial1.flush() ;
349
350 if (flag)
351 {
352 for (int ii=1 ; ii<=nn ; ii++){}
353 //Serial1.flush() ;
354 } else Serial.println("No conectado") ;
355
356 //Acabar ciclo
357 t2=millis() ;
358 if (t2-t1>T) {digitalWrite(RT,HIGH) ;}
359 else {digitalWrite(RT,LOW) ;}
360 while (t2-t1<T) {t2=millis() ;}
361 t1=millis() ;
362 }
363
364 //Actualización de los encoders
365 void encoderA_chA(){if(digitalRead(pinENCA_chA)==digitalRead(pinENCA_chB)) quadA--;
else quadA++;}
366 void encoderA_chB(){if(digitalRead(pinENCA_chA)==digitalRead(pinENCA_chB)) quadA++;
else quadA--;}
367
368 void encoderB_chA(){if(digitalRead(pinENCB_chA)==digitalRead(pinENCB_chB)) quadB--;
else quadB++;}
369 void encoderB_chB(){if(digitalRead(pinENCB_chA)==digitalRead(pinENCB_chB)) quadB++;
else quadB--;}
370
371 void encoderC_chA(){if(digitalRead(pinENCC_chA)==digitalRead(pinENCC_chB)) quadC--;
else quadC++;}
372 void encoderC_chB(){if(digitalRead(pinENCC_chA)==digitalRead(pinENCC_chB)) quadC++;
else quadC--;}
373
374 void encoderD_chA(){if(digitalRead(pinENCD_chA)==digitalRead(pinENCD_chB)) quadD--;
else quadD++;}
375 void encoderD_chB(){if(digitalRead(pinENCD_chA)==digitalRead(pinENCD_chB)) quadD++;
```

```
else quadD--;}


// --------------------------------
// motor
//
// Select the motor (1-4), the command,
// and the speed (0-255).
// The commands are: FORWARD, BACKWARD, BRAKE, RELEASE.
//
void motor(int nMotor, int command, int speed)
{
int motorA, motorB;

if (nMotor >= 1 && nMotor <= 4)
{
switch (nMotor)
{
case 1:
motorA = MOTOR1_A;
motorB = MOTOR1_B;
break;
case 2:
motorA = MOTOR2_A;
motorB = MOTOR2_B;
break;
case 3:
motorA = MOTOR3_A;
motorB = MOTOR3_B;
break;
case 4:
motorA = MOTOR4_A;
motorB = MOTOR4_B;
break;
default:
break;
}

switch (command)
{
case FORWARD:
motor_output (motorA, HIGH, speed);
motor_output (motorB, LOW, -1); // -1: no PWM set
break;
case BACKWARD:
motor_output (motorA, LOW, speed);
motor_output (motorB, HIGH, -1); // -1: no PWM set
break;
case BRAKE:
// The AdaFruit library didn't implement a brake.
// The L293D motor driver ic doesn't have a good
// brake anyway.
// It uses transistors inside, and not mosfets.
// Some use a software break, by using a short
// reverse voltage.
// This brake will try to brake, by enabling
```

```
// the output and by pulling both outputs to ground.
// But it isn't a good break.
motor_output (motorA, LOW, 255); // 255: fully on.
motor_output (motorB, LOW, -1); // -1: no PWM set
break;
case RELEASE:
motor_output (motorA, LOW, 0); // 0: output floating.
motor_output (motorB, LOW, -1); // -1: no PWM set
break;
default:
break;
}
}
}

// --------------------------------
// motor_output
//
// The function motor_ouput uses the motor driver to
// drive normal outputs like lights, relays, solenoids,
// DC motors (but not in reverse).
//
// It is also used as an internal helper function
// for the motor() function.
//
// The high_low variable should be set 'HIGH'
// to drive lights, etc.
// It can be set 'LOW', to switch it off,
// but also a 'speed' of 0 will switch it off.
//
// The 'speed' sets the PWM for 0...255, and is for
// both pins of the motor output.
// For example, if motor 3 side 'A' is used to for a
// dimmed light at 50% (speed is 128), also the
// motor 3 side 'B' output will be dimmed for 50%.
// Set to 0 for completelty off (high impedance).
// Set to 255 for fully on.
// Special settings for the PWM speed:
// Set to -1 for not setting the PWM at all.
//
void motor_output (int output, int high_low, int speed)
{
int motorPWM;

switch (output)
{
case MOTOR1_A:
case MOTOR1_B:
motorPWM = MOTOR1_PWM;
break;
case MOTOR2_A:
case MOTOR2_B:
motorPWM = MOTOR2_PWM;
break;
case MOTOR3_A:
case MOTOR3_B:
```

```
487 motorPWM = MOTOR3_PWM;
488 break;
489 case MOTOR4_A:
490 case MOTOR4_B:
491 motorPWM = MOTOR4_PWM;
492 break;
493 default:
494 // Use speed as error flag, -3333 = invalid output.
495 speed = -3333;
496 break;
497 }
498
499 if (speed != -3333)
500 {
501 // Set the direction with the shift register
502 // on the MotorShield, even if the speed = -1.
503 // In that case the direction will be set, but
504 // not the PWM.
505 shiftWrite(output, high_low);
506
507 // set PWM only if it is valid
508 if (speed >= 0 && speed <= 255)
509 {
510 analogWrite(motorPWM, speed);
511 }
512 }
513 }
514
515 // ---------------------------------
516 // shiftWrite
517 //
518 // The parameters are just like digitalWrite().
519 //
520 // The output is the pin 0...7 (the pin behind
521 // the shift register).
522 // The second parameter is HIGH or LOW.
523 //
524 // There is no initialization function.
525 // Initialization is automatically done at the first
526 // time it is used.
527 //
528 void shiftWrite(int output, int high_low)
529 {
530 static int latch_copy;
531 static int shift_register_initialized = false;
532
533 // Do the initialization on the fly,
534 // at the first time it is used.
535 if (!shift_register_initialized)
536 {
537 // Set pins for shift register to output
538 pinMode(MOTORLATCH, OUTPUT);
539 pinMode(MOTORENABLE, OUTPUT);
540 pinMode(MOTORDATA, OUTPUT);
541 pinMode(MOTORCLK, OUTPUT);
542
```

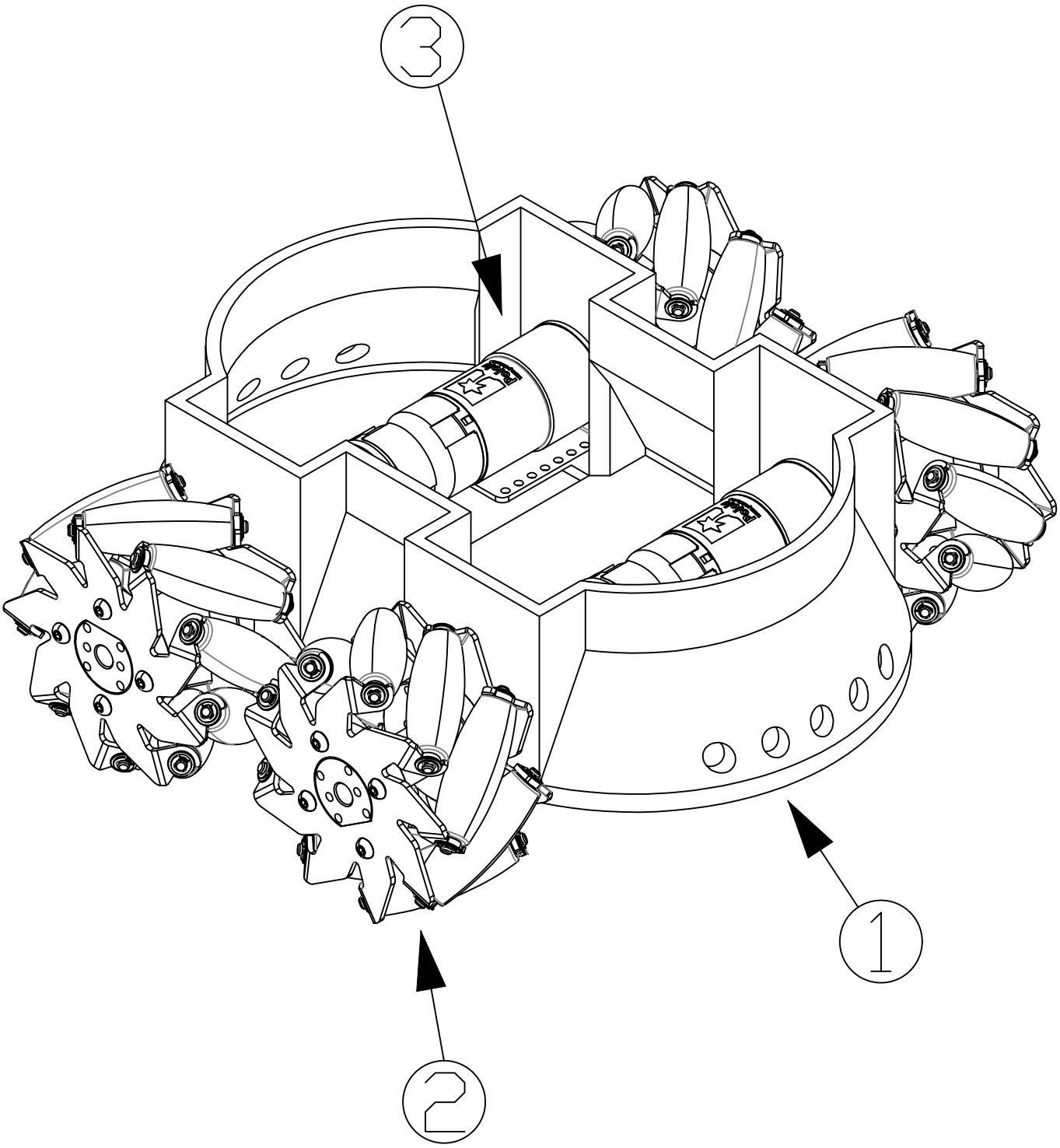
```
543 // Set pins for shift register to default value (low);
544 digitalWrite(MOTORDATA, LOW);
545 digitalWrite(MOTORLATCH, LOW);
546 digitalWrite(MOTORCLK, LOW);
547 // Enable the shift register, set Enable pin Low.
548 digitalWrite(MOTORENABLE, LOW);
549
550 // start with all outputs (of the shift register) low
551 latch_copy = 0;
552
553 shift_register_initialized = true;
554 }
555
556 // The defines HIGH and LOW are 1 and 0.
557 // So this is valid.
558 bitWrite(latch_copy, output, high_low);
559
560 // Use the default Arduino 'shiftOut()' function to
561 // shift the bits with the MOTORCLK as clock pulse.
562 // The 74HC595 shiftregister wants the MSB first.
563 // After that, generate a latch pulse with MOTORLATCH.
564 shiftOut(MOTORDATA, MOTORCLK, MSBFIRST, latch_copy);
565 delayMicroseconds(5); // For safety, not really needed.
566 digitalWrite(MOTORLATCH, HIGH);
567 delayMicroseconds(5); // For safety, not really needed.
568 digitalWrite(MOTORLATCH, LOW);
569 }
        570
```

Bibliografía

- Santiago Martínez, Rafael Sisto. Control y Comportamiento de Robots Omnidireccionales. Facultad de Ingeniería, Universidad de la República. Montevideo, Uruguay.
- Suárez Arriaga, A.E. y Sánchez Balpuesta, A.M. Plataforma Móvil Omnidirecional de Cuatro Llantas Suecas (Mecanum) en Configuración "AB". Facultad de Ingeniería, Universidad Nacional Autónoma de México. México.
- https://playground.arduino.cc/Main/AdafruitMotorShiel. Consultado el 1 de julio de 2018.
- https://www.pololu.com/product/2824. Consultado el 9 de julio de 2018.
- https://store.makeblock.com/maker-kits-STEM/100mm-mecanum-wheel-set-with-4mm-shaft-connector-spcc. Consultado el 9 de julio de 2018.
- https://store.arduino.cc/usa/arduino-due. Consultado el 9 de julio de 2018.
- https://www.prometec.net/motorshieldv1/. Consultado el 9 de julio de 2018.
- https://www.makeblock.es/productos/bateria_7.4/. Consultado el 9 de julio de 2018.
- https://www.prometec.net/bt-hc06/. Consultado el 9 de julio de 2018.

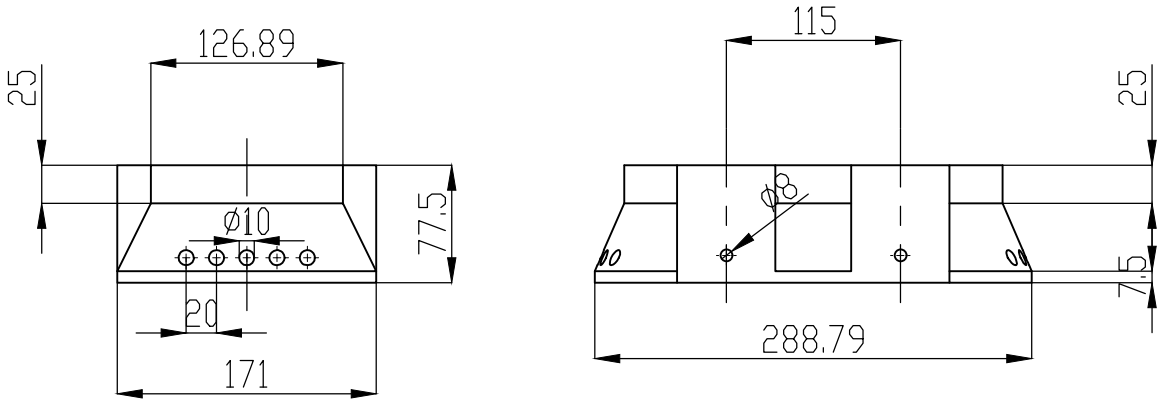
Planos

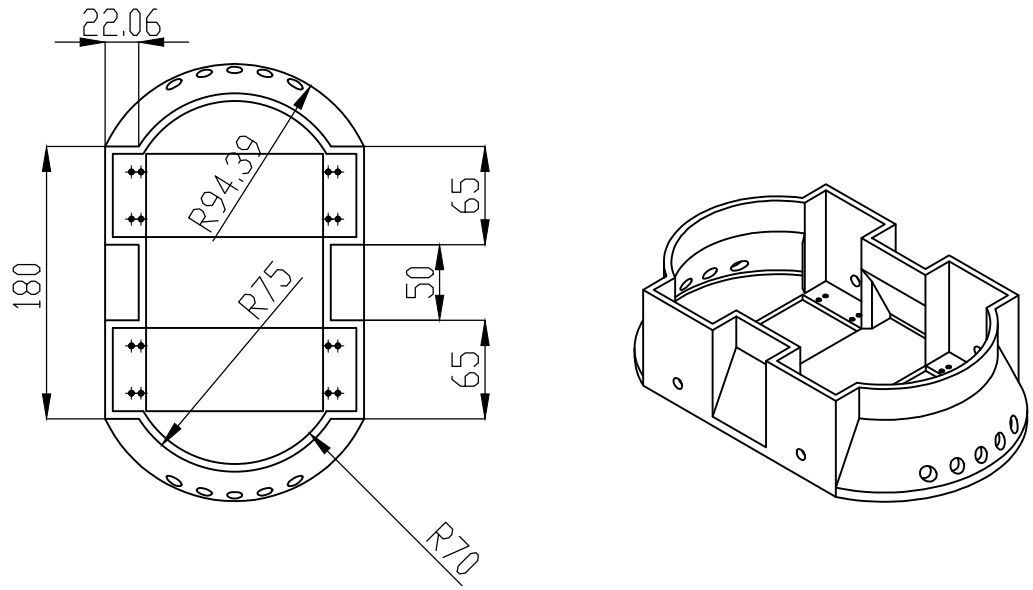
En las siguientes páginas se encuentran los planos del montaje y de sus componentes con sus dimensiones y puntos de ensamblaje.



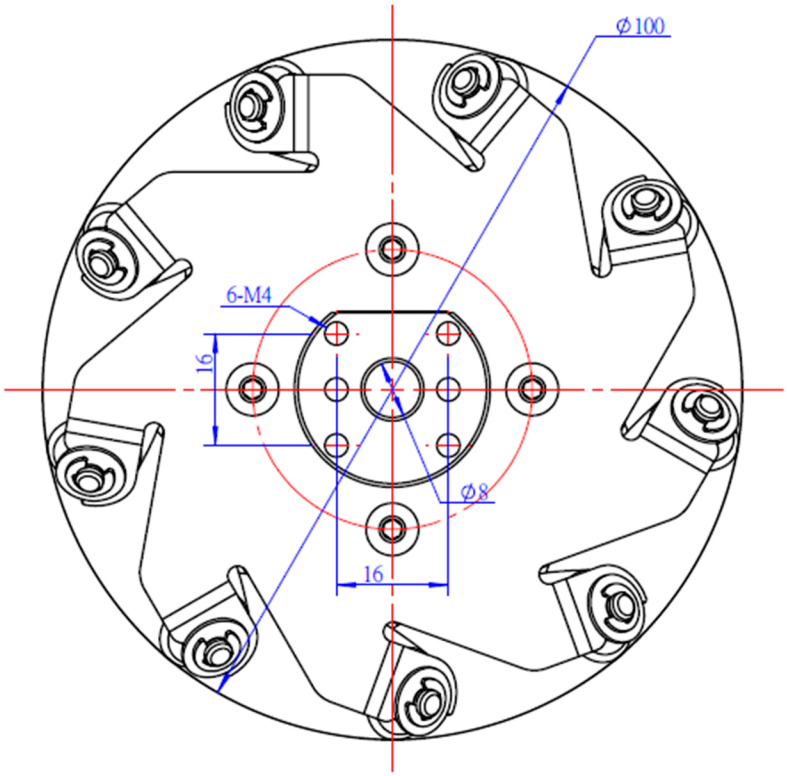
NÚMERO	COMPONENTE
1	CHASIS
2	RUEDAS MECANUM
3	MOTORES

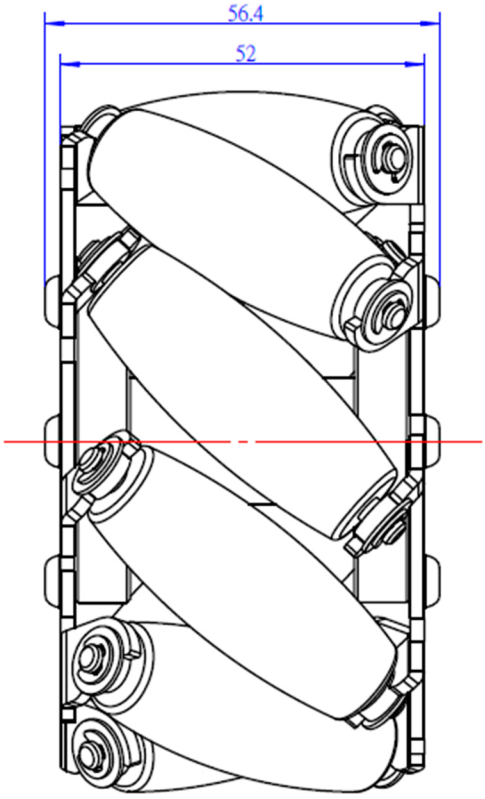
ESCALA:	FIRMA:	ENTIDAD	PIEZA
1:1		ETSID - UPV	MONTAJE
FECHA: 10/07/18			PLANO Nº: 01
DIBUJADO: Adrián Campos Siurana			





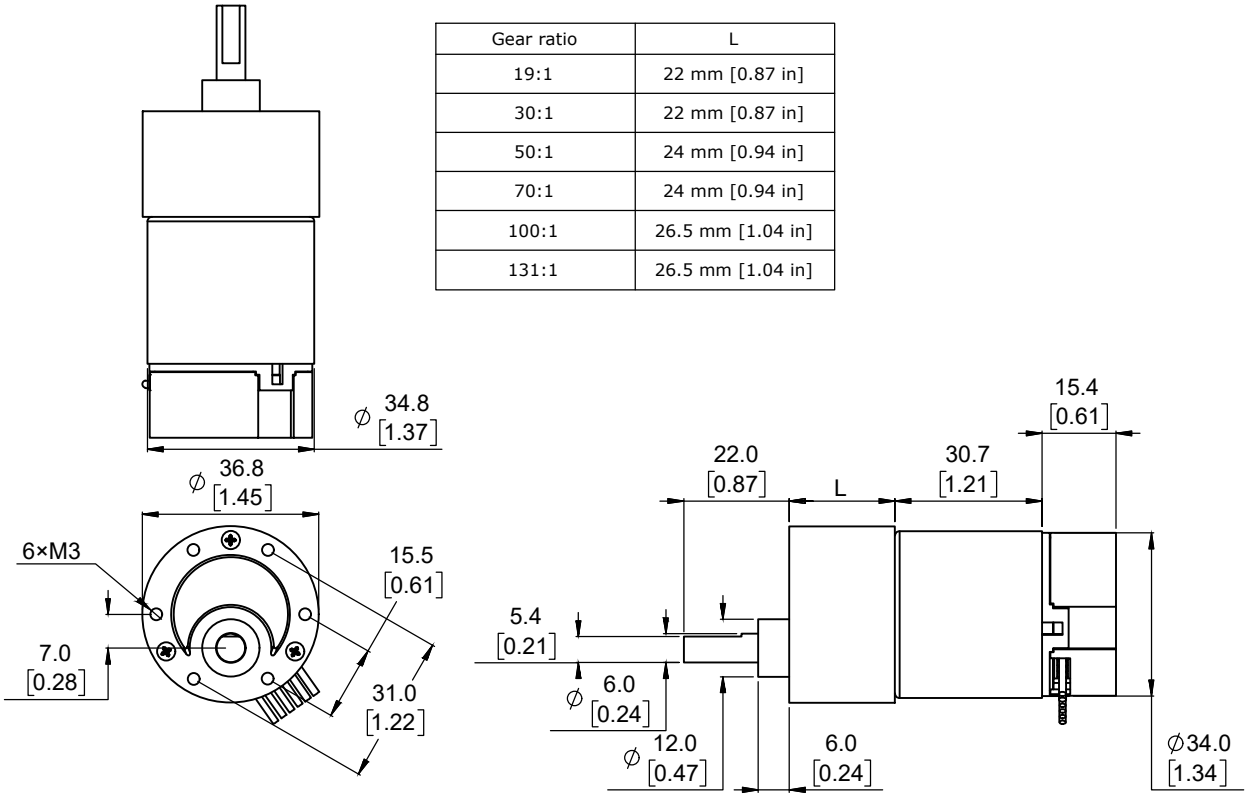
ESCALA:	FIRMA:	ENTIDAD	PIEZA
1:1		ETSID - UPV	CHASIS
FECHA:	10/07/18		
DIBUJADO		PLANO Nº:	02
Adrián Campos Siurana			





ESCALA:	FIRMA:	ENTIDAD	PIEZA
1:1		MAKEBLOCK	RUEDAS MECANUM
FECHA:		PLANO Nº:	03
DIBUJADO			

Gear ratio	L
19:1	22 mm [0.87 in]
30:1	22 mm [0.87 in]
50:1	24 mm [0.94 in]
70:1	24 mm [0.94 in]
100:1	26.5 mm [1.04 in]
131:1	26.5 mm [1.04 in]



ESCALA:	FIRMA:	ENTIDAD	PIEZA
1:1		POLOLU	MOTORES
FECHA:	27/07/17		
DESCRIPCIÓN		PLANO Nº:	04
37D mm Metal Gearmotors (with encoders)			