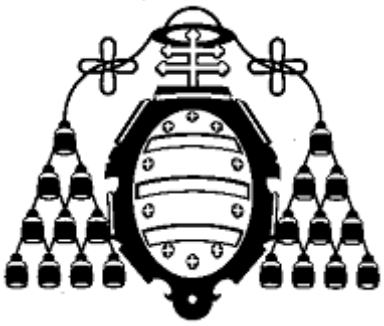


UNIVERSIDAD DE OVIEDO

CENTRO INTERNACIONAL DE POSTGRADO

MÁSTER EN INGENIERÍA MECATRÓNICA

TRABAJO FIN DE MÁSTER

DISEÑO, DESARROLLO E IMPLEMENTACIÓN DEL SISTEMA DE DESPLAZAMIENTO DE UN ROBOT DE INSPECCIÓN Y SUBSANACIÓN DE CHAPA GRUESA

JULIO 2019

Fernando Peña Cambón **Ignacio Álvarez García**

AGRADECIMIENTOS

Antes de comenzar el desarrollo de este Trabajo Fin de Máster, me gustaría agradecer tanto a Rafael Corsino González como a Ignacio Álvarez su apoyo durante todo el proceso de desarrollo de este proyecto, así como su ayuda y orientación durante mi estancia en el Área de Ingeniería de Sistemas y Automática, la cual me ha brindado la oportunidad de participar en este proyecto de investigación.

Al Departamento de Construcción e Ingeniería de Fabricación, por su colaboración en la producción de componentes del prototipo.

A Alberto García, profesor del máster, por su valiosa ayuda y su dedicación para llevar adelante los diseños.

También, agradecer a Sara López, Álvaro Fernández y Sara Roos-Hoefgeest su cooperación como compañeras/os de equipo, y por crear un buen ambiente de trabajo durante las prácticas.

Y, por supuesto, a mi familia y amigos, y, en especial, a María, por su apoyo incondicional en todo momento.

Con esto, doy comienzo al desarrollo de este Trabajo Fin de Máster.

RESUMEN

Este Trabajo Fin de Máster se encuentra desarrollado en el marco del proyecto de investigación "Investigación y desarrollo de robots de inspección y subsanación de imperfecciones en chapa gruesa", con referencia CN-18-011, desarrollado en el Área de Ingeniería de Sistemas y Automática, del Departamento de Ingeniería Eléctrica, Electrónica, de Computadores y Sistemas, de la Universidad de Oviedo.

En este trabajo se abarca el proceso completo de desarrollo e implementación de un sistema de desplazamiento para un robot, que tendrá como tareas la detección, evaluación y subsanación de imperfecciones en chapa gruesa metálica, tras su proceso de fabricación.

En primer lugar, se desarrolla una fase de investigación previa, de cara a conocer las tecnologías existentes para la obtención de sistemas móviles omnidireccionales. En ella se comprueban ejemplos comerciales, de investigación y patentes para realizar un estudio completo.

A continuación, se define un prototipo para su completo desarrollo, hasta alcanzar una solución física real. Para ello, se desarrolla cada una de las partes de ingeniería realizada -mecánica, electrónica, eléctrica, de software- en las que se detalla el procedimiento a seguir para la consecución de los objetivos que se establezcan.

Una vez completada la fase de diseño, se procede a la fabricación y selección de componentes, para su posterior montaje. Finalmente se desarrollan pruebas sobre el prototipo final para poder dar pie al desarrollo de los siguientes pasos en el proyecto de investigación.

PALABRAS CLAVE

Industria - Inspección - Investigación – Prototipo – Robótica - Subsanación

ÍNDICE GENERAL

1. ANTECEDENTES

Este Trabajo Fin de Máster surge de un proyecto de investigación titulado “Investigación y desarrollo de robots de inspección y subsanación de imperfecciones en chapa gruesa”, con referencia CN-18-011, desarrollado en el Área de Ingeniería de Sistemas y Automática, del Departamento de Ingeniería Eléctrica, Electrónica, de Computadores y Sistemas, de la Universidad de Oviedo.

En este proyecto de investigación, se propone el desarrollo de un equipo capaz de inspeccionar y subsanar errores en chapa gruesa, dentro de un entorno industrial. Este proceso, originalmente, era realizado por operarios/as de la planta industrial, lo que conlleva un largo proceso de inspección, con su posterior mecanizado, registro de las operaciones realizadas, etc para la realización del control de calidad tras la producción. Por ello, se busca desarrollar un equipo capaz de desplazarse de forma autónoma en el área de trabajo, de tal manera que pueda planificar su trayectoria en función de los errores detectados sobre el metal. Además, debe disponer de una herramienta de mecanizado capaz de llevar a cabo el trabajo de subsanación de las imperfecciones e impurezas de la superficie metálica. Se trata de una tecnología con alto grado de innovación, ya que no existen equipos en la actualidad capaces de desempeñar la totalidad de las tareas que este proceso de inspección requiere.

En la .Figura 1.1, se muestra una representación aproximada del escenario en el que se encontraría el robot en su lugar de trabajo, con la chapa de acero apoyada sobre el suelo. Esta chapa, de dimensiones aproximadas de 18 x 10 metros, será la superficie por la que el robot se desplaza autónomamente según su algoritmo de navegación, buscando nuevas imperfecciones, planificando cómo mecanizarlas, y arreglando las superficies afectadas. Además, el equipo será capaz de evitar todos aquellos obstáculos que puedan aparecer en su entorno, así como emplear medidas de seguridad en posibles casos de que pueda haber personal trabajando a su alrededor.

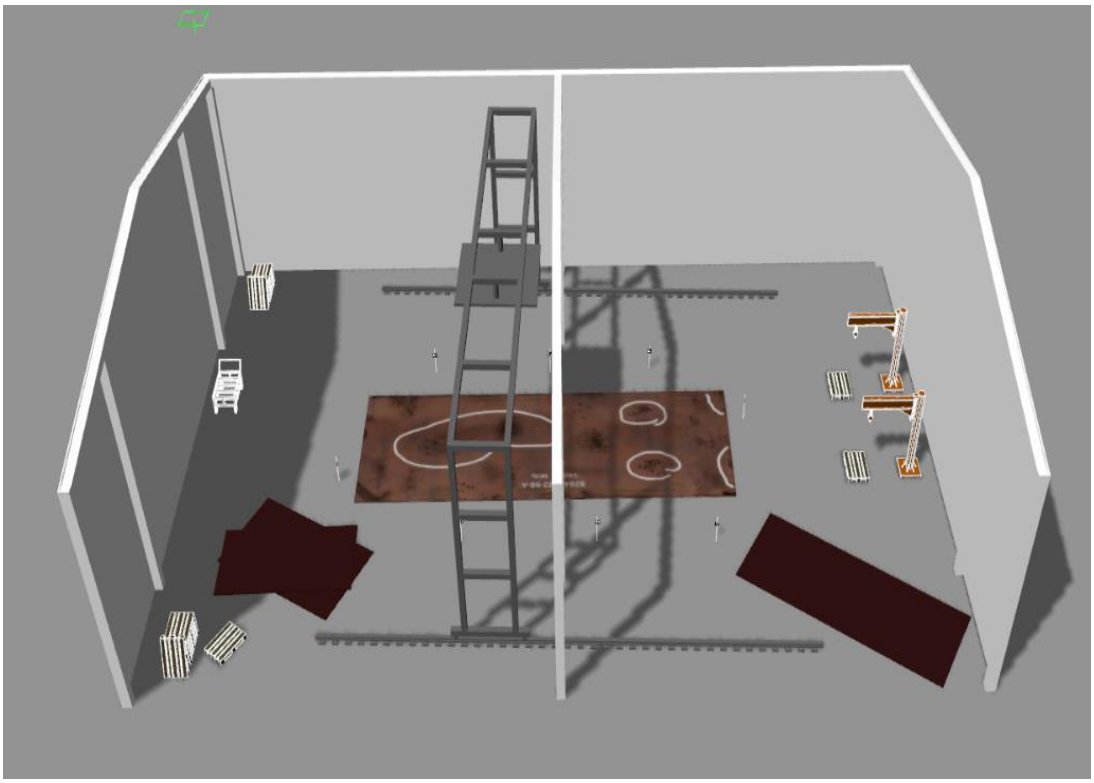


Figura 1.1. Modelo 3D de la nave industrial en la que se plantea el área de trabajo del robot.

El desarrollo del proyecto de investigación global se podría dividir en cuatro grandes ramas:

- Diseño e implementación del sistema de movimiento del prototipo.
- Diseño e implementación del sistema de mecanizado de las imperfecciones en la chapa.
- Diseño de los algoritmos de navegación en el entorno y planificación del mecanizado.
- Implementación de sistemas de percepción de errores en la chapa y del entorno del robot en el área de trabajo.

Concretamente, en este Trabajo Fin de Máster, se desarrollará únicamente el primer punto de la lista, teniendo como objetivo principal desarrollar una plataforma sobre la que se pueda montar el resto de subsistemas necesarios para el dispositivo, que sea capaz de satisfacer las condiciones del movimiento necesarias para la consecución de sus tareas sobre la chapa.

2. OBJETIVOS

El proceso de elaboración de este Trabajo Fin de Máster, como trabajo final de los estudios del Máster Universitario en Ingeniería Mecatrónica, conlleva una serie de objetivos:

- Aplicación de los conocimientos adquiridos durante la etapa formativa en ingeniería mecatrónica, tales como el diseño mecánico de prototipos mecatrónicos, el diseño electrónico y su implementación, el desarrollo de software para complementar el funcionamiento del prototipo, etc.
- Investigación sobre tecnologías de cinemática omnidireccional para el diseño del robot y la subsanación de errores en chapa gruesa.
- Desarrollo de un prototipo físico de un robot omnidireccional, desde su diseño conceptual hasta la ingeniería de detalle, incluyendo los cálculos necesarios para su correcto dimensionamiento mecánico.
- Implementación del modelo cinemático acorde al diseño omnidireccional seleccionado.
- Integración de aquella electrónica necesaria para el funcionamiento del dispositivo, el control de los diferentes subsistemas que puedan intervenir en él, etc.
- Implementación de un programa descentralizado que se encargue del control del movimiento del robot, pudiendo interaccionar con el resto de subsistemas a bordo del prototipo, a modo de caja negra.
- Diseño y dimensionamiento eléctrico del sistema, para garantizar un nivel de funcionamiento adecuado y una autonomía suficiente para los objetivos de diseño.
- Fabricación y montaje de un prototipo físico real.
- Realización de pruebas sobre un prototipo físico.
- Integración de todas las subtareas desarrolladas como partes de diferentes disciplinas de la rama de ingeniería, en un solo proyecto, como es característico dentro de la ingeniería mecatrónica.

En resumen, el objetivo de este proyecto multidisciplinar consiste en el desarrollo de un prototipo mecatrónico industrial, en el que se abarquen todos los objetivos de diseño establecidos,

3. ESTADO DEL ARTE

3.1. Clasificación de robots

La robótica es una disciplina de la ingeniería que se encuentra en continuo crecimiento con los numerosos avances tecnológicos que se han ido logrando a lo largo de la historia. Esta disciplina surge de la necesidad de crear máquinas que reduzcan el esfuerzo a la hora de realizar un trabajo, automatizar tareas repetitivas, minimizar el riesgo en labores que conlleven ciertos peligros, etc., en general, proporcionar una mayor comodidad para el ser humano.

La expansión de esta disciplina técnica y el continuo desarrollo de nuevas tecnologías es tal, que existen infinidad de criterios para la clasificación de robots, en función de su movimiento, la tarea a desempeñar, su morfología, sus grados de libertad, su tipo de control… Sin embargo, una clasificación pertinente para el tema que se desarrolla en este proyecto podría comenzar haciendo una distinción entre robots fijos y móviles, refiriéndose como fijos a aquellos que desempeñen su labor con una estructura fija, únicamente teniendo movimientos entre sus eslabones, mientras que los móviles hacen alusión a aquellos capaces de desplazarse bien sea por un terreno sólido, o un medio continuo como el aire o agua.


Figura 3.1 y Figura 3.2. Robot fijo (izquierda) y móvil (derecha)



Figura 3.3, Figura 3.4 y Figura 3.5. Robots móviles terrestre, aéreo y acuático, respectivamente.

Concretamente, en el grupo de los robots móviles terrestres, nos encontramos con diferentes sistemas de movimiento, según el mecanismo empleado para su consecución, como ruedas, articulaciones, orugas, etc. Sin embargo, el marco de este proyecto estará centrado en el desarrollo de robots móviles terrestres, dotados de movilidad mediante el empleo de ruedas.

Dentro del grupo de los robots con ruedas, la primera diferencia se encuentra al separar robots omnidireccionales y no omnidireccionales o no-holonómicos. Un robot se considera omnidireccional si es capaz de moverse en cualquier dirección del plano cartesiano, independientemente de su orientación, lo que, es decir, su bastidor no posee ninguna restricción de velocidad. Un ejemplo de movimiento omnidireccional sería el de un carrito de supermercado. Por contraposición, los no-holonómicos son aquellos que son incapaces de moverse en cualquier dirección, sin realizar movimientos adicionales para su orientación, como podría ser el caso de un coche.

3.2. Clasificación de sistemas omnidireccionales

Existen muy diversas formas de obtener una motricidad omnidireccional, según el mecanismo y/o la construcción que se emplee en el desarrollo del robot, atendiendo a múltiples configuraciones cinemáticas [1].

El caso más básico que se puede encontrar es una rueda sencilla, como la de un monociclo (Figura 3.6), sobre la que se aplica un momento sobre el eje de la rueda que produzca un giro sobre su único punto de apoyo. Sin embargo, esta solución, a pesar de satisfacer los criterios de esta clasificación, no tiene una utilidad práctica para el proyecto.

Figura 3.6. Monociclo.

De manera similar, se pueden emplear apoyos esféricos para garantizar la omnidireccionalidad del movimiento. Sin embargo, teniendo en cuenta las dificultades constructivas y de funcionamiento que esto supone, no es una de las categorías más empleadas para este tipo de robots móviles, aun existiendo ciertos casos, como el que se muestra en la Figura 3.7.

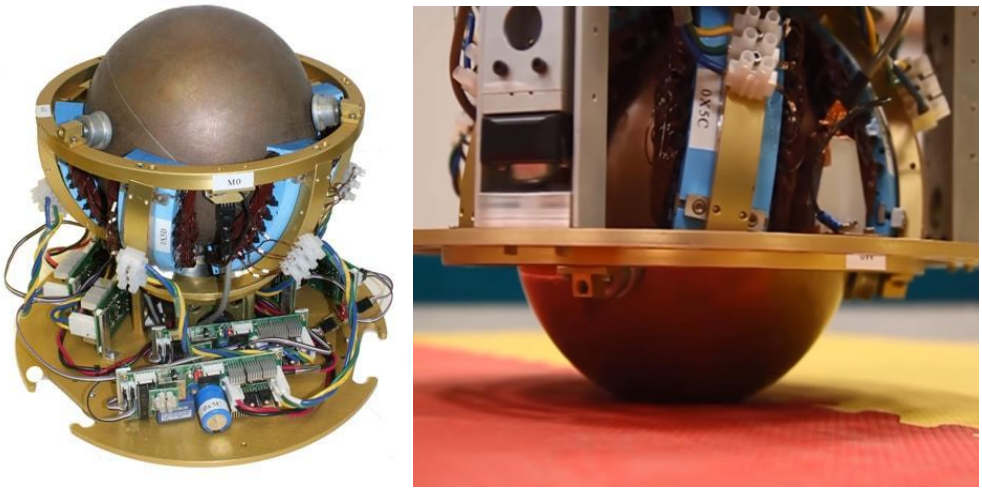


Figura 3.7. SIMbot, robot de apoyo esférico.

Otro caso, de los más extendidos en la actualidad, es el empleo de ruedas omnidireccionales, “Swedish wheels” u “Omniwheels”, como se muestran en la Figura 3.8. Este tipo de ruedas tienen la particularidad de poseer rodillos móviles a lo largo de su perímetro. Dentro de esta categoría, existen dos configuraciones claramente diferenciadas: en primer lugar, existen ruedas que poseen los rodillos cuyos ejes de rotación se encuentran tangentes a la circunferencia de la rueda. Pueden disponer de una o más hileras de rodillos, según el modelo.

Figura 3.8. Ruedas omnidireccionales.

Con este tipo de ruedas, resulta necesario emplear una configuración específica de montaje, ya que los rodillos tienen un movimiento pasivo y el movimiento global del robot resulta de la combinación de los giros de las ruedas portantes de los rodillos. En la Figura 3.9 se muestra una de las configuraciones más habituales.

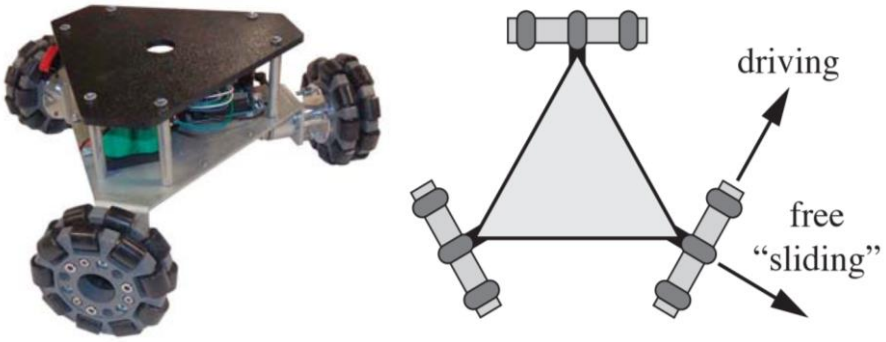


Figura 3.9. Montaje triangular de ruedas omnidireccionales.

Por otra parte, existen las llamadas ruedas "Mecanum" (véase Figura 3.10), que emplean un sistema similar a las anteriores, con la particularidad de que los rodillos se encuentran inclinados con un cierto ángulo alrededor del perímetro de la rueda. En este caso solo disponen de una hilera de rodillos. El funcionamiento de estas ruedas es idéntico y, únicamente difieren en la cinemática global de las ruedas y de su configuración de montaje.

Figura 3.10. Rueda mecanum.

La configuración más extendida de este tipo de ruedas es empleando cuatro de ellas, distribuidas de forma rectangular, como es el caso de la empresa KUKA Robotics, en la Figura 3.11 o la de Robotnik, en la Figura 3.12.

Figura 3.11. Plataforma omnidireccional del Youbot (KUKA Robotics).

Figura 3.12. SUMMIT-XL STEEL, de la empresa Robotnik.

Por otro lado, existe otra solución técnica para la obtención de un sistema omnidireccional, que consiste en el empleo de ruedas motrices que sean capaces de orientarse sobre sí mismas. En la actualidad, se pueden encontrar múltiples patentes y soluciones comerciales para la creación de sistemas de este tipo, en las que el principio de funcionamiento es común a la mayoría, divergiendo cada una en un mecanismo o diseño diferente para la consecución de la rueda orientable. Algunos ejemplos son los de la empresa Neobotix (véase Figura 3.13)

Figura 3.13. "Omni-drive module" de la empresa Neobotix.

En todas ellas, la rueda es conducida por dos motores, los cuales se encargan de la tracción motora de la propia rueda y de la orientación del eje de la rueda.

4. ESPECIFICACIONES DE DISEÑO

Como en el diseño de cualquier prototipo mecatrónico, resulta esencial establecer una serie de criterios como punto de partida, de cara a la obtención de unos resultados acordes con lo deseado en el inicio de este complejo proceso.

Por este motivo, en la Tabla 4.1, se recogen los principales criterios de partida para la orientación en el diseño del robot.

Velocidad máxima	1 m / s
Autonomía	1-2 horas
Alimentación	Baterías DC
Superficie de navegación	Acero
Pendiente máxima	20º
Aceleración máxima	0.1 m / s^2
Masa del prototipo máxima	20-30 kg

Tabla 4.1. Especificaciones de diseño del prototipo.

Estos parámetros servirán para el desarrollo del prototipo en diferentes fases de cálculo y/o definición, aunque puedan surgir otros nuevos a lo largo del desarrollo de este Trabajo Fin de Máster.

5. DISEÑO DEL PROTOTIPO

En este capítulo se abarcan todos los procedimientos seguidos para la consecución de un diseño completo del prototipo de robot de subsanación de chapa, haciendo hincapié en todos los aspectos de gran relevancia, como el diseño mecánico, electrónico, eléctrico y de software.

5.1. Diseño mecánico

Dentro de la dimensión mecánica del proyecto, resulta necesario seguir unas pautas estructuradas y coherentes para alcanzar un diseño final satisfactorio. Por ello, se detallarán en los siguientes subapartados cada uno de los aspectos más importantes en el desarrollo del prototipo.

5.1.1. *DISEÑO CONCEPTUAL*

Tras la realización de un estudio de las numerosas posibilidades que existen en la actualidad para la creación de un robot de movimiento omnidireccional, como se detalla en el Capítulo 3, Estado del Arte, surge la necesidad de tomar una serie de decisiones acerca del concepto del robot, para poder ir desarrollando la fase de ingeniería en posteriores pasos.

Las dos principales soluciones que se consideran viables para su desarrollo en un prototipo son las ruedas omnidireccionales de rodillos o *mecanum* (Figura 3.8 y Figura 3.10), y las ruedas con tracción propia y auto orientables (Figura 3.13).

Estas últimas, surgen como idea inicial de partida, pues existen múltiples maneras de obtener un mecanismo que permita la distribución de la motricidad en dos movimientos independientes: la tracción de la rueda y la orientación de su eje de rotación. Además, por la posibilidad de fabricarse mediante tecnologías de fabricación aditiva, dentro de las capacidades del proyecto en el que se enmarca este trabajo, resulta ser una opción interesante ya que requiere pocos recursos para obtener un prototipo final. Además, al poseer una rueda con neumático, puede presentar un buen comportamiento frente al deslizamiento, al presentar un buen agarre.

Por estos motivos, se procede a un boceto preliminar de dicho concepto, para su posterior materialización. Con los primeros bocetos (véase Figura 5.1), se pudieron detectar los principales problemas que podría conllevar la realización de dicho diseño en el caso de su construcción física:

- Por ejemplo, la necesidad de que los motores se encuentren en el bastidor fijo del módulo, ya que resulta imperativo como criterio de diseño que el módulo pueda orientar la rueda girando indefinidamente, por lo que no se podrían ubicar elementos con cables en la parte móvil de éste.
- Como consecuencia directa del problema anterior, se requiere una transmisión del motor de tracción más larga y compleja para poder impulsar la rueda.
- A pesar de que pudiera ser solucionado a partir de la programación del módulo, existiría un movimiento diferencial entre el giro de la rueda y la orientación de ésta, por lo que habría que introducir una compensación de la diferencia de giro, lo que podría inducir a una menor precisión del movimiento.
- Otro problema, relacionado con la transmisión del movimiento, consiste en la necesidad de respetar una distancia entre ejes para un movimiento seguro sin colisiones de la rueda motriz, por lo que se necesitaría recurrir a transmisiones flexibles que garanticen seguridad en este sentido.

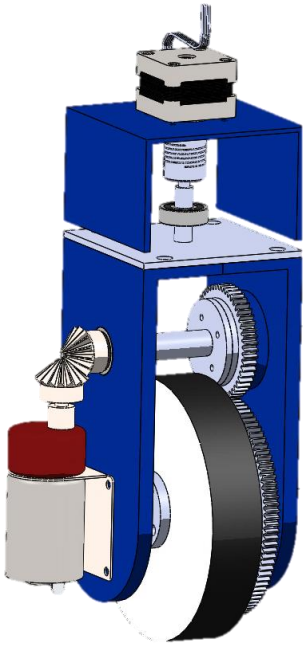


Figura 5.1. Primer diseño conceptual de rueda auto-orientable.

Tras un primer análisis del boceto preliminar, se procede al desarrollo de un modelo más refinado, de cara a satisfacer los criterios iniciales, más la solución de problemas surgidos en el desarrollo previo. Como se muestra en la Figura 5.2, se presenta un modelo más adecuado, en el que se puede apreciar cómo la motricidad del módulo viene desde el bastidor de éste, y se deriva una transmisión hasta la rueda. Así mismo, se puede observar que el eje de giro de la horquilla no coincide verticalmente con el centro del punto de contacto con la rueda, lo cual supondría una singularidad dentro del cálculo de la cinemática del módulo.

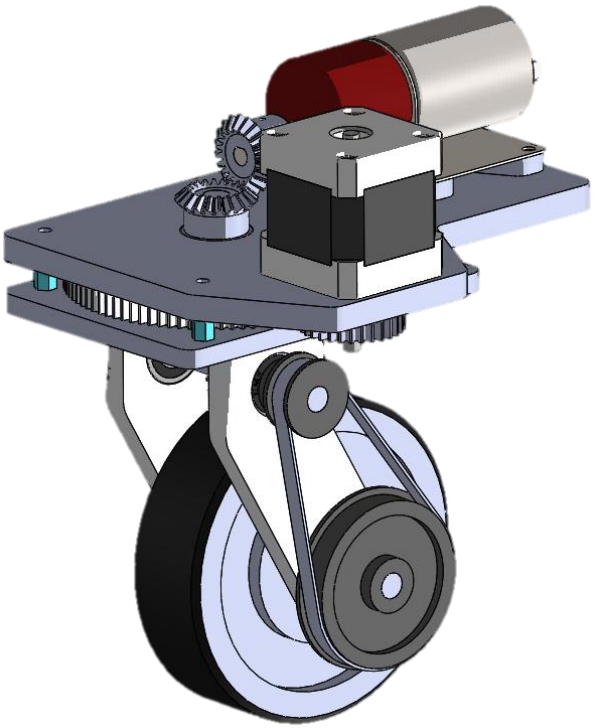


Figura 5.2. Modelo inicial con correcciones.

Tras el continuo refinado del modelo, se llega a un modelo como el mostrado en la Figura 5.3, el cual se encuentra orientado a ser fabricado mediante prototipado rápido. Sin embargo, la necesidad de optimizar costes y flexibilizar el diseño, conducentes a la recursión a las tecnologías de fabricación aditiva, suponen una pérdida en otros aspectos como la calidad en el diseño en cuanto a precisiones y tolerancias, acabados, resistencia mecánica, etc., lo cual conduciría a problemas en las transmisiones, necesidad de ajustar tolerancias para garantizar distancias entre agujeros y ejes, entre otros, y llevaría un largo proceso de ajuste en el diseño.

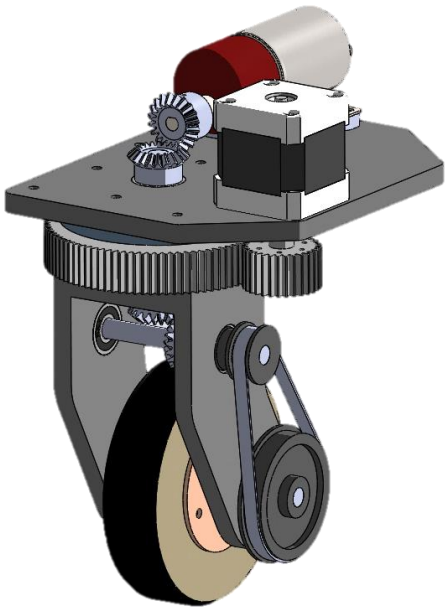


Figura 5.3. Diseño final de rueda motriz auto-orientable.

Por otro lado, el cliente para el cuál se desarrolla el Por los motivos expuestos, finalmente se descarta este concepto, a pesar de su viabilidad técnica, ya que no se podrían garantizar resultados acordes a los criterios de partida de manera totalmente satisfactoria, a excepción de que se recurriesen a técnicas de fabricación por arranque de viruta en materiales metálicos.

A razón de estos hechos, junto con la preferencia del cliente para el cual se desarrolla el proyecto, la solución que se adopta finalmente es la del empleo de ruedas mecanum (Figura 5.4), ya que también tienen un gran campo de aplicación a nivel industrial y la viabilidad en su implementación es completa, además de que existe un gran volumen de bibliografía haciendo uso de esta tecnología, a pesar de que presenten un menor agarre de las ruedas. Por todo esto, se desarrollará una plataforma que emplee cuatro ruedas de este tipo, para simplificar el diseño.

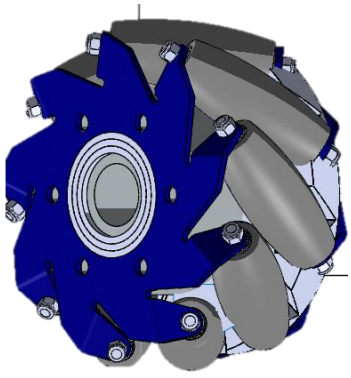


Figura 5.4. Modelo 3D de rueda mecanum.

5.1.2. *DEFINICIÓN CINEMÁTICA*

Ahora bien, una vez se define la tecnología de las ruedas omnidireccionales, resulta necesario conocer el comportamiento cinemático del modelo, para posteriores pasos en el diseño.

Una manera de plantear el modelo cinemático es a partir de un análisis individual de la cinemática de cada rueda, y finalmente establecer un modelo global del conjunto, tal y como se propone en la bibliografía consultada [2]. Para comenzar, se plantea un sistema de coordenadas cartesiano local sobre el centro del robot, como se propone en la Figura 5.5.

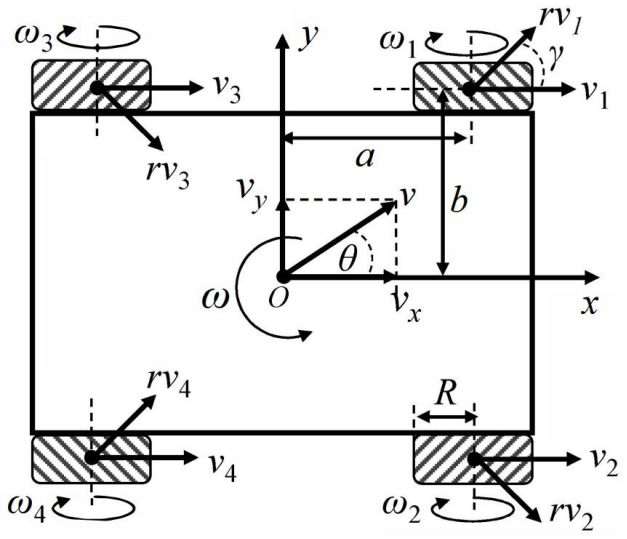


Figura 5.5. Gráfico del modelo cinemático de la plataforma con cuatro ruedas Mecanum [2].

Con el fin de facilitar la interpretación del gráfico, se recogen los significados de cada una de las magnitudes que aparecen, en la Tabla 5.1.

Magnitud	Significado
x, y	Ejes de coordenadas
R	Radio de la rueda Mecanum
v, v_x, v_y	Velocidades cartesianas globales de la plataforma
ω	Velocidad angular del robot
a, b	Distancias cartesianas de la rueda al centro del robot
θ	Ángulo del vector de velocidad global respecto de x
γ	Ángulo de posición del rodillo
$ω_i$	Velocidad de giro de la rueda “i”
v_i	Velocidad lineal de la rueda “i”
rv_i	Velocidad lineal del rodillo de la rueda “i”

Tabla 5.1. Variables que intervienen en la cinemática del robot.

Con las variables definidas, se puede proceder a la definición cinemática del robot.

En primer lugar, la velocidad global de la plataforma se puede descomponer en sus coordenadas cartesianas del siguiente modo, según las ecuaciones (5.1) y (5.2):

$$v_x = v\ cos\,\theta \quad (5.1)$$

$$v_y = v\ \sin\theta \quad (5.2)$$

Respecto a los componentes de la velocidad de cada rueda, se puede establecer una ecuación para cada eje, según las distancias al centro del robot y los ángulos de los rodillos, de cada rueda. Teniendo en cuenta que los conjuntos de las medidas son:

$$a_i = \{a, a, -a, -a\} \quad (5.3)$$

$$b_i = \{b, -b, b, -b\} \quad (5.4)$$

$$\gamma_i = \left\{\frac{\pi}{4}, -\frac{\pi}{4}, -\frac{\pi}{4}, \frac{\pi}{4}\right\} \quad (5.5)$$

Para:

$$i = \{1,2,3,4\} \quad (5.6)$$

Cabe destacar que, en la ecuación (5.5), ya se han incluido los valores numéricos, ya que, en prácticamente todos los modelos comerciales, el ángulo de los rodillos se monta a 45º.

La descomposición, en x e y, por lo tanto, será:

$$v_i + rv_i \cos\gamma_i = v_x - b_i\,\omega \quad (5.7)$$

$$rv_i \sin\gamma_i = v_y + a_i\omega \quad (5.8)$$

Con las ecuaciones (5.7) y (5.8), se puede sustituir empleando la relación trigonométrica de la tangente, para llegar a:

$$v_i = v_x - b_i\omega - \frac{v_y + a_i\omega}{\tan\gamma_i} \quad (5.9)$$

Como simplificación, dado que aparece la tangente del ángulo γ, se pueden sustituir por los valores resultantes de los ángulos existentes. Es decir:

$$\tan\gamma_i = \{1, -1, -1{,}1\} \quad (5.10)$$

Como consecuencia de este desarrollo matemático, se obtienen cuatro ecuaciones, una para la velocidad lineal de cada rueda, que posteriormente se puede transformar en la velocidad de giro a partir del radio de la rueda:

$$v_1 = v_x - v_y - a\omega - b\omega \quad (5.11)$$

$$v_2 = v_x + v_y + a\omega + b\omega \quad (5.12)$$

$$v_3 = v_x + v_y - a\omega - b\omega \quad (5.13)$$

$$v_4 = v_x - v_y + a\omega + b\omega \quad (5.14)$$

Finalmente, se puede llegar a la formulación matricial, para la obtención de las velocidades de giro de cada rueda.

$$\begin{bmatrix}\omega_1\\ \omega_2\\ \omega_3\\ \omega_4\end{bmatrix} = \frac{1}{R}\begin{bmatrix}1 & -1 & -(a+b)\\ 1 & 1 & (a+b)\\ 1 & 1 & -(a+b)\\ 1 & -1 & (a+b)\end{bmatrix}\begin{bmatrix}v_x\\ v_y\\ \omega\end{bmatrix} \quad (5.15)$$

Con la ecuación (5.15), queda completamente definida la cinemática inversa del robot, sin tener en cuenta la orientación de éste. Este modelo cinemático será posteriormente implementado en el diseño del control del prototipo desde un microcontrolador u ordenador a bordo.

5.1.3. *DIMENSIONAMIENTO Y SELECCIÓN DE COMPONENTES COMERCIALES*

Dado que, para poder diseñar el prototipo físico resulta imprescindible incluir los motores que impulsarán el prototipo, es necesario hacer un predimensionamiento de los motores a utilizar, para posteriormente proceder al diseño físico de los utillajes y componentes anexos a éstos.

Para realizar los cálculos, se tendrán en cuenta los criterios de partida fijados en el Capítulo 4.

En primer lugar, se determinará la velocidad de funcionamiento. Para ello, se partirá de una rueda supuesta de 100 mm de diámetro, ya que se trata de una medida muy empleada en las ruedas mecanum comerciales.

Una primera intuición, podría ser el cálculo de la velocidad nominal de la rueda a partir de la velocidad lineal:

$$\omega_{Rueda} = \frac{v_{robot}}{R_{Rueda}} = \frac{1\ \frac{m}{s}}{0{,}05\ m} = 20\ \frac{rad}{s} \sim 191\ rpm \quad (5.16)$$

Sin embargo, dado que la cinemática inversa es conocida y fácilmente implementable, es posible evaluar la velocidad del conjunto de ruedas para cualquier dirección del vector velocidad, lo que se amplía el rango de velocidades comprobadas.

Por ello, se diseña un script en MATLAB (véase Anexo I) para parametrizar las velocidades de todas las ruedas en función del ángulo "θ", del vector de velocidad, para una velocidad "v" de 1 m/s, constante. Los resultados se muestran en forma de gráfico en la Figura 5.6.

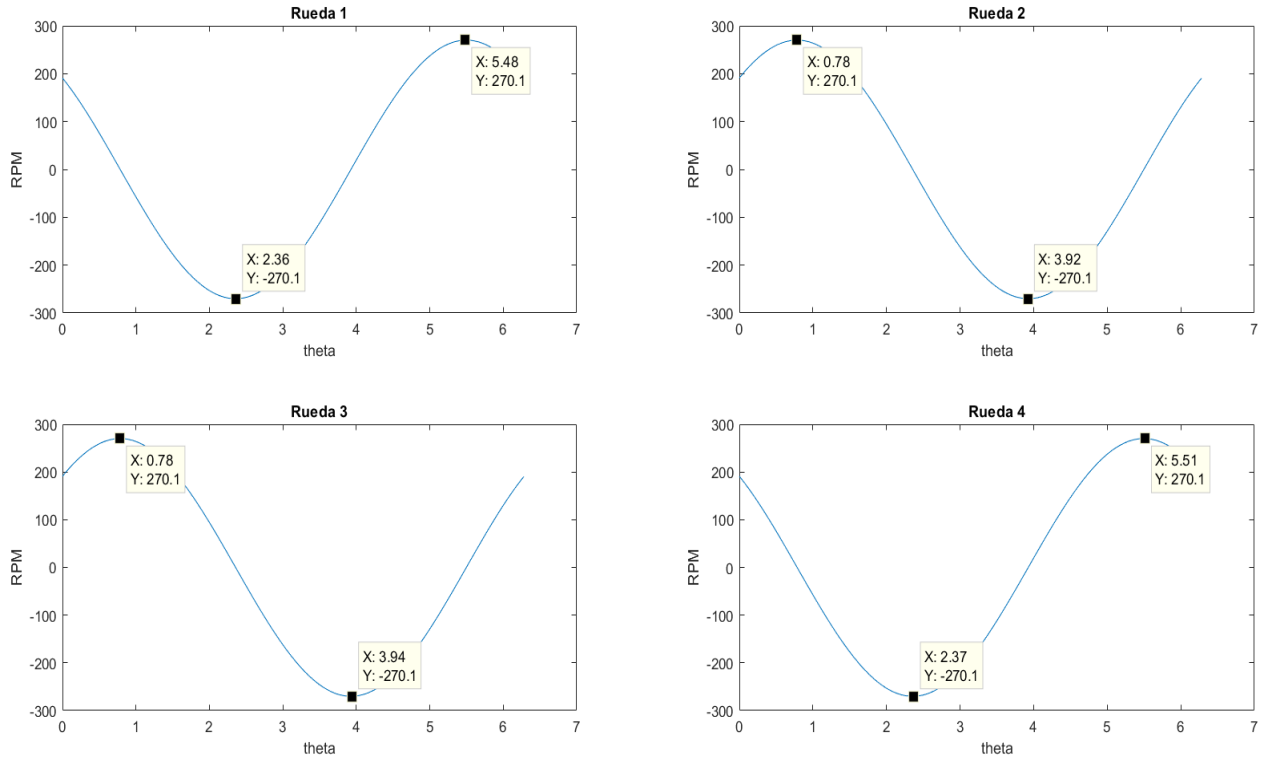


Figura 5.6. Velocidades angulares de las ruedas, en rpm, en función del ángulo del vector de velocidad del robot, en radianes.

Como se puede comprobar, todas las ruedas siguen un perfil de velocidad similar, teniendo todas en común un máximo y mínimo de velocidad de valor absoluto 270 rpm. Este valor, levemente mayor al calculado en la ecuación (5.16), servirá de punto de partida para la selección de los motores comerciales que se empleen.

Por otra parte, resulta fundamental estimar el par necesario para mover cada rueda, ya que necesita ser capaz de garantizar un par en servicio suficiente para sobrevenir cualquier situación que se contemple.

Para calcular el par necesario, se tendrán en cuenta las aceleraciones máximas estimadas, la masa del prototipo estimada, la pendiente del terreno máxima y el radio de las ruedas, tal como queda propuesto en la ecuación (5.17).

$$\mathrm{T} = \frac{(a_{max} + g * \sin\alpha) * m * R}{n_{motores}} \qquad (5.17)$$

Donde:

- a_{max} es la aceleración máxima del robot.
- g es la aceleración por la gravedad.
- α es la pendiente máxima por vencer por el robot.
- m es la masa total del robot.
- R es el radio de la rueda mecanum.
- $n_{motores}$ es el número de motores que impulsan al robot.

Sustituyendo en la ecuación (5.17) los valores propuestos al comienzo de este desarrollo, se puede obtener un par de:

$$\mathrm{T} = \frac{\left(0{,}1\,\frac{m}{s^2} + 9{,}81\,\frac{m}{s^2} * \sin\frac{20º * \pi}{180º}\right) * 30\ kg * 0{,}1\ m}{4} = 1{,}295\ N * m \qquad (5.18)$$

Basándose en un criterio conservador, se podría incluir una eficiencia del 70% en la transmisión del motor, para garantizar mayor seguridad en el servicio, dado que no se contempla el modelo dinámico de las ruedas mecanum en este cálculo.

$$T_{seguridad} = \frac{T}{70\%} = 1{,}85\ N * m \qquad (5.19)$$

Con esto, se podría determinar una potencia de partida para la selección del motor, mediante el siguiente cálculo:

$$P = T_{seguridad} * \omega_{Rueda} = 1.85\ N * 28{,}28\ \frac{rad}{s} = \ 52{,}3\ W \qquad (5.20)$$

De este modo, los motores quedan completamente dimensionados y se puede proceder a la búsqueda de los modelos comerciales que, en este caso, se seleccionarán del distribuidor “Elmeq” [3]. Los parámetros de dimensionamiento del motor se reflejan en la Tabla 5.2.

Velocidad nominal	270 rpm
Par	1,85 N
Potencia	52,3 W

Tabla 5.2. Resumen de los cálculos para el dimensionamiento de los motores del robot.

Concretamente, resultan de interés motores eléctricos de corriente continua, ya que el robot es alimentado por baterías. Se descarta el empleo de motores paso a paso por el riesgo de salto de pasos que se pueda producir en su funcionamiento, dando lugar a movimientos imprecisos. Este

problema se acentúa, sobre todo, si tiene que operar a bajas velocidades, como ocurre en los casos de parada y arranque.

En el catálogo del distribuidor citado, se pueden encontrar motores de corriente continua y motores de tipo "*brushless*", los cuales incorporan sensores de efecto Hall que permiten el cierre de un lazo de control para la posición y velocidad, lo que supone una gran ventaja al no necesitar la adición de encóders u otros sensores para tal función.

Es por este motivo que la búsqueda se centrará en la gama de motores "*brushless*", de tal manera que se seleccionará un juego de motor y etapa reductora para satisfacer las necesidades de diseño de este proyecto.

El motorreductor seleccionado, en este caso, es el KF65 / BG42x30, que se muestra en la Figura 5.7. Este motorreductor tiene una potencia de 65W, suficientes para la solicitación en servicio del proyecto. Los datos nominales de funcionamiento, para la salida de la etapa reductora, se muestran en la Tabla 5.3.

Figura 5.7. Motorreductor brushless KF65 / BG42x30, de ELMEQ.

Velocidad nominal	100,42 rpm
Par	4,44 N
Potencia	65 W

Tabla 5.3. Datos del motor seleccionado.

Como se puede apreciar, se selecciona un motor de menor velocidad que la solicitada en el inicio, ya que se trata de un requisito impuesto de manera preliminar, pero, al tratarse de un prototipo a nivel experimental, no resulta tan imprescindible como la garantía de par suficiente para movilizar el robot.

Todas las características y detalles técnicos del motor y de la caja reductora se encuentran en las hojas de características anexas a esta memoria, en el Anexo III.

En lo referente a la selección de las ruedas, efectivamente se escogen unas acordes al diámetro inicialmente establecido, de 100 mm. Estas ruedas son seleccionadas del catálogo de Digi-key [4], de la marca Makeblock. Sus características también se incluyen en las hojas de características anexas, en el Anexo IV. Como detalle, hay que destacar que se debe escoger un juego de cuatro ruedas, dos a izquierdas y dos a derechas, para poder implementar la cinemática correctamente. Su montaje y distribución se realizará según el diagrama mostrado en la Figura 5.5, para seguir fielmente el modelo cinemático descrito.

Figura 5.8. Rueda Mecanum izquierda, de la marca Makeblock.

Como última comprobación, ya conocidos los motores y ruedas a emplear, se puede realizar una aproximación del orden de velocidad en el que se desplace el robot, haciendo una aproximación de la cinemática directa en base al modelo cinemático inverso planteado. Esto se hace de la siguiente manera, a partir de la ecuación (5.15):

$$\begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix} = \left[\frac{1}{R} \begin{bmatrix} 1 & -1 & -(a+b) \\ 1 & 1 & (a+b) \\ 1 & 1 & -(a+b) \\ 1 & -1 & (a+b) \end{bmatrix} \right]^{-1} \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix} \qquad (5.21)$$

Como encontrar los máximos y mínimos de velocidad puede convertirse en un largo y tedioso proceso, para un cálculo aproximado, se realiza una estimación suponiendo que todos los motores giran a su velocidad nominal, y se determina el módulo del vector velocidad.

Para realizar el cálculo, se emplea el script de MATLAB anteriormente mencionado, para realizar el cálculo matricial planteado en la ecuación (5.21), realizando una serie de modificaciones.

Como resultado, se obtiene una velocidad de 0,52 m/s, lo que da idea de que el orden de magnitud en el que se encuentra el desplazamiento del robot es de la mitad del planteado inicialmente.

5.1.4. *DISEÑO CONSTRUCTIVO*

Teniendo una mayor noción del funcionamiento del robot, y conociendo algunos de los componentes que lo conforman, se puede comenzar con el boceto de la plataforma que da estructura al robot.

En primer lugar, cabe decir que, dado que se trata de un prototipo a nivel de investigación y desarrollo, y no de un modelo para producción industrial, no se tendrán en cuenta aspectos como la optimización del diseño para sistemas productivos de gran volumen tanto en el diseño para la fabricación y montaje como la reducción de costes.

Para obtener la estructura, se emplean barras normalizadas de aluminio de 45 mm, de longitudes de 0,5 m, colocadas como se muestra en la Figura 5.9.

Figura 5.9. Cuadro del bastidor del prototipo.

Para su unión, se emplearán elementos conectores en ángulo recto, colocados en las esquinas interiores del cuadro. Dicha pieza se encontrará detallada en el Anexo X, de planos.

Sobre esta estructura, se colocará una chapa de aluminio, que sirve para dar soporte al resto de componentes que irán a bordo del prototipo. La chapa será perforada de tal manera que se puedan ir colocando todos los componentes sobre ella, distribuidos de forma optimizada.

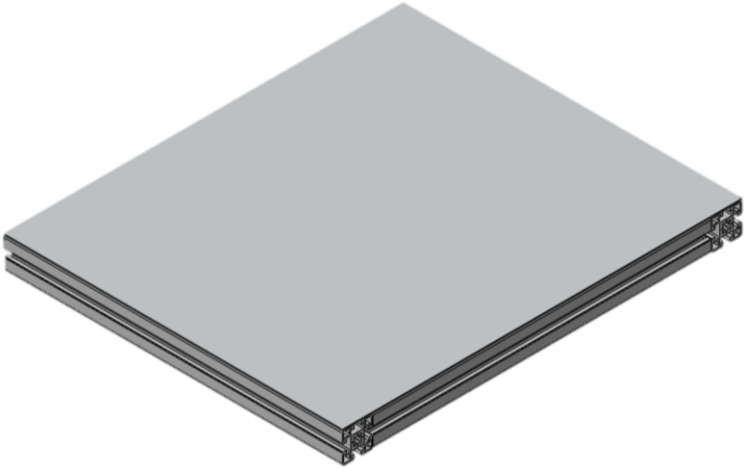


Figura 5.10. Estructura del prototipo con una chapa de aluminio, conformando la base del modelo.

Para unir los motores y las ruedas a la plataforma, es necesario diseñar elementos de unión, tal y como se muestra en la Figura 5.11. Esta escuadra se diseña para poder acoplar el motor, respetando la distancia entre agujeros y su eje. Además, cuenta con otros agujeros en la otra cara para poder ser atornillado a la estructura, aprovechando las ranuras de las barras de aluminio. El motor se unirá mediante tornillos a la escuadra, como se muestra en la Figura 5.12.

El conjunto, se ubicará en las cuatro esquinas de la estructura, como se expone en la Figura 5.13. Todas las piezas y uniones quedan reflejadas de manera detallada en los planos del conjunto del prototipo.

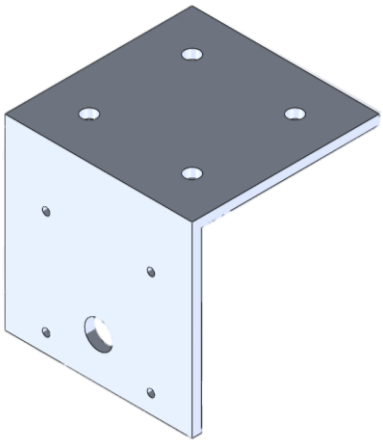


Figura 5.11. Escuadra de unión del motor a la plataforma.

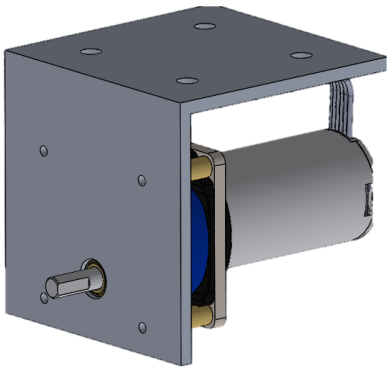


Figura 5.12. Montaje del motor en la escuadra.

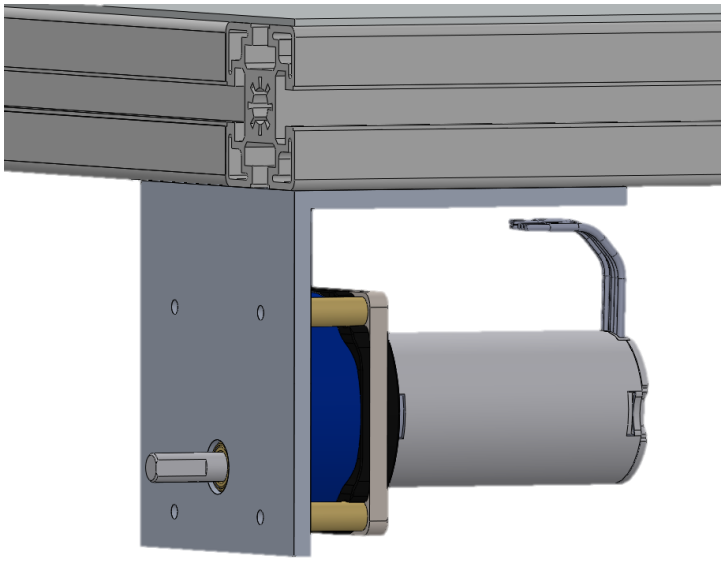


Figura 5.13. Montaje del motor y la escuadra en la esquina del robot.

Para la unión de la rueda al motor, se diseña un acoplamiento específico (véase Figura 5.14), que se puede atornillar a la rueda, y bloquear en la parte ranurada del eje del motor.

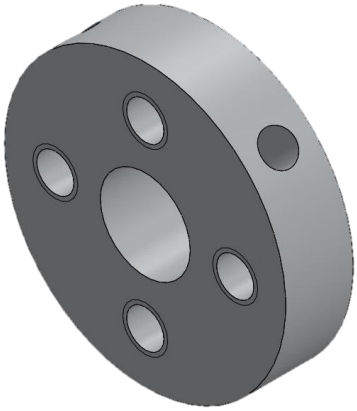


Figura 5.14. Acoplamiento.

De esta manera, se restringen los movimientos relativos de la rueda respecto del eje del motor a través de tornillos prisioneros, por lo que se asegura un funcionamiento sin movimientos anómalos.

En la Figura 5.15, se refleja el modelo 3D del bastidor del prototipo, con las ruedas mecanum y el utillaje de los motores. En ella se resume la apariencia básica del modelo, sin entrar en detalle en aspectos como la electrónica a bordo, las baterías, etc., que se documentarán en los documentos de planos anexos, con información sobre las dimensiones.

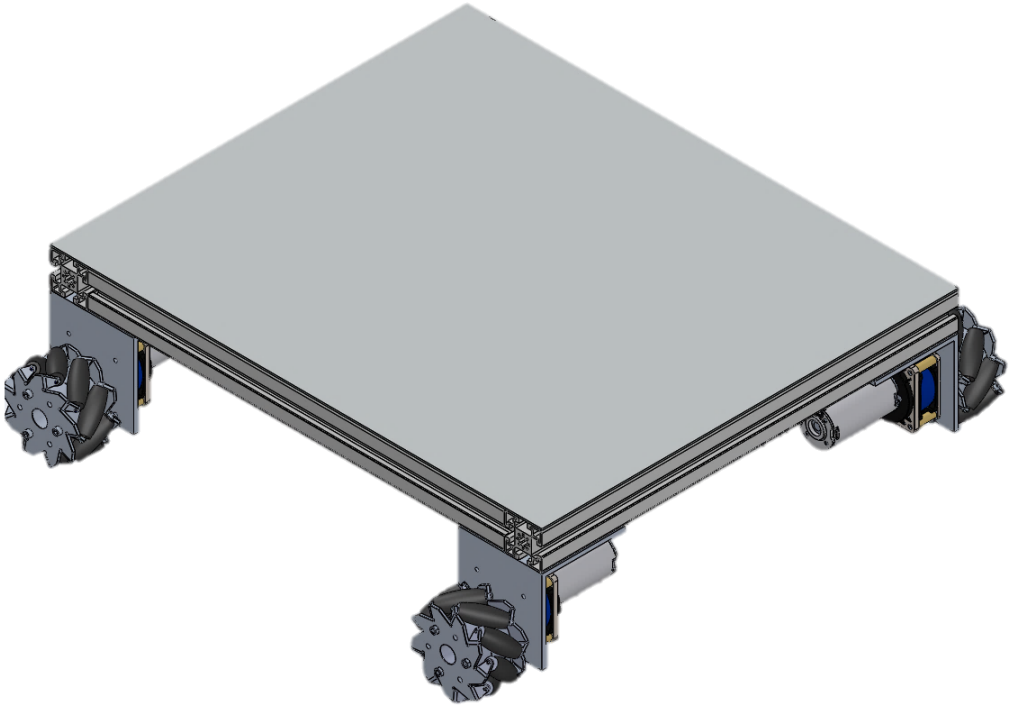


Figura 5.15. Montaje de la plataforma omnidireccional.

5.2. Diseño electrónico

Una vez concluido el epígrafe sobre el diseño mecánico del dispositivo, es momento de detallar la rama electrónica y de control del proyecto. Esta abarca aspectos como la metodología empleada para el diseño y control de los motores, así como otros aspectos de interés relacionados con el control global del robot.

5.2.1. CONTROL DE LOS MOTORES

Como en cualquier aplicación en la que se empleen motores eléctricos, resulta necesario implementar una interfaz electrónica que permita la comunicación y control de estos, de tal manera que se pueda desarrollar un programa en un PC o microcontrolador que se encargue de la gestión de los motores.

Por este motivo, se debe hacer una búsqueda de dispositivos comerciales que den solución a este problema. Para tal fin, una de las consultas a distribuidores se ha hecho a la empresa TRINAMIC [5], la cual es referente en tecnologías de este tipo y dispone de un amplio catálogo de tarjetas de control. Concretamente, disponen de una sección de tarjetas de control para motores brushless de corriente continua (BLDC).

De cara a determinar qué modelos se adecúan más al prototipo, un factor clave es el suministro eléctrico necesario. Dado que los motores seleccionados en el epígrafe anterior trabajan a un nivel de 24V, es conveniente que las tarjetas sean compatibles con ese mismo nivel de tensión, para minimizar el número de etapas de adaptación de tensión desde las baterías, minimizando así las pérdidas.

BLDC cDriver™ Modules

Control and drive up to six axes standalone or from a host over UART or CAN interfaces. We significantly simplify the programming with our TMCL or CANopen motion control languages.

Name	Axes	Motor Supply	Phase Current, RMS	a/b/n incremental	analog	CAN	Hall	RS232	RS485	USB	TMCL	CANopen	break before make	short detection	slope control	Current Sens
TMCM-1630-2C	1	14.5...55V	10.0A	✓	✓	✓	✓	✓			✓		✓	✓	✓	✓
TMCM-1630-4U	1	14.5...55V	10.0A	✓	✓		✓		✓	✓	✓		✓	✓	✓	✓
TMCM-1633	1	14.5...48V	10.0A	✓	✓	✓	✓	✓				✓	✓	✓	✓	✓
TMCM-1640	1	14.5...28.5V	5.0A	✓	✓		✓		✓	✓	✓		✓	✓	✓	✓

Figura 5.16. Listado de tarjetas para el control de motores brushless de TRINAMIC.

Sin embargo, todas las tarjetas tienen niveles de alimentación similares (véase Figura 5.16). Por consiguiente, hay que ceñirse a otros criterios para la selección del modelo.

Un criterio determinante, por ejemplo, podría ser el protocolo de comunicación, que resulta de interés que sea compatible con el formato serie USB, para poder realizar todas las pruebas necesarias en diferentes equipos, previamente a la aplicación final. Con ello, la búsqueda se reduce a dos tarjetas, las cuales únicamente difieren en los niveles de alimentación. Por ello, se determina que la placa más adecuada para este fin es el modelo TMCM-1630-4U (véase Figura

5.17), dado que puede proporcionar una mayor corriente de pico en caso de demandas puntuales, por lo que se asegura un mejor servicio del motor.

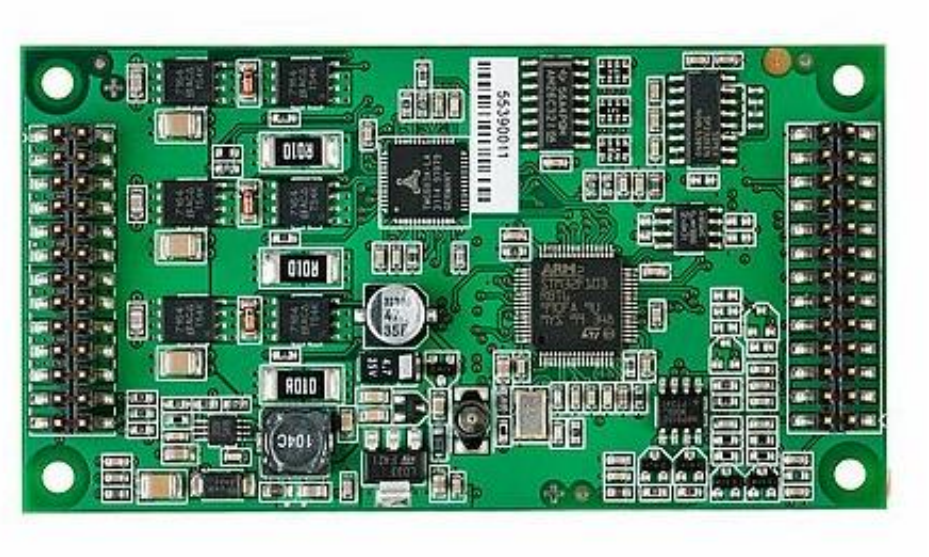


Figura 5.17. Vista superior de la tarjeta seleccionada, el modelo TMCM-1630-4U.

Esta tarjeta permite la retroalimentación de los sensores de efecto Hall, de los que sí se dispone en los motores BLDC seleccionados, por lo que se puede realizar un lazo de control para la velocidad y/o la posición de éstos, como se indica en el diagrama de la Figura 5.18.

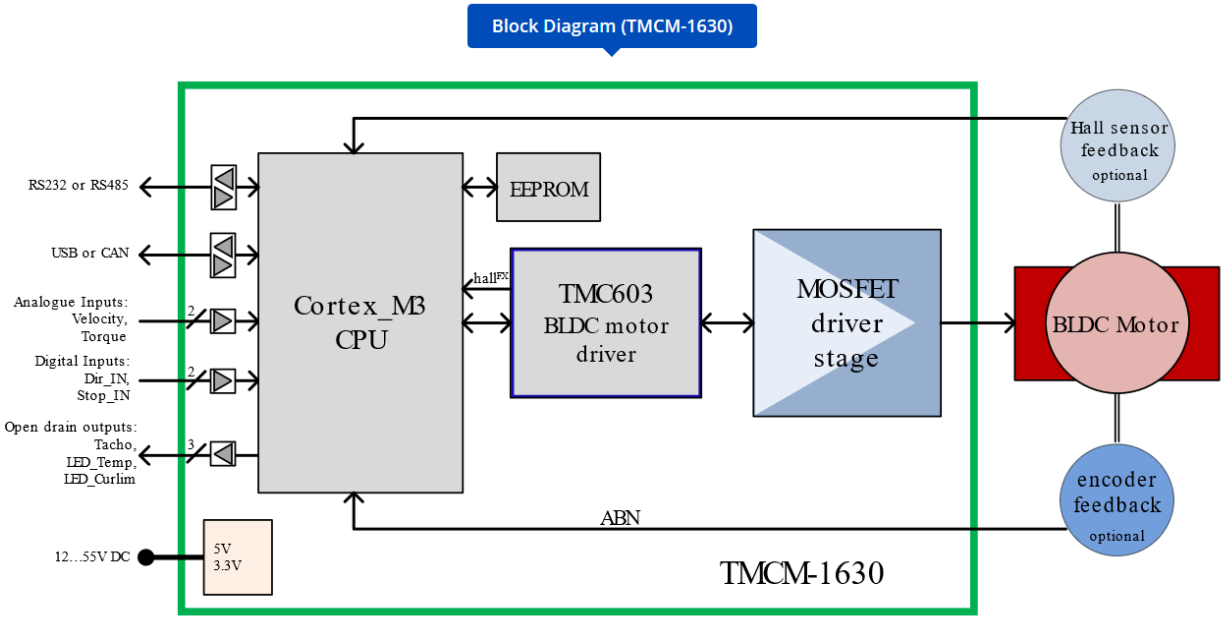


Figura 5.18. Diagrama de bloques interno de la tarjeta de control.

Este “driver” posee un microcontrolador interno que se encarga de la gestión a bajo nivel de las comunicaciones, la interfaz física del motor, etc., lo cual reduce el trabajo de diseño. Además, posee un lenguaje de programación propio, el TMCL, que permite enviar comandos de manera estructurada al motor, creando así una “estandarización” en las comunicaciones, de manera independiente a las conexiones externas.

Este protocolo de comunicaciones será el mismo que se emplea en la sección del proyecto de investigación encargado de la subsanación de imperfecciones en la chapa, para el control de equipos encargados de las herramientas para el mecanizado.

Además, con el fin de facilitar las conexiones, dentro del catálogo de TRINAMIC existen kits de evaluación de módulos de control, que permiten una fácil conexión, ahorrando así el diseño de una PCB para la aplicación final. Concretamente, para este módulo, existe el kit BB-163X (Figura 5.19).

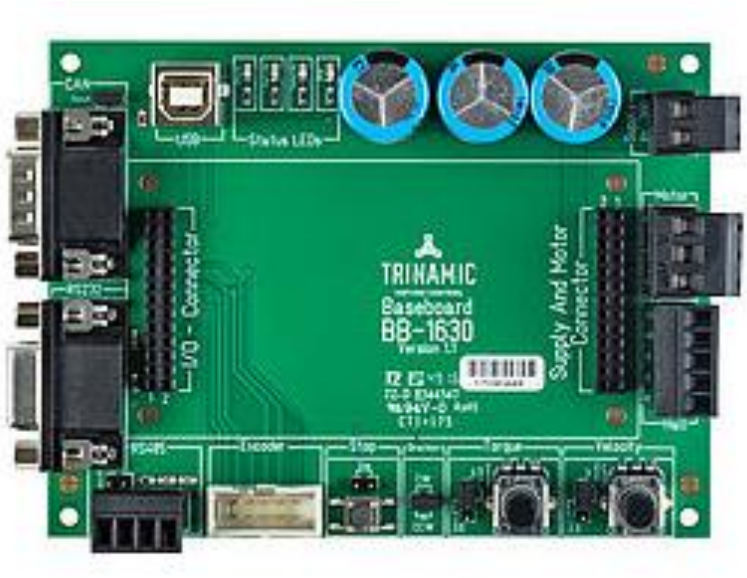


Figura 5.19. Placa de evaluación BB-163X, compatible con el módulo TMC-1630.

Esta placa, de formato “Plug & Play” dispone de conectores compatibles con el módulo TMC-1630 para su inserción directa. Además, cuenta con diferentes conectores para la alimentación del sistema, la conexión de los bobinados del motor, la conexión del cableado de los sensores de efecto Hall, y los múltiples modos de comunicación disponibles para el driver.

En la Figura 5.20, se muestra un diagrama en el que se pretende señalar la interfaz física de los módulos seleccionados para su integración con el resto de prototipo. Estas conexiones se realizan para cada motor, por lo que, en total, se deben integrar 4 placas y su interfaz en el diseño.

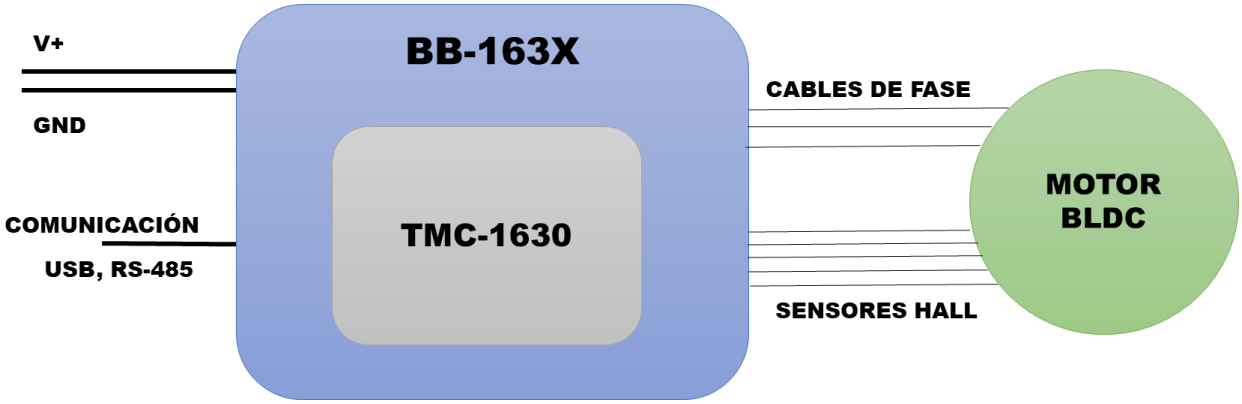


Figura 5.20. Diagrama de las conexiones del módulo TMC-1630 en la placa BB-163X.

Con la integración de estas tarjetas de control se logra simplificar el diseño electrónico, facilitando el control de los motores, sin necesidad de realizar un diseño del lazo de control, que ya se integra dentro del funcionamiento de estos dispositivos.

5.2.2. *PROGRAMACIÓN DE LOS MOTORES*

Como ya se había mencionado, las tarjetas de control disponen de un lenguaje de programación propio, independiente del protocolo de comunicación que se utilice con ellas. Este lenguaje consiste en una serie de comandos predefinidos, para los que existe una correspondencia numérica en el intérprete que llevan incorporado.

Para comprender de forma más detallada el funcionamiento de este sistema de comandos, TRINAMIC facilita una IDE (véase Figura 5.21) que permite el control en tiempo real del motor conectado a la tarjeta. Además, cuenta con una interfaz de desarrollo para programación en TMCL, que permite la depuración de errores en el código y la grabación de éste en el microcontrolador que porta la tarjeta del driver.

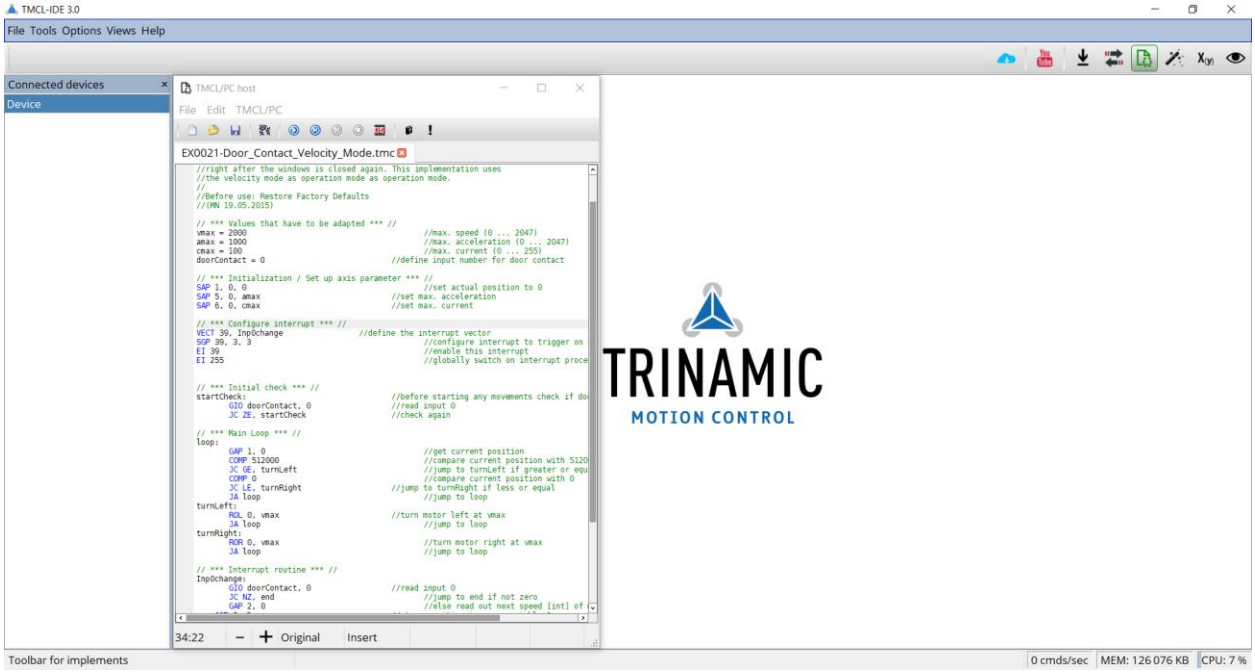


Figura 5.21. IDE de TRINAMIC para Windows.

Sin embargo, para esta aplicación en concreto, existe una opción que resulta de gran interés para el desarrollo del prototipo. Esto es, concretamente, la opción de enviar comandos en tiempo real, de tal manera que, en lugar de grabar los comandos en el microcontrolador, es posible enviarlos mediante comunicación serie USB o RS-485.

De este modo, es posible crear un programa principal en otro dispositivo, de manera que la tarjeta de control se encuentra en una capa inferior en el control del prototipo, y se encarga únicamente de procesar la información que le llega, y proporcionar las respuestas necesarias en cada caso.

Los comandos, como se muestran en la Figura 5.22, se estructuran en un paquete de 9 bytes, de tal manera que cada grupo de bytes se corresponde con un elemento del comando: la dirección, la instrucción, los parámetros del comando, y un último byte de comprobación.

Bytes	Meaning
1	Module address
1	Command number
1	Type number
1	Motor or Bank number
4	Value (MSB first!)
1	Checksum

Figura 5.22. Formato binario de los comandos TMCL.

Tras el envío del comando, el módulo enviará una respuesta en la que se recoge información como el estado, es decir, si el comando se ha enviado correctamente, los valores de parámetros propios de la respuesta, en caso de tener información de interés, y, de nuevo, el byte de comprobación. Esta respuesta se encuentra estructurada de una manera análoga a la de el comando (véase la Figura 5.23), por lo que la implementación en el programa general resulta más sencilla.

Bytes	**Meaning**
1	Reply address
1	Module address
1	Status (e.g. 100 means *no error*)
1	Command number
4	Value (MSB first!)
1	Checksum

Figura 5.23. Formato binario de las respuestas a comandos TMCL.

El listado de comandos completo y su información complementaria se encuentra recogido en la hoja de características del firmware de la tarjeta TMC-1630, que sirve de apoyo para el desarrollo del programa de manejo de la plataforma omnidireccional.

5.2.3. *CONTROL DEL PROTOTIPO*

Habiendo escogido la interfaz electrónica que gestionará el manejo de los motores, es necesario incluir un dispositivo, jerárquicamente superior, que se encargue del control global del prototipo, y a su vez sea capaz de comunicarse con las tarjetas de control de los motores.

Para el desarrollo del prototipo, en el inicio, se parte de un PC sobre el que se ejecuta un sistema operativo basado en UNIX, concretamente, Ubuntu 16.04 (Figura 5.24). Este consta de puertos USB con los que podrá implementarse el programa de comunicación serie con los dispositivos de TRINAMIC.

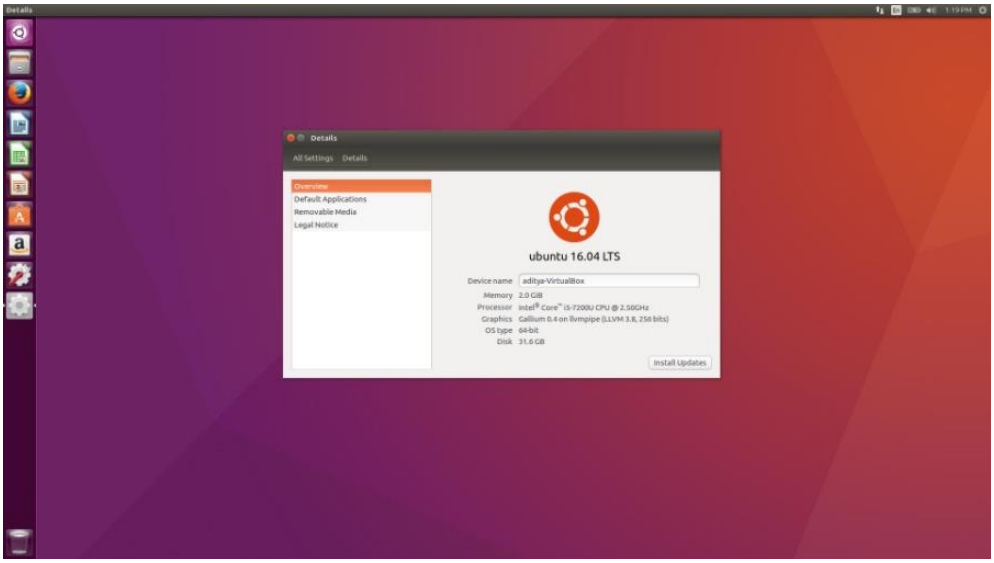


Figura 5.24. Imagen del escritorio del sistema operativo Ubuntu 16.04.

No obstante, con el fin de minimizar el espacio y peso del dispositivo, el programa principal deberá ser gestionado por un PC que sea capaz de ir a bordo de la plataforma, sin necesidad de realizar grandes cambios en el programa de control. Es por ello, que una de las opciones más idóneas es el uso de una Raspberry Pi (Figura 5.25), ya que consiste en un PC de bajo coste y tamaño, que funciona con sistemas operativos basados en UNIX, lo cual lo convierte en un dispositivo con grandes similitudes al PC empleado para el desarrollo inicial.

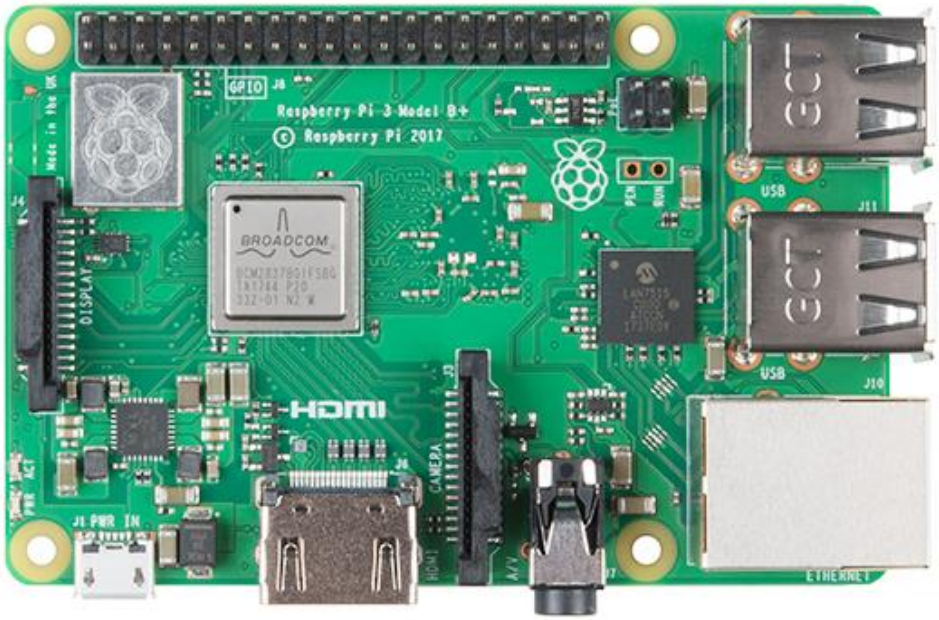


Figura 5.25. Vista superior de una Raspberry Pi 3B+.

Dado que el diseño del programa será muy similar tanto en el PC como en la Raspberry Pi, es posible desarrollar un programa que permita el trabajo con los cuatro motores de manera simultánea.

Como se indica en las hojas de características de la tarjeta TCM-1630, es posible comunicarse con agentes externos bien por protocolo USB o mediante una conexión RS-485, como se muestra en la Figura 5.26 y en la Figura 5.27.

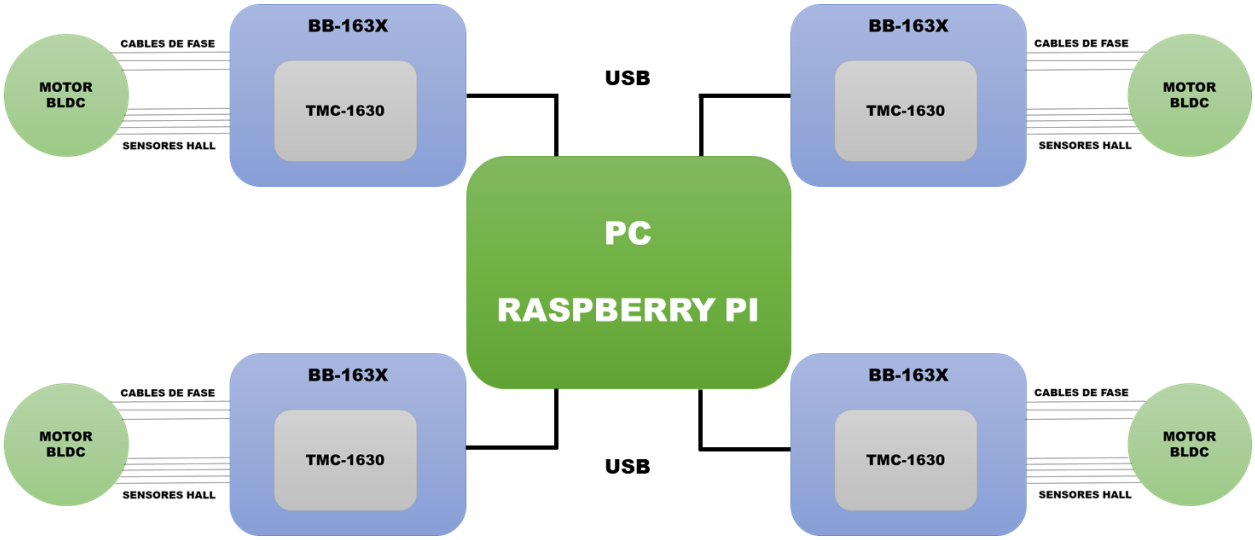


Figura 5.26. Diagrama de conexiones de los motores con el dispositivo de control empleando conexiones en formato USB.

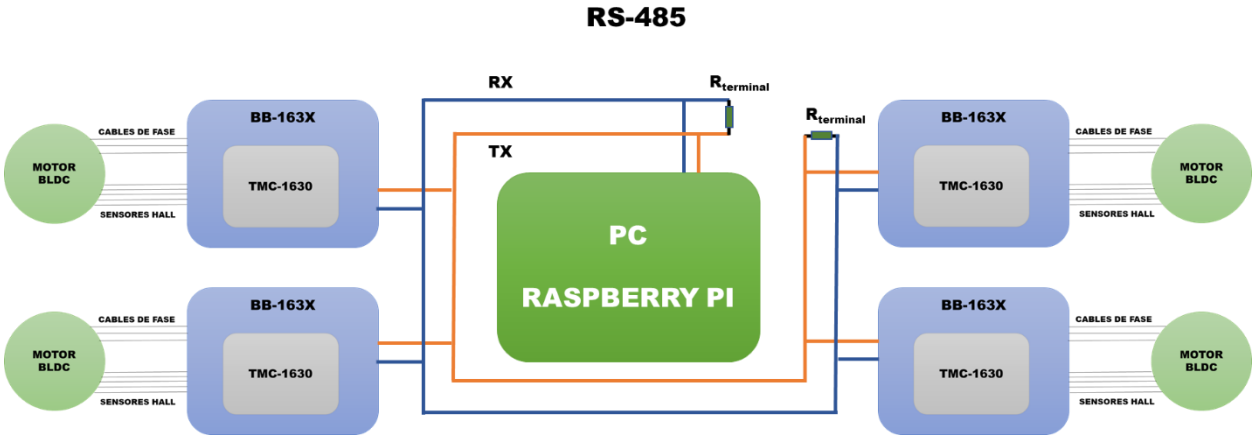


Figura 5.27. Conexión del ordenador central con las tarjetas de control con un sistema RS-485 "Half-Duplex".

Tanto la conexión USB como la RS-485 son de tipo maestro (Raspberry Pi) – esclavo (motores). La principal diferencia en la comunicación entre ambos modos de conexión es que resulta necesario indicar la dirección de la tarjeta en el caso de la conexión RS-485, para enviar el comando, ya que todos los módulos se encuentran en el mismo bus de datos. Sin embargo, dado que el USB, que funciona como una conexión en estrella como nodo central el maestro, a pesar de necesitar la inclusión de la dirección del módulo para emplear un formato correcto del comando, el microcontrolador ignora ese byte de la instrucción. Por este motivo, el programa puede funcionar de manera idéntica, con la salvedad de que en el modo USB se abren cuatro comunicaciones serie desde el software de control, y en el RS-485 sólo se abre una.

5.3. Diseño eléctrico

Como se establece en los requisitos de diseño, el dispositivo debe ser autónomo, por lo que debe carecer de alimentación externa. En otras palabras, debe portar unas baterías que sean capaces de garantizar un suministro energético durante el tiempo de autonomía preestablecido. Por ello, se definirán los niveles energéticos que hacen falta para el funcionamiento del prototipo y, a continuación, se dimensionarán baterías para su selección y compra.

5.3.1. *ADAPTACIÓN DE NIVELES ELÉCTRICOS*

En primer lugar, se debe establecer el máximo nivel de tensión de funcionamiento que se debe emplear en el circuito eléctrico del robot. Dado que los motores son los componentes que mayor tensión de alimentación exigen, serán los que determinan la de la alimentación global del sistema.

Por lo tanto, el nivel superior de tensión será de 24V. Como no existen grandes cantidades de componentes, el siguiente nivel principal será el de la alimentación del ordenador central. En el caso de la Raspberry Pi, los parámetros de la alimentación son los recogidos en la Tabla 5.4.

Medio de alimentación	Micro-USB, Pin GPIO
Tensión nominal	5V DC
Corriente nominal	2,5 A

Tabla 5.4. Datos de la alimentación nominales de la Raspberry Pi 3B+.

La solución más directa sería el empleo de módulos de adaptación de tensión comerciales, como el mostrado en la Figura 5.28, ya que alternativas como divisores resistivos no son tan eficientes.

Figura 5.28. Convertidores DC/DC para la adaptación de tensión de la marca TRACO POWER.

En el caso de esta aplicación, será suficiente con un módulo que adapte de 24V a 5V, sin etapas intermedias, tal y como se indica en la Figura 5.29.

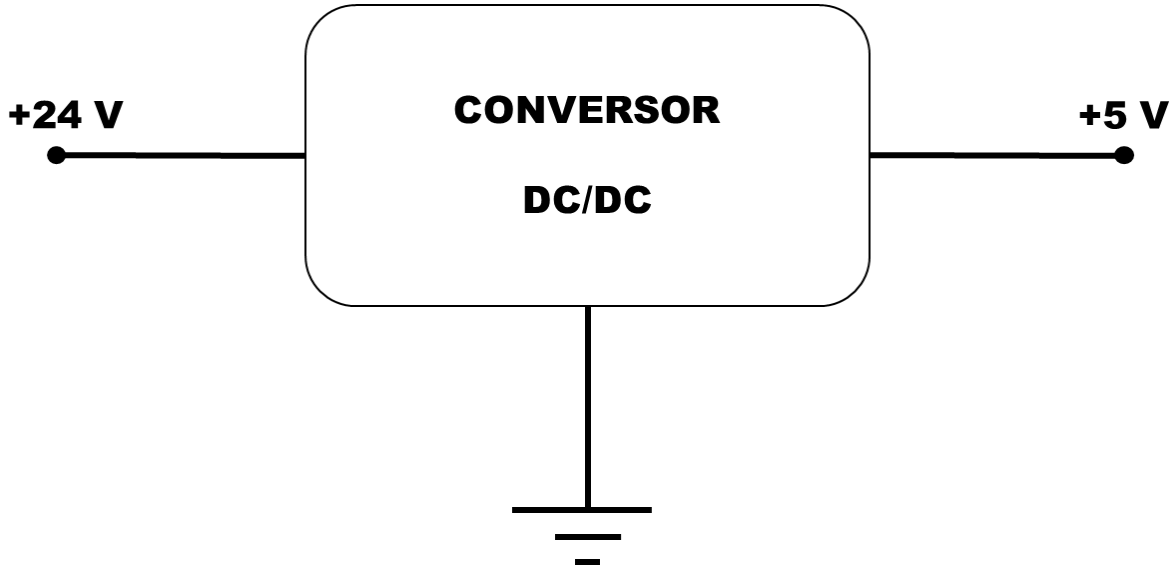


Figura 5.29. Etapa de adaptación eléctrica de tensión.

Concretamente, se selecciona el modelo TEN 60-2411, de la TEN 60 Series, de TRACO POWER [6], que se adecúa a la aplicación que se está diseñando, y garantiza una alta eficiencia en el servicio. Sus principales características se encuentran recogidas en la Tabla 5.5.

Voltaje de entrada	18-36 V DC
Voltaje de salida	5 V DC
Corriente de salida máxima	12 A
Eficiencia	90%

Tabla 5.5. Características de funcionamiento del TEN 60-2411.

5.3.2. *DIMENSIONADO DE LAS BATERÍAS*

Para el dimensionado de las baterías se debe realizar una estimación de los consumos que tendrán los diferentes componentes que aparecen en el diseño. Dado que los consumos se centrarán en los motores y en el ordenador de control del robot, se hará una estimación con un leve sobredimensionado para asegurar otros factores que no se hayan tenido en cuenta.

Es importante destacar que los motores, al ser controlados mediante los drivers anteriormente especificados, es posible realizar una configuración de funcionamiento que se almacene en la memoria de éstos. Concretamente, existe un comando con el que se puede restringir el consumo máximo de corriente.

Esta medida conlleva una consecuencia directa, que se traduce en una limitación del par proporcionado por el motor. Para conocer en qué medida disminuye el par en función de la corriente máxima, se recurre a la hoja de características del motor para conocer la constante de par, que relaciona el par ejercido con la corriente consumida.

Dado que a la salida del motor se encuentra una etapa de reducción de velocidad, por el principio de la conservación de la energía mecánica, el par se multiplica a la misma razón. Por ello, el par resultante útil en esta aplicación será el proporcionado por la salida del reductor, como se muestra en la ecuación (5.22), y será el que sirva de criterio limitante para determinar si la corriente administrada es suficiente o no.

$$T_{Reductor} = i_{reductor} * T_{Motor} = 35{,}65 * T_{Motor} \quad (5.22)$$

La relación del par y la corriente viene determinada por la siguiente ecuación:

$$T_{Motor} = K_{Par} * I_{Motor} \quad (5.23)$$

Si se sustituye la ecuación (5.23) en la (5.22), se tiene la siguiente relación entre el par de salida y la corriente consumida por el motor:

$$T_{Reductor} = 35{,}65 * K_{Par} * I_{Motor} \quad (5.24)$$

Si se despeja la corriente del motor, y se sustituye el par por el mínimo necesario para satisfacer el dimensionamiento realizado en el epígrafe del cálculo de par necesario de los motores, se puede obtener la corriente mínima necesaria:

$$I_{Motor} = \frac{T_{Reductor}}{35{,}65 * K_{Par}} = \frac{1{,}85\ Nm}{35{,}65 * 0{,}059\ \frac{Nm}{A}} = 0{,}88\ A \quad (5.25)$$

Para garantizar un margen de seguridad en los casos de haber picos de corriente durante el servicio, se debe seleccionar un valor levemente mayor. Por ejemplo, se puede limitar la corriente a 1 A, para las cuatro tarjetas de los drivers. Con este valor, se puede hacer un consumo total como el mostrado en la Tabla 5.6, con un consumo total de 6,5 A en una hora de servicio.

		Parámetros de funcionamiento			
Componente	**Cantidad**	**Tensión (V)**	**Corriente (A)**	**Potencia (W)**	**Consumo total (A)**
TMC-1630	**4**	24	1	24	**4**
Raspberry Pi	**1**	5	2,5	12,5	**2,5**
				TOTAL	**6,5**

Tabla 5.6. Cálculo de la capacidad eléctrica necesaria en función de los consumos principales para 1 hora de funcionamiento.

Con este consumo, se puede determinar las características necesarias de las baterías. Para ello, es suficiente con seleccionar una batería con capacidad mayor a la estimada. Por ejemplo, se pueden emplear baterías de 12 V en serie, para conseguir una tensión nominal de 24 V, con una

capacidad moderada, el lugar de recurrir a una de 24 V de la misma capacidad, lo cual supone un aumento significativo de peso en el modelo.

Es por este motivo que, para este caso, se selecciona una batería de la marca Upower, de 12 V, como la mostrada en la Figura 5.30, la cual posee 12 Ah de capacidad.

Figura 5.30. Batería UP12-12, de 12V, de Upower.

Como se expone en la Figura 5.31, el circuito equivalente de las dos baterías en serie se convierte en una batería equivalente de la misma capacidad y el doble de tensión, por lo que se puede determinar el tiempo de autonomía del prototipo en base al consumo total:

$$t_{Autonomía} = \frac{C_{batería\ equivalente}}{I_{consumo\ estimado}} = \frac{12\ Ah}{6{,}5\ A} = 1{,}85\ horas \qquad (5.26)$$

Es decir que, según la ecuación (5.26), se podrá tener una autonomía de 1,85 horas, lo cual queda por encima de los parámetros de diseño, en cuanto a autonomía, de la Tabla 4.1, satisfaciendo así dicho criterio.

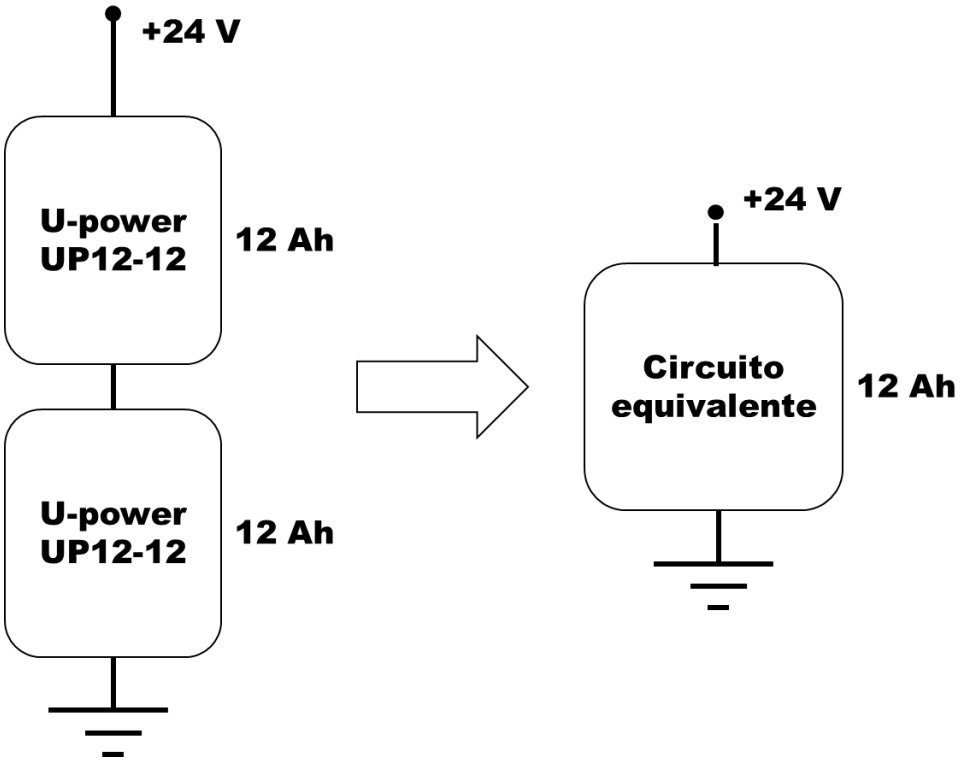


Figura 5.31. Circuito equivalente del montaje en serie de dos baterías de 12V y 12Ah de capacidad.

5.4. Diseño de software

En este apartado se desarrolla todos los aspectos relacionados con el desarrollo del programa de control del ordenador central a bordo. Para ello, se hace uso del entorno gráfico de desarrollo, Qt Creator, en el lenguaje orientado a objetos, C++.

5.4.1. *PROGRAMA PRINCIPAL*

En primer lugar, se debe realizar un esquema principal del programa que se debe realizar, para luego ser concretado en mayor detalle en cada una de sus secciones, y finalmente definido completamente en el código anexo a este documento (véase Anexo II).

Dicho programa principal, cuya descripción gráfica se muestra en la Figura 5.32, a modo de diagrama de flujo, se apoyará en el desarrollo de diferentes clases específicas para la resolución de tareas individuales, como el cálculo cinemático del robot, para obtener la velocidad de giro que se debe ordenar a cada motor, otra para el desarrollo específico de los comandos y su transformación en formato binario.

El programa se ejecutará de la siguiente manera: en primer lugar, se establecerán todas las configuraciones iniciales que sean necesarias para el correcto funcionamiento del programa; a continuación, cuando se hayan terminado los ajustes, se procede a la apertura de la comunicación serie con los motores, que resulta un problema específico del sistema operativo empleado – en este caso, Linux – , por lo que sólo se desarrollará una solución para este caso; posteriormente, si se ha logrado abrir la comunicación, se recibirá una orden de velocidad; con el valor de la velocidad, se realizará el cálculo de la cinemática descrita para el conjunto de ruedas Mecanum, para determinar la velocidad de giro de los motores; después, una subrutina deberá configurar el arreglo de 9 bytes para indicar la velocidad de giro; finalmente, se escribe a través del puerto serie y se espera a que se vuelva a ejecutar el lazo de control.

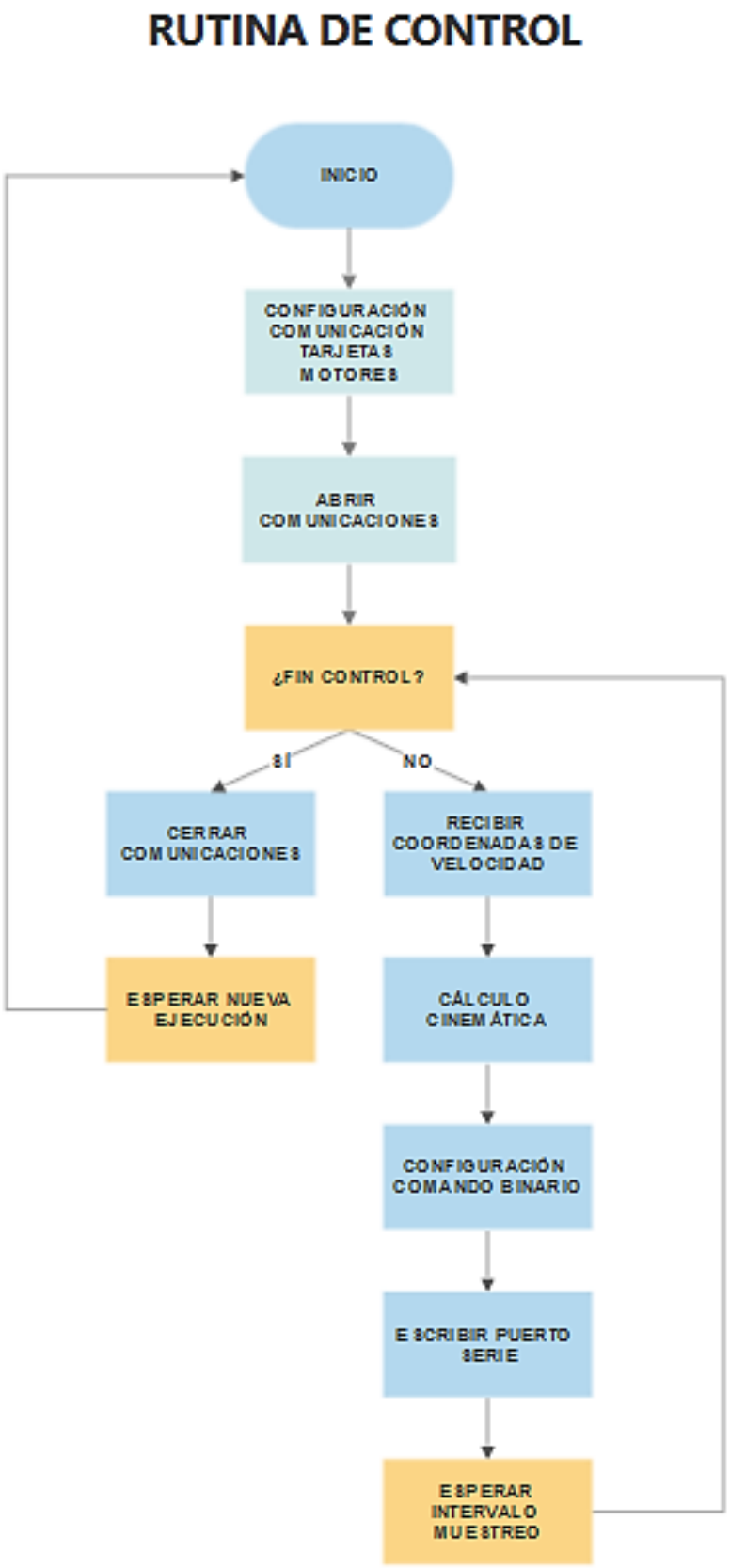


Figura 5.32. Diagrama de flujo del programa principal de control de la plataforma omnidireccional.

5.4.2. *CONFIGURACIÓN INICIAL*

Para realizar la configuración inicial (véase Figura 5.33), solo se precisa de establecer algunos de los parámetros principales para la comunicación, tales como el nombre del puerto con el que se quiere establecer comunicación, los parámetros de la comunicación como la velocidad de transmisión, el tamaño de la unidad de transmisión, bits de stop o paridad, etc.

En Linux, el manejo de los puertos serie se realiza como si se trabajase sobre un archivo más, por lo que a la hora de implementar el código se debe especificar la ruta del dispositivo en el sistema operativo.

Como detalle, salvo que se conecten los motores de manera secuencial, se debe comprobar y corregir la posición de cada uno, por lo que se introduce una subrutina que se encarga de enviar una instrucción preguntando a la tarjeta por su dirección, para comprobar su posición en el sistema global.

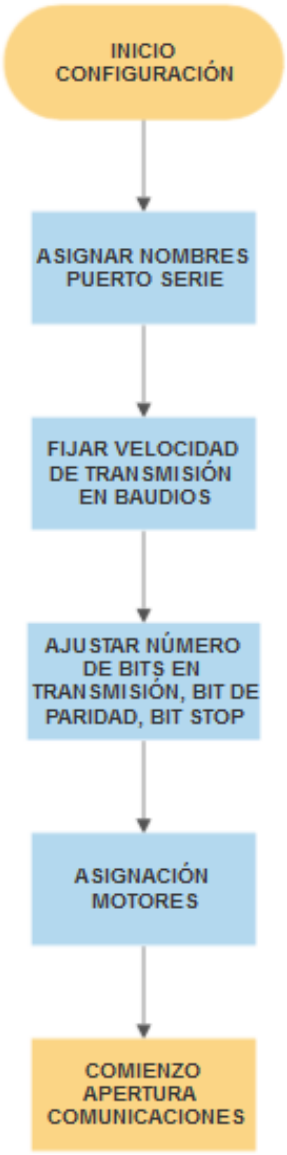


Figura 5.33. Diagrama de flujo de la configuración inicial.

5.4.3. *CÁLCULO DE LA CINEMÁTICA*

Como ya se había definido en el epígrafe 5.1.2, el modelo cinemático inverso del prototipo queda descrito por las ecuaciones obtenidas al final de su desarrollo. La clase desarrollada en este programa funciona de la siguiente manera: se recogen las coordenadas de velocidad (velocidad lineal y su ángulo, y la velocidad angular), y se realiza el cálculo matricial a partir de la matriz de geometría.

El resultado de este cálculo será un vector de 4 coordenadas, que se corresponderán con la velocidad de giro de cada rueda. Para determinar la velocidad de giro necesaria en el motor, es suficiente con multiplicar esas velocidades por la reducción de velocidad del motorreductor.

5.4.4. *CONFIGURACIÓN DEL COMANDO*

Para la preparación del comando a enviar a las 4 tarjetas que se encargan del control de los motores, se desarrolla una clase específica en el programa principal. Esta clase tiene como objetivo crear un vector de bytes en el que se codifiquen cada uno de los elementos que conforman el comando, como se muestra en la Tabla 5.7.

Dirección	**Comando**	**Tipo**	**Memoria**	**Valor**				**Checksum**
1	1	1	1	1	1	1	1	1

Tabla 5.7. Descomposición en bytes del comando para la tarjeta de TRINAMIC.

Para dicho subprograma (véase Figura 5.34), será necesario descomponer el argumento del valor del comando en 4 bytes, lo cual se puede hacer fácilmente mediante operaciones binarias, como se muestra en la Tabla 5.8, donde el operador “>>” significa desplazamiento de bits a la derecha N posiciones, y el operador “&” significa la operación lógica AND realizada bit a bit.

1º Byte	Valor >> 24
2º Byte	Valor >> 16
3º Byte	Valor >> 8
4º Byte	Valor & 0xFF

Tabla 5.8. Descomposición del argumento del comando en 4 bytes.

El byte del checksum será un byte de comprobación, que se determinará haciendo la suma binaria de todos los demás bytes del comando, incluida la dirección.

Para cada comando, el módulo de control enviará una respuesta devolviendo parámetros de interés como el valor del comando, en caso de ser una solicitud de un valor numérico de funcionamiento, o, por ejemplo, también, el estado tras la recepción del comando. Esto es, en otras palabras, si el comando se ha recibido en un formato correcto, si se ha cargado en la

memoria EEPROM del microcontrolador, si hay algún argumento del comando erróneo, si el checksum de comprobación está mal, si el comando no está disponible, etc.

La respuesta recibe una estructura similar a la del comando que se envía, con unas leves diferencias, recogidas en la Tabla 5.9, donde se descompone la información de la respuesta en 9 bytes.

Dirección	Modulo	Estado	Comando	Valor				Checksum
1	1	1	1	1	1	1	1	1

Tabla 5.9. Descomposición en bytes de la respuesta al comando.

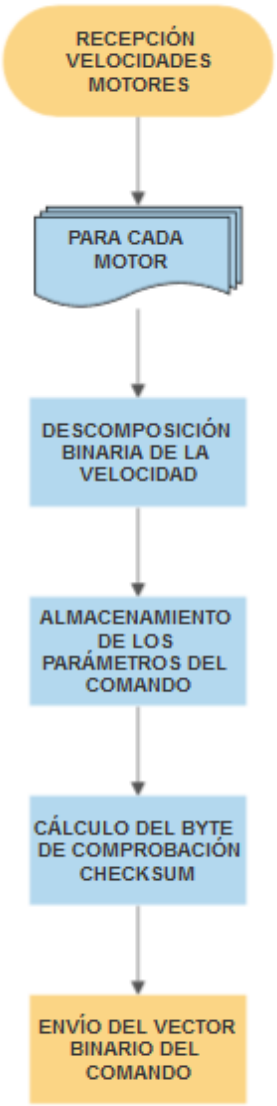


Figura 5.34. Diagrama de flujo de la preparación del comando.

Por lo general, la lectura de la respuesta no siempre sería estrictamente necesaria, aunque sí recomendable, para asegurar el correcto funcionamiento de la escritura de comandos.

Sin embargo, para la fase inicial de configuración del programa, en la que se debe comprobar qué motor se corresponde con el puerto serie configurado, sí es necesario realizar la lectura de la respuesta, para poder recibir el valor de la identidad de la tarjeta, para poder así corregir las nomenclaturas.

Con esto, se cierra el diseño conceptual del programa de control de la plataforma omnidireccional. Sin embargo, la implementación final y la resolución de errores propios del código generado no se reflejan en este epígrafe.

En el código anexo, se podrán comprobar cómo se solventaron cada uno de los problemas específicos de programación en el lenguaje C++.

5.5. Diseño final

Como conclusión de este diseño, se muestra en la Figura 5.35 el modelo final del prototipo diseñado en 3D, en el que se encuentran los elementos seleccionados en fases posteriores al diseño mecánico. En el Anexo X, donde se encuentran todos los planos del prototipo, se detalla cómo se han incluido estas últimas incorporaciones al diseño.

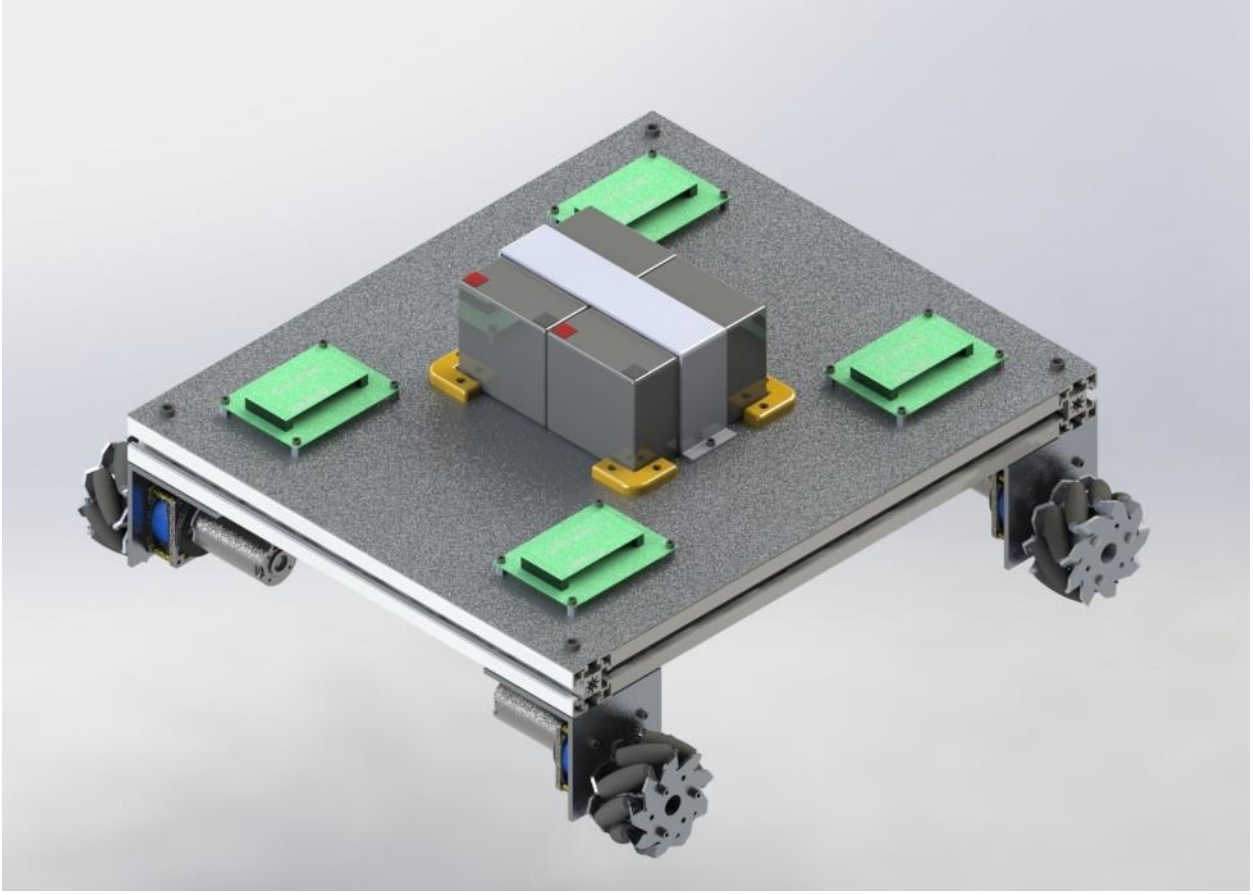


Figura 5.35. Imagen renderizada del modelo final de la plataforma omnidireccional diseñada.

6. PRESUPUESTO DEL PROTOTIPO

MÁSTER EN INGENIERÍA MECATRÓNICA						
Nº ORDEN	**CONCEPTOS**	**Nº UNIDADES**	**FABRICANTE**	**TIPO**	**PRECIO UNITARIO**	**TOTAL**
1	**Motores BLDC**	4	ELMEQ	Mecánica	182,00	728,00 €
2	**Driver de motor BLDC TMC1630**	4	TRINAMIC	Electrónica	173,00	692,00 €
3	**Kit evaluación BB-163X**	4	TRINAMIC	Electrónica	93,86	375,44 €
4	**Módulo de adaptación de tensión**	1	TRACO POWER	Electrónica	94,23	94,23 €
5	**Raspberry Pi 3B+**	1	RPi Foundation	Electrónica	40,29	40,29 €
6	**Barras de aluminio normalizado**	4	RS Components	Mecánica	8,40	33,60 €
7	**Chapa metálica**	1	Universidad de Oviedo	Mecánica	15,00	15,00 €
8	**Acoplamiento**	4	Universidad de Oviedo	Mecánica	10,50	42,00 €
9	**Escuadras de aluminio**	4	Universidad de Oviedo	Mecánica	6,25	25,00 €
10	**Batería 12V 12Ah**	2	U-power	Electrónica	16,94	33,88 €
11	**Soporte baterías 3D PLA**	5	Universidad de Oviedo	Mecánica	2	10,00 €
12	**Horas de ingeniería**	200	Mano de obra		10	2.000 €
					SUBTOTAL	**4.089,44 €**
			Gastos generales		**13 %**	**531,63 €**
			TOTAL			**4.621,07 €**

7. PROPUESTAS DE MEJORA

En este capítulo se contemplan posibles mejoras en el diseño o en las pautas del desarrollo de éste, para proyectos de misma índole. Las propuestas se centrarán en aspectos relacionados con los tratados a lo largo de este proyecto.

7.1. Mejoras de aspectos mecánicos

- En primer lugar, en el caso de que este proyecto se quisiera ampliar a un marco de carácter más industrial, para poder ser producido y comercializado, deberían mejorarse aspectos como el aislamiento y cierre del prototipo, de cara a un mejor aspecto estético, además de poder proporcionar mayor seguridad.
- Se podrían emplear piezas mecanizadas con mejores tolerancias dimensionales para garantizar un mejor diseño y montaje, además de controlar dimensiones con influencia crítica en el funcionamiento, como la separación entre las ruedas.
- Por otro lado, se debería hacer un estudio comparativo del comportamiento de diferentes sistemas omnidireccionales en el área de trabajo, en función de la adherencia, ya que el modelo actualmente tiene únicamente 4 contactos puntuales con el suelo, mientras que otras ruedas tendrán una mayor superficie de apoyo.

7.2. Mejoras de aspectos electrónicos

- Se podrían añadir sensores para realizar un seguimiento del movimiento de la plataforma, para controlar la veracidad del movimiento estimado, además de poder llegarse a realizar un lazo de control cerrado para optimizar la trayectoria. Concretamente, esta es una de las misiones del proyecto de investigación y se ha ido realizando de manera paralela en el proyecto. En futuras etapas del proyecto se irá realizando esta integración en el prototipo definitivo.

7.3. Mejoras de aspectos eléctricos

- Como propuesta, se podría optimizar la selección de componentes a emplear para minimizar el número de etapas de adaptación de nivel de tensión, consiguiendo así una mayor eficiencia en el suministro energético, además de aumentar la autonomía.
- Se deberían añadir elementos de seguridad como fusibles frente a la alimentación del prototipo, para evitar posibles daños en el equipo a bordo.

7.4. Mejoras de aspectos de software

- Como en muchas aplicaciones de robótica a nivel de prototipos de investigación [7], el software ROS (Robot Operating System) es ampliamente utilizado. Por ello, además de la enorme cantidad de software ya desarrollado para robótica en este entorno, se convierte en una solución muy deseable a ser implementada para el control del prototipo. Además, dispone de múltiples herramientas para la comunicación con otros subsistemas que puedan ser implementados sobre el prototipo final.
- Dado que los algoritmos de navegación que se han ido desarrollando a lo largo de este proyecto han sido implementados en ROS, un objetivo específico será la implementación de la recepción de los comandos de velocidad a través de este

entorno, ya que los algoritmos de navegación son los que determinan las velocidades a enviar a la plataforma.

- Se podrían desarrollar aplicaciones GUI, o de interfaz gráfica, para la monitorización y control del robot en tiempo real, pudiendo ser una herramienta muy útil en su uso final.

8. CONCLUSIONES

Tras la realización de este Trabajo Fin de Máster, como última etapa de los estudios del Máster Universitario en Ingeniería Mecatrónica, se puede llegar a una serie de conclusiones en base al trabajo realizado, a saber:

- En primer lugar, cabe destacar que, como se había mencionado al comienzo de este proyecto, se han abarcado múltiples disciplinas del ámbito de la ingeniería para llegar a lograr un único objetivo común: la realización de un prototipo mecatrónico funcional.
- Se ha llevado a cabo un estudio completo del estado del arte de las diferentes tecnologías empleadas a niveles de investigación, comercial y de patentes, para la obtención de sistemas de cinemática omnidireccional.
- Se ha llevado a cabo el cálculo del modelo cinemático del conjunto de ruedas mecanum que dotan de movilidad al robot.
- Se ha realizado un diseño mecánico del prototipo, que lo dota de integridad estructural suficiente, además de la capacidad de portar todos los elementos necesarios para su funcionamiento.
- Además, se han realizado todos los cálculos necesarios para dimensionar los motores empleados en el diseño, de tal manera que presenten un servicio fiable en el escenario planteado.
- Se han llevado a cabo los cálculos necesarios para dimensionar el suministro energético del dispositivo, de manera que cumpla con los criterios de autonomía mínima.
- Se han seleccionado aquellos componentes comerciales necesarios para el funcionamiento electrónico del control de los motores, además de componentes empleados para la adaptación de niveles eléctricos.
- Se ha desarrollado un programa de control basado en Linux para el control de múltiples dispositivos mediante comunicación serie, en el que se han contemplado todos los aspectos del diseño global.
- Se han logrado integrar satisfactoriamente todos los elementos resultantes tras el diseño de cada una de las ramas técnicas del proyecto, dando lugar al prototipo de plataforma omnidireccional obtenida.
- Se ha obtenido un prototipo sobre el que se podrían implementar todos aquellos elementos y subsistemas que puedan ser necesarios para el fin último del robot, consistente en el desempeño de tareas de inspección y subsanación de errores en chapa gruesa.

En resumen, se trata de un proceso largo y complejo que involucra múltiples y diversos conocimientos de numerosas disciplinas técnicas, por lo que se requiere de una buena coordinación para poder integrar todos los trabajos realizados. En realidad, si se tratase del caso de un prototipo a nivel industrial, se requeriría seguir un proceso iterativo de refinado de los diseños para obtener un modelo optimizado a nivel de costes, fabricación, apariencia, etc., por lo que se trataría de un proceso aún más largo.

Sin duda, se trata de un proyecto de ingeniería muy acorde con el perfil de la Ingeniería Mecatrónica, en el que se han podido poner en práctica todos aquellos conocimientos adquiridos a lo largo del programa de Máster, además de adquirir otros tantos nuevos que han servido para complementar a los previos.

9. REFERENCIAS

[1] F. C. Park y K. M. Lynch, Modern Robotics: Mechanics, Planning, and Control, Cambridge: Cambridge University Press, 2017.

[2] E. Maulana, M. M. Aziz y H. Veri, «Inverse Kinematic Implementation of Four-Wheels Mecanum Drive Mobile Robot Using Stepper Motors,» *International Seminar on Intelligent Technology and Its Applications,* pp. 51-55, 2015.

[3] ELMEQ, «ELMEQ: MOTORES ELÉCTRICOS, MOTORREDUCTORES Y REDUCTORES DE VELOCIDAD,» [En línea]. Available: https://www.elmeq.es/. [Último acceso: Mayo 2019].

[4] Digi-key, «Digi-Key Electronics,» [En línea]. Available: https://www.digikey.es/. [Último acceso: Mayo 2019].

[5] TRINAMIC, «TRINAMIC,» [En línea]. Available: https://www.trinamic.com/. [Último acceso: Mayo 2019].

[6] TRACO POWER, «TRACO POWER,» [En línea]. Available: https://www.tracopower.com. [Último acceso: Junio 2019].

[7] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler y A. Ng, «ROS: an open-source Robot Operating System,» *Willow Garage,* 2009.

10. ANEXOS

LISTADO DE ANEXOS

ANEXO I: Código de MATLAB para el cálculo de la cinemática

ANEXO II: Código del control de la plataforma omnidireccional

ANEXO III: HOJA DE CARACTERÍSTICAS DE LOS MOTORES

ANEXO IV: HOJA DE CARACTERÍSTICAS DE LAS RUEDAS MECANUM

ANEXO V: HOJA DE CARACTERÍSTICAS DE LAS BATERÍAS

ANEXO VI: HOJA DE CARACTERÍSTICAS DE LA RASPBERRY PI 3B+

ANEXO VII: HOJA DE CARACTERÍSTICAS DEL REGULADOR DE TENSIÓN

ANEXO VIII: HOJA DE CARACTERÍSTICAS DEL KIT DE EVALUACIÓN BB-163X

ANEXO IX: HOJA DE CARACTERÍSTICAS DEL DRIVER TMC-1630

ANEXO X: PLANOS

ANEXO I: Código de MATLAB para el cálculo de la cinemática

```
clc, clear all
%% CÁLCULO CINEMÁTICO
%Constantes
a = 0.5;
b = 0.5;
R = 0.1/2;

%Matriz de la geometría
M = [1 -1 -(a+b);1 1 (a+b);1 1 -(a+b);1 -1 (a+b)] ./ R;

vtheta = [];
w_rad_s = [];
w_rev_m = [];

%Se hace el cálculo con valores de theta de 0 a 360°
for theta = 0:0.01:2*pi
    %Variables de entrada
    vx = cos(theta);
    vy = sin(theta);
    w  = 0;

    %Cálculo de la cinemática inversa
    m = [vx vy w];
    res = M*m';                  %Resultados en rad/s
    res2 = 60*(M*m')/(pi*2);     %Resultados en rpm

    %Se almacenan los resultados
    vtheta = [vtheta theta];
    w_rad_s = [w_rad_s res];
    w_rev_m = [w_rev_m res2];
end


%% REPRESENTACIÓN GRÁFICA
figure(1)
subplot(2,2,1)
plot(vtheta,w_rev_m(1,:))
xlabel('theta')
ylabel('RPM')
title('Rueda 1')
subplot(2,2,2)
plot(vtheta,w_rev_m(2,:))
xlabel('theta')
ylabel('RPM')
title('Rueda 2')
subplot(2,2,3)
plot(vtheta,w_rev_m(3,:))
xlabel('theta')
```

```
ylabel('RPM')
title('Rueda 3')
subplot(2,2,4)
plot(vtheta,w_rev_m(4,:))
xlabel('theta')
ylabel('RPM')
title('Rueda 4')

%% APROXIMACIÓN DE LA CINEMÁTICA DIRECTA

%Velocidades de los motores a 100 rpm
w1 = 100;
w2 = 100;
w3 = 100;
w4 = 100;

%Vector de velocidades de las ruedas
vw = [w1 w2 w3 w4]';
%Conversión a radianes/segundo
vw = 2 * pi .* vw / 60;

%Aproximación de la cinemática directa
vvel = M\vw;

%Modulo del vector velocidad del robot
vel_global = sqrt(vvel(1)^2 + vvel(2)^2);
```

ANEXO II: Código del control de la plataforma omnidireccional

I. MAIN.CPP

```
#include "PuertoSerieMotor.h"

int main()
{
   PuertoSerieMotor s;
   return 0;
}
```

II. INITPUERTOSERIE.H

```
#ifndef INITPUERTOSERIE_H
#define INITPUERTOSERIE_H


int InicializarPuertoSerie(const char *pcPuerto,int iVelocidad, char cParidad, int iDatos,int iStop);

#endif // INITPUERTOSERIE_H
```

III. INITPUERTOSERIE.CPP

```
#include <stdio.h>
#include <stdlib.h>
#include <termios.h>
#include <errno.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/ioctl.h>

int InicializarPuertoSerie(const char *pcPuerto,int iVelocidad, char cParidad, int iDatos,int iStop)
{
    struct termios oldtio, newtio;
    int iHandlePuerto;

    //Primer paso, abrir el puerto
    iHandlePuerto = open(pcPuerto, O_RDWR | O_NOCTTY);

    if (iHandlePuerto < 0)
       return iHandlePuerto;

    fcntl(iHandlePuerto, F_SETOWN, getpid());
    tcgetattr(iHandlePuerto, &oldtio);

    //    Configurar velocidad
    switch (iVelocidad)
    {
       case 9600:
          newtio.c_cflag = B9600;          break;
       case 19200:
          newtio.c_cflag = B19200;       break;
       case 38400:
```

```
            newtio.c_cflag = B38400;        break;
        case 57600:
            newtio.c_cflag = B57600;        break;
        case 115200:
            newtio.c_cflag = B115200;       break;
        default:
            close(iHandlePuerto);
            return -1;
    }

    //    Configurar bits de datos
    switch (iDatos)
    {
        case 5:
            newtio.c_cflag |= CS5 | CLOCAL | CREAD; break;
        case 6:
            newtio.c_cflag |= CS6 | CLOCAL | CREAD; break;
        case 7:
            newtio.c_cflag |= CS7 | CLOCAL | CREAD; break;
        case 8:
            newtio.c_cflag |= CS8 | CLOCAL | CREAD; break;
        default:
            close(iHandlePuerto);
            return -2;
    }

    //    Configurar Paridad
    switch (cParidad)
    {
        case 'N':
            newtio.c_iflag = IGNPAR ;           break;
        case 'P':
            newtio.c_cflag |= PARENB;           break;
        case 'I':
            newtio.c_cflag |= PARENB | PARODD;    break;
        default:
            close(iHandlePuerto);
            return -3;
    }

    //    Configurar Bits de stop
    switch (iStop)
    {
        case 1:
            break;
        case 2:
            newtio.c_cflag |= CSTOPB;           break;
        default:
            close(iHandlePuerto);
            return -4;
    }

    newtio.c_oflag = 0;
    newtio.c_lflag = 0;
    newtio.c_cc[VMIN] = 9;
```

```
    newtio.c_cc[VTIME] = 5;

    cfmakeraw(&newtio);

    tcflush(iHandlePuerto, TCIFLUSH);
    tcsetattr(iHandlePuerto, TCSANOW, &newtio);

    return iHandlePuerto;
}
```

IV. COMANDOTMCL.H

```
#ifndef COMANDOTMCL_H
#define COMANDOTMCL_H

//Opcodes of all TMCL commands that can be used in direct mode
#define TMCL_ROR 1
#define TMCL_ROL 2
#define TMCL_MST 3
#define TMCL_MVP 4
#define TMCL_SAP 5
#define TMCL_GAP 6
#define TMCL_STAP 7
#define TMCL_RSAP 8
#define TMCL_SGP 9
#define TMCL_GGP 10
#define TMCL_STGP 11
#define TMCL_RSGP 12
#define TMCL_RFS 13
#define TMCL_SIO 14
#define TMCL_GIO 15
#define TMCL_SCO 30
#define TMCL_GCO 31
#define TMCL_CCO 32

//Opcodes of TMCL control functions (to be used to run or abort a TMCL program in the module)
#define TMCL_APPL_STOP 128
#define TMCL_APPL_RUN 129
#define TMCL_APPL_RESET 131

//Options for MVP commandds
#define MVP_ABS 0
#define MVP_REL 1
#define MVP_COORD 2

//Options for RFS command
#define RFS_START 0
#define RFS_STOP 1
#define RFS_STATUS 2

#include <stdlib.h>
#include <stdio.h>

class comandoTMCL
```

```
{
public:
    explicit comandoTMCL();

private:
    unsigned char address = 0x01, command = 0, type = 0, motor_bank = 0;
    unsigned char value[4];
    unsigned char checksum = 0;
public:
    unsigned char status;
    int respuesta = 1;

public:
    void ConfigComando(unsigned char address, unsigned char command, unsigned char type,
unsigned char motor_bank, long int value);
    unsigned char TXbuffer[9], RXbuffer[9];


    unsigned char getAddress() const;
    void setAddress(const unsigned char &value);

    unsigned char getCommand() const;
    void setCommand(const unsigned char &value);

    unsigned char getType() const;
    void setType(const unsigned char &value);

    unsigned char getMotor_bank() const;
    void setMotor_bank(const unsigned char &value);

    void setValue(const long int &val);

    unsigned char getChecksum() const;
    void setChecksum(const unsigned char &value);

private:
    void TraducirComandoEnviado();
public:
    void TraducirRespuesta();

private:
    void calc_checksum();

};

#endif // COMANDOTMCL_H
```

V. COMANDOTMCL.CPP

```
#include "comandoTMCL.h"

comandoTMCL::comandoTMCL()
{
}

unsigned char comandoTMCL::getChecksum() const
{
    return checksum;
}

void comandoTMCL::setChecksum(const unsigned char &value)
{
    checksum = value;
}

unsigned char comandoTMCL::getMotor_bank() const
{
    return motor_bank;
}

void comandoTMCL::setMotor_bank(const unsigned char &value)
{
    motor_bank = value;
}

void comandoTMCL::setValue(const long int &val)
{
    value[0] = static_cast<unsigned char>(val >> 24);
    value[1] = static_cast<unsigned char>(val >> 16);
    value[2] = static_cast<unsigned char>(val >>  8);
    value[3] = static_cast<unsigned char>(val & 0xff);
}

unsigned char comandoTMCL::getType() const
{
    return type;
}

void comandoTMCL::setType(const unsigned char &value)
{
    type = value;
}

unsigned char comandoTMCL::getCommand() const
{
    return command;
}

void comandoTMCL::setCommand(const unsigned char &value)
{
    command = value;
}
```

```
unsigned char comandoTMCL::getAddress() const
{
    return address;
}

void comandoTMCL::setAddress(const unsigned char &value)
{
    address = value;
}

void comandoTMCL::ConfigComando(unsigned char address_n, unsigned char command_n,
unsigned char type_n, unsigned char motor_bank_n, long int value_n)
{
    //Se asignan los valores a todos los parámetros del comando:
    setAddress(address_n);
    setCommand(command_n);
    setType(type_n);
    setMotor_bank(motor_bank_n);
    setValue(value_n);

    //Y se genera automáticamente la traducción a caracteres para la comunicación serie:
    TraducirComandoEnviado();

}

void comandoTMCL::TraducirComandoEnviado()
{
    //Se asignan los parametros del comando
    TXbuffer[0] = (address);
    TXbuffer[1] = (command);
    TXbuffer[2] = (type);
    TXbuffer[3] = (motor_bank);
    TXbuffer[4] = (value[0]);
    TXbuffer[5] = (value[1]);
    TXbuffer[6] = (value[2]);
    TXbuffer[7] = (value[3]);
    TXbuffer[8] = 0;                    //Se reinicia el checksum

    //Finalmente, se calcula el byte checksum
    calc_checksum();

    TXbuffer[8] = (checksum);
}

void comandoTMCL::TraducirRespuesta()
{
    status = RXbuffer[2];
    respuesta = (RXbuffer[4] << 24) | (RXbuffer[5] << 16) | (RXbuffer[6] <<  8) | (RXbuffer[7]);
}


void comandoTMCL::calc_checksum()
{
```

```
    //Para comunicación serie USB es necesario enviar un ultimo byte como comprobante del
comando:
    int i;
    for(i = 0, checksum = 0; i < 8; i++)
            checksum += TXbuffer[i];

}
```

VI. ROBOTKINEMATICS.H

```
#ifndef ROBOTKINEMATICS_H
#define ROBOTKINEMATICS_H

#include <math.h>
#include <list>
#include <iostream>
#include <unistd.h>

//Definición de las constantes físicas del robot:
#define as 0.5f            //metros
#define bs 1.0f            //metros
#define Rs 0.05f           //metros
#define i_red_motor 35.6    //adimensional (rpm/rpm)
#define pi 3.1416f         //cte

using namespace std;

class robotKinematics
{
public:
    explicit robotKinematics();
    explicit robotKinematics(float vel, float ang, float vel_ang);

    //Funciones parámetros de entrada:
    void setVelocity(float value);
    void setTheta(float value);
    void setW(float value);

private:
    float velocity;
    float theta;
    float w;

    float ikMatrix[4][3] = {{1/Rs,-1/Rs,-(as+bs)/Rs}, {1/Rs,1/Rs,(as+bs)/Rs}, {1/Rs,1/Rs,-
(as+bs)/Rs}, {1/Rs,-1/Rs,(as+bs)/Rs}};
    float vector[3];

public:
    float w_wheels[4];

public:
    void inverseKinematics(float vel, float ang, float vel_ang);

};
```

```
#endif // ROBOTKINEMATICS_H
```

VII. ROBOTKINEMATICS.CPP

```
#include "robotKinematics.h"

robotKinematics::robotKinematics()
{
    setVelocity(0);
    setTheta(0);
    setW(0);
}

robotKinematics::robotKinematics(float vel, float ang, float vel_ang)
{
    setVelocity(vel);
    setTheta(ang);
    setW(vel_ang);
}


void robotKinematics::setVelocity(float value)
{
    velocity = value;
}

void robotKinematics::setTheta(float value)
{
    theta = value;
}

void robotKinematics::setW(float value)
{
    w = value;
}

void robotKinematics::inverseKinematics(float vel, float ang, float vel_ang)
{
    int i, j;
    float vx, vy, sum;

    //Se configuran las variables de entrada
    setVelocity(vel);
    setTheta(ang);
    setW(vel_ang);

    //Se descompone la velocidad en componentes cartesianas
    vx = velocity * cos(theta*pi/180);
    vy = velocity * sin(theta*pi/180);
```

```
    //Se define el vector para el cálculo de la cinemática inversa
    vector[0] = vx;
    vector[1] = vy;
    vector[2] =  w;

    std::cout << "vx: "<< vector[0]  << "\tvy: " << vector[1] << "\tw:" << vector[2] << "\n";

    //Se realiza el producto de la matriz por el vector, calculando así las velocidades angulares
de cada rueda
    for (i = 0, sum = 0; i < 4; i++) {
        w_wheels[i] = 0;

        for (j = 0, sum = 0; j < 3; j++){
            sum += (ikMatrix[i][j] * vector[j]) * static_cast<float>(i_red_motor);
        }
        w_wheels[i] = sum;
    }

    std::cout << "Vel de cada motor:";
    for (i = 0;i < 4;i++) {
        std::cout << w_wheels[i] << "\t";
    }
    std::cout << "\n";
}
```

VIII. PUERTOSERIEMOTOR.H

```
#ifndef PUERTOSERIEMOTOR_H
#define PUERTOSERIEMOTOR_H

#include <list>
#include <string>
#include <iostream>
#include <unistd.h>
#include "initpuertoserie.h"
#include "comandoTMCL.h"
#include "robotKinematics.h"

#define n_motores 4

using namespace std;
class PuertoSerieMotor
{
public:
    explicit PuertoSerieMotor();
    ~PuertoSerieMotor();

private:
    int serialMotor1, serialMotor2, serialMotor3, serialMotor4;
    comandoTMCL motor1, motor2, motor3, motor4;
    int motores_activos;
    int IDmotores[4] = {1,1,1,1};
    float vel_motores[4];
    bool configIDmotor = false;
```

```
    void ConfigSerialMotor();
    void AsignarMotores();
    void CalcVelMotores(float vel, float theta, float w);
    void ConfigComandosVelMotores();
    void RWComandos();

private:
    void OnSerialDataAvailable();
};

#endif // PUERTOSERIEMOTOR_H
```

IX. PUERTOSERIEMOTOR.CPP

```
#include "PuertoSerieMotor.h"

PuertoSerieMotor::PuertoSerieMotor()
{
    /* Sobre el robot visto por encima:

                        IZQUIERDA
                  M3 ---------------------- M1
                     |                      |
                     |          y           |
                     |          |           |
           BACK      |          |___   x    |   FRONT
                     |                      |
                     |                      |
                     |                      |
                  M4 ---------------------- M2
                        DERECHA                        */

    //Registramos el número de motores conectados:
    motores_activos = 0;
    ConfigSerialMotor();

    //Una vez recogidos todos los puertos serie disponibles, en caso de que hubiese:
    if(motores_activos){
        std::cout << "Configuracion del puerto serie establecida\n";
        std::cout << "Puertos serie disponibles: ("<<motores_activos<<")\n\n";

        //Antes de proceder al programa principal, hay que asignar
        // el motor correctamente a su posición física
        AsignarMotores();

        //Se procede al tratamiento del puerto serie:
        OnSerialDataAvailable();
    }
    else
        std::cout << "Error: No hay puertos disponibles\n";
}
```

```
PuertoSerieMotor::~PuertoSerieMotor()
{
    //Se cierran las comunicaciones serie:
    close(serialMotor1);
    close(serialMotor2);
    close(serialMotor3);
    close(serialMotor4);
    std::cout << "Cierre comunicaciones" << endl;
}

void PuertoSerieMotor::ConfigSerialMotor()
{
    serialMotor1 = InicializarPuertoSerie("/dev/ttyACM0",9600,'N',8,1); //ttyUSB0

    if ( !(serialMotor1 < 0) ){
        motores_activos++;
        serialMotor2 = InicializarPuertoSerie("/dev/ttyACM1",9600,'N',8,1);

        if ( !(serialMotor2 < 0) ){
            motores_activos++;
            serialMotor3 = InicializarPuertoSerie("/dev/ttyACM2",9600,'N',8,1);

            if ( !(serialMotor3 < 0) ) {
                motores_activos ++;
                serialMotor4 = InicializarPuertoSerie("/dev/ttyACM3",9600,'N',8,1);

                if ( !(serialMotor4 < 0) )
                    motores_activos++;
            }
        }
    }
}

void PuertoSerieMotor::AsignarMotores()
{
    /* Se configura el comando para conocer el address interno de cada motor,
       que se corresponde con sus respectiva posición del modelo cinemático*/
    motor1.ConfigComando(0x01,TMCL_GGP,66,0,0);
    motor2.ConfigComando(0x01,TMCL_GGP,66,0,0);
    motor3.ConfigComando(0x01,TMCL_GGP,66,0,0);
    motor4.ConfigComando(0x01,TMCL_GGP,66,0,0);

    //Se ejecutan los comandos y se almacenan las respuestas
    RWComandos();
    //Se desactiva el flag para no leer el puerto serie en la ejecución del programa
    configIDmotor = false;

    /* Con la ID de cada tarjeta registrada en cada motor, se procede a la corrección de IDs
       ya que, al conectar los motores, adquieren nombres independientes a su orden en el
       modelo cinemático  */
    IDmotores[0] = motor1.respuesta - 1;
    IDmotores[1] = motor2.respuesta - 1;
    IDmotores[2] = motor3.respuesta - 1;
    IDmotores[3] = motor4.respuesta - 1;
}
```

```
void PuertoSerieMotor::CalcVelMotores(float vel, float theta, float w)
{
    int i;
    robotKinematics cinematica;

    //Calculo de la cinematica inversa:
    cinematica.inverseKinematics(vel,theta,w);

    //Asignación de los valores de velocidad al array de esta clase
    for (i = 0; i < 4; i++)
    {
        //Corrección por ID de cada motor
        vel_motores[IDmotores[i]] =  cinematica.w_wheels[i];
        /* Según el lado del motor, se invierte el sentido de giro,
         * ya que todos tienen el sentido antihorario como positivo
         * y la cinemática no tiene en cuenta dicho problema */
        if (!((IDmotores[i]+1) % 2))
            vel_motores[IDmotores[i]] = (-1) * vel_motores[IDmotores[i]]; //Motor par
    }
}

void PuertoSerieMotor::ConfigComandosVelMotores()
{
    int i;

    for (i = 0; i < motores_activos; i++){
        if (vel_motores[i] > 0) {
            switch (i) {
            case 0:
                motor1.ConfigComando(0x01,TMCL_ROR,0,0,vel_motores[i]);
                break;
            case 1:
                motor2.ConfigComando(0x01,TMCL_ROR,0,0,vel_motores[i]);
                break;
            case 2:
                motor3.ConfigComando(0x01,TMCL_ROR,0,0,vel_motores[i]);
                break;
            case 3:
                motor4.ConfigComando(0x01,TMCL_ROR,0,0,vel_motores[i]);
                break;
            }
        }
        else if (vel_motores[i] == 0) {
            switch (i) {
            case 0:
                motor1.ConfigComando(0x01,TMCL_MST,0,0,0);
                break;
            case 1:
                motor2.ConfigComando(0x01,TMCL_MST,0,0,0);
                break;
            case 2:
                motor3.ConfigComando(0x01,TMCL_MST,0,0,0);
                break;
            case 3:
```

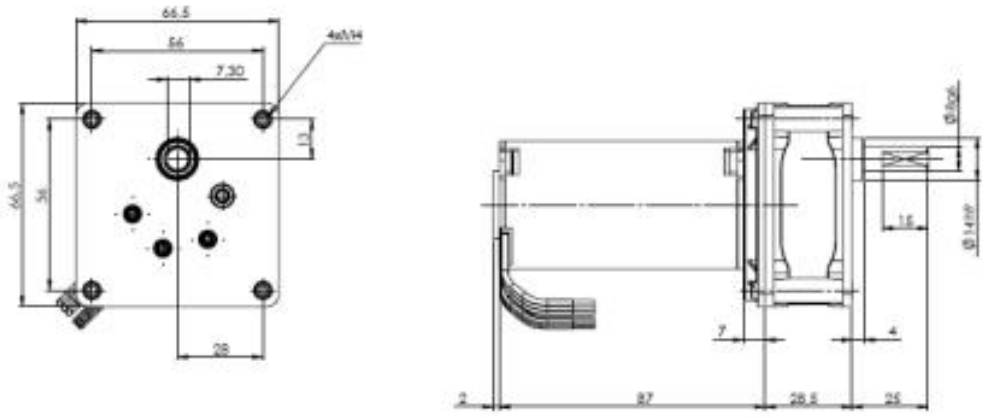
```
                motor4.ConfigComando(0x01,TMCL_MST,0,0,0);
                break;
            }
        }
        else {
            switch (i) {
            case 0:
                motor1.ConfigComando(0x01,TMCL_ROL,0,0,(-1)* (vel_motores[i]));
                break;
            case 1:
                motor2.ConfigComando(0x01,TMCL_ROL,0,0,(-1)* (vel_motores[i]));
                break;
            case 2:
                motor3.ConfigComando(0x01,TMCL_ROL,0,0,(-1)* (vel_motores[i]));
                break;
            case 3:
                motor4.ConfigComando(0x01,TMCL_ROL,0,0,(-1)* (vel_motores[i]));
                break;
            }
        }
    }
}

void PuertoSerieMotor::RWComandos()
{
    int nBytesWritten = 0, nBytesRead = 0;

    //Se escriben los comandos en los respectivos motores
    if (motores_activos >= 1) {
        //Se escribe el comando, asegurándose de que el tamaño de la instrucción es de 9 bytes
        nBytesWritten = write(serialMotor1,motor1.TXbuffer,9);
        if (configIDmotor){
        //Si se ha llegado a enviar el comando completo, se lee la respuesta
        nBytesRead = read(serialMotor1,motor1.RXbuffer,9);
        if (nBytesRead == 9)
            motor1.TraducirRespuesta();
        }
        if (motores_activos >= 2 && nBytesWritten == 9) {
            //Se escribe el comando, asegurándose de que el tamaño de la
            //instrucción es de 9 bytes
            nBytesWritten = write(serialMotor2,motor2.TXbuffer,9);
            if (configIDmotor){
            //Si se ha llegado a enviar el comando completo, se lee la respuesta
            nBytesRead = read(serialMotor2,motor2.RXbuffer,9);
            if (nBytesRead == 9)
                motor2.TraducirRespuesta();
            }
            if (motores_activos >= 3 && nBytesWritten == 9){
                //Se escribe el comando, asegurándose de que el tamaño de la
                //instrucción es de 9 bytes
                nBytesWritten = write(serialMotor3,motor3.TXbuffer,9);

                if (configIDmotor){
                nBytesRead = read(serialMotor3,motor3.RXbuffer,9);
                //Si se ha llegado a enviar el comando completo, se lee la respuesta
```

```
            if (nBytesRead == 9)
               motor3.TraducirRespuesta();
            }
            if (motores_activos >= 4 && nBytesWritten == 9) {
               //Se escribe el comando, asegurándose de que el tamaño de la
               //instrucción es de 9 bytes
               nBytesWritten = write(serialMotor4,motor4.TXbuffer,9);

               if (nBytesWritten == 9 && configIDmotor){
                  //Si se ha llegado a enviar el comando completo, se lee la respuesta
                  nBytesRead = read(serialMotor4,motor4.RXbuffer,9);
                  if (nBytesRead == 9)
                     motor4.TraducirRespuesta();
               }
            }
         }
      }
   }
}

void PuertoSerieMotor::OnSerialDataAvailable()
/*  Función de tratamiento del puerto serie, una vez configurada la comunicación*/
{

   std::cout <<"________INICIO COMUNICACION SERIE________\n";
   //Rutina de control en bucle:
      while (true) {

      //RUTINA DE CONTROL:

         //Se recibe el valor de velocidad
         RecibirVelocidad();

         //Se hace el cálculo de la cinemática
         CalcVelMotores(v,theta,w);

         //Se configuran los respectivos comandos de velocidad para cada rueda:
         ConfigComandosVelMotores();

         //Se escriben las instrucciones binarias del comando TMCL
         // en los respectivos puertos serie
         std::cout << "Enviando comando...\n\n";
         RWComandos();

         //Tiempo de espera para repetir ciclo
         sleep(1);
      //FIN RUTINA DE CONTROL
   }
}
```

ANEXO III: Hoja de características de los motores

KELVIN
www.kelvingear.com

Gearbox + Motor KF65-BG42



KF65

TECHNICAL CHARACTERISTICS

High endurance gearbox for heavy duty continuous workload in any position, at room temperature from -15 to 50°C, with **torque load up to 6.5 Nm, steady load.**

- **Box.** Made of two aluminium plates and an aluminium tubular cover. Frontal mounting by four M4 threaded holes.
- **Gear set.** Hobbed spur gear set with steel pinions and gear wheels, with case superficial heat anti-friction treatment.
- **Output shaft.** Ø8 mm. steel shaft, 25 mm usable length, with a flat. Incorporates and turns on sintered bushings.
- **Output shaft load:**
  Axial direction, pull or push 60 N ≈ 6 Kg.
  Radial direction, at 10 mm from box 60 N ≈ 6 Kg.
- **Lubrication.** Lithium grade 2 grease.
- **Weight.** With maximal number of stages: 0.95 Kg

MOTOR COUPLING:

- **Direct C.:** DUNKER BG42 type 12 or 24 V

■ **OPTIONAL:**

- Speed regulation with electronic controller.

Avoid impacts on the output shaft when assembling or disassembling parts on it, this could damage the gearbox.

Your special requests are welcome.

1

Standard ratios

Gearbox-KF65

KELVIN

BRUSHLESS DC MOTORS MODEL: Dunker BG42															
			BG42x30 12V			BG42x30 24V			BG42x30KI 12V			BG42x30KI 24V		
Reduction ratio i = X:1	Stages	Torque factor	No load speed n_0 (r.p.m.)	Nominal Speed n_N (r.p.m.)	Nominal Torque (N.m)	No load speed n_0 (r.p.m.)	Nominal Speed n_N (r.p.m.)	Nominal Torque (N.m)	No load speed n_0 (r.p.m.)	Nominal Speed n_N (r.p.m.)	Nominal Torque (N.m)	No load speed n_0 (r.p.m.)	Nominal Speed n_N (r.p.m.)	Nominal Torque (N.m)
15,97	3	11,64	262,37	208,52	2,01	257,36	224,17	1,99	269,25	226,05	1,10	257,98	229,81	1,93
35,85	3	25,99	117,53	93,41	4,50	115,29	100,42	4,44	120,62	101,26	2,62	115,57	102,96	4,31
61,77	4	40,53	67,83	53,91	Ex. Torque max. 6,5 Nm	66,54	57,96	Ex. Torque max. 6,5 Nm	69,61	58,44	4,09	66,70	59,41	Ex. Torque max. 6,5 Nm
93,65	4	61,44	44,74	35,56		43,89	38,23		45,92	38,55	6,21	43,99	39,19	
115,4	5	68,14	36,31	28,86		35,62	31,02		37,26	31,28		35,70	31,80	
160,08	5	94,53	26,17	20,80		25,67	22,36		26,86	22,55		25,74	22,93	
191,24	5	112,93	21,91	17,41		21,49	18,72		22,48	18,88		21,54	19,19	
303,15	5	179,01	13,82	10,98		13,56	11,81		14,18	11,91		13,59	12,11	
395,92	5	233,79	10,58	8,41		10,38	9,04		10,86	9,12		10,41	9,27	

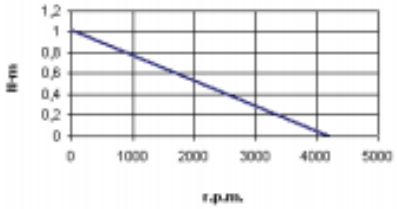
Ex Exceeds maximal admissible torque

NO LOAD SPEED/NOMINAL TORQUE
Motor BG 42x30-12V= 4190 r.p.m./1,02Nm.
Motor BG 42x30-24V= 4110 r.p.m./1,52Nm.
Motor BG 42x30 KI-12V= 4300 r.p.m./0,288Nm.
Motor BG 42x30 KI-24V= 4120 r.p.m./0,445Nm.

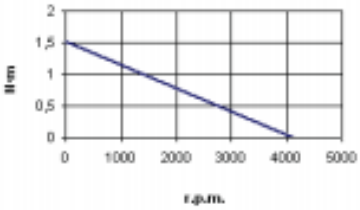
WARNING: The load might reduce final speed up to 40%.

CURVES

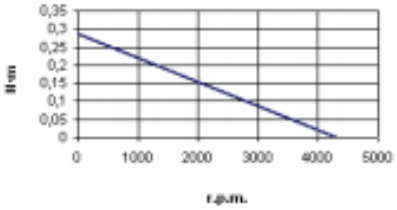
BG42x30 12V



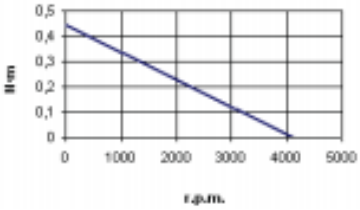
BG42x30 24V



BG42x30KI 12V



BG42x30KI 24V



GEARBOX TIPS:
Noise: noise level depends on load symmetry, location (avoid acoustic resonance), and rotation speed; the lower the speed on the input shaft (motor), the lower the noise.

2

ANEXO IV: Hoja de características de las ruedas Mecanum

Makeblock

100mm Left Mecanum Wheel with 4mm Shaft Connector (SPCC)

SKU: 87055 Weight: 560.00 Gram

What Is 100mm Mecanum Wheel?

Makeblock Mecanum wheels allow your robot to not only travel forward and backward, but also side to side. With 4 mecanum wheels, you can build a car/robot that can move in all direction. This wheel is supposed to work together with other 3 Mecanum wheels (2 right, 1 left).

Features

? Mecanum wheels allow a robot to achieve omni-directional movement while supporting large weights;

? with all Makeblock parts;

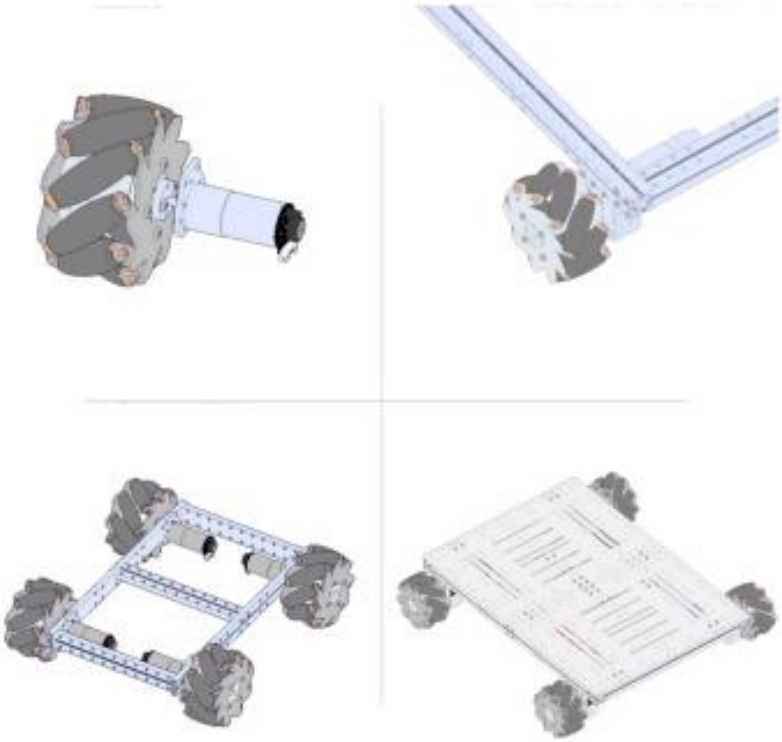
? Work with encoder motor or stepper motor;

? Compatible with 4mm and 8mm motor shaft;

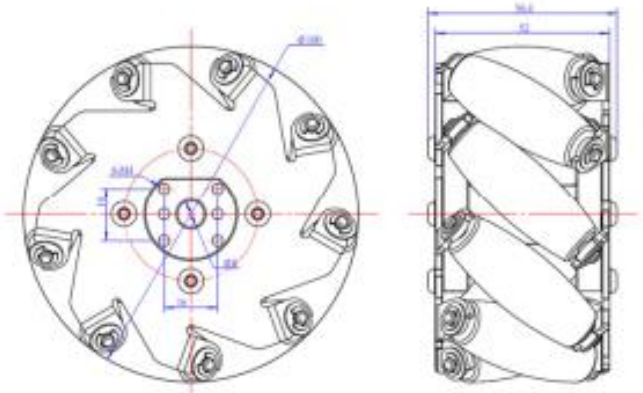
? Cold rolled SPCC steel made, strong rigidity and high impact resistance;

? Easy to assemble;

Demo:



Size Chart:



Specifications	
SKU	87055
Product Name	100mm Left Mecanum Wheel with 4mm Shaft Connector (SPCC)
Gross Weight	560g (19.75oz)
Diameter	4 inches (100mm)
Width	52mm
Rollers	9 x Rubber Roller
Plates	2 x SPCC Plate
Load Capacity	30KG
Package Content (Quantity x Part Name)	1 xLeft Mecanum Wheel 1 x 4mm shaft connector 2 x Screw M4*14 2 x Headless Set Screw M3*5

https://store.makeblock.com/100mm-left-mecanum-wheel-with-4mm-shaft-connector-spcc/2-8-19

ANEXO V: Hoja de características de las baterías

Upower
Unlimited power

UP SERIES-AGM Battery

UP12-12

Specification		
Nominal Voltage	12V	
Nominal Capacity (10HR)	12.0AH	
Dimension	Length	151±1mm (5.95 inches)
	Width	98±1mm (3.86 inches)
	Container Height	95±1mm (3.74 inches)
	Total Height (with Terminal)	101±1mm (3.98 inches)
Approx Weight	Approx 3.80 Kg (8.38 lbs)	
Terminal	T1 / T2	
Container Material	ABS	
Rated Capacity	12.72 AH/0.636A	(20hr, 1.80V/cell, 25°C/77°F)
	12.0 AH/1.20A	(10hr, 1.80V/cell, 25°C/77°F)
	10.4 AH/2.08A	(5hr, 1.75V/cell, 25°C/77°F)
	9.48 AH/3.16A	(3hr, 1.75V/cell, 25°C/77°F)
	7.38 AH/7.38A	(1hr, 1.60V/cell, 25°C/77°F)
Max. Discharge Current	180A (5s)	
Internal Resistance	Approx 18.0mΩ	
Operating Temperature Range	Discharge: -15~50°C (5~122°F)	
	Change : 0~40°C (32~104°F)	
	Storage : -15~40°C (5~104°F)	
Nominal Operating Temperature Range	25±3°C (77±5°F)	
Cycle Use	Initial Charging Current less than 3.6A.Voltage. 14.4V~15.0V at 25°C(77°F)Temp. Coefficient -30mV/°C	
Standby Use	No limit on Initial Charging Current Voltage. 13.5V~13.8V at 25°C(77°F)Temp. Coefficient -20mV/°C	
Capacity affected by Temperature	40°C (104°F)	103%
	25°C (77°F)	100%
	0°C (32°F)	86%
Self Discharge	Batteries may be stored for up to 6 months at 25°C (77°F) and then a freshening charge is requiered.	

Constant Current Discharge (Amperes) at 25°C (77°F)														
F.V/Time	10min	15min	20min	30min	45min	1h	2h	3h	4h	5h	6h	8h	10h	20h
1.85V/cell	15.5	12.9	11.0	8.99	6.80	5.70	3.64	2.884	2.34	1.89	1.66	1.33	1.13	0.630
1.80V/cell	19.8	15.6	13.0	10.6	7.91	6.39	3.97	3.103	2.49	2.03	1.78	1.41	1.20	0.636
1.75V/cell	21.7	17.0	14.0	11.0	8.21	6.68	4.12	3.160	2.56	2.08	1.83	1.43	1.21	0.642
1.70V/cell	23.7	18.2	14.7	11.5	8.54	6.90	4.28	3.248	2.62	2.13	1.86	1.45	1.22	0.654
1.65V/cell	25.6	19.4	15.6	12.1	8.75	7.13	4.40	3.386	2.71	2.19	1.91	1.47	1.25	0.662
1.60V/cell	27.8	20.7	16.7	12.8	9.12	7.38	4.55	3.490	2.79	2.27	1.95	1.49	1.26	0.666

Constant Power Discharge (Watts) at 25°C (77°F)														
F.V/Time	10min	15min	20min	30min	45min	1h	2h	3h	4h	5h	6h	8h	10h	20h
1.85V/cell	29.1	24.6	21.1	17.5	13.3	11.2	7.20	5.72	4.65	3.77	3.32	2.66	2.28	1.270
1.80V/cell	36.8	29.3	24.6	20.3	15.4	12.5	7.80	6.12	4.94	4.03	3.54	2.82	2.41	1.280
1.75V/cell	39.8	31.6	26.2	20.9	15.8	13.0	8.06	6.21	5.04	4.12	3.63	2.86	2.43	1.291
1.70V/cell	42.4	33.3	27.4	21.6	16.4	13.4	8.36	6.37	5.16	4.22	3.70	2.90	2.46	1.314
1.65V/cell	45.4	35.1	28.9	22.6	16.6	13.7	8.55	6.62	5.31	4.32	3.77	2.94	2.50	1.329
1.60V/cell	48.1	36.9	30.4	23.7	17.2	14.1	8.79	6.79	5.46	4.45	3.84	2.96	2.53	1.335

Upower
Unlimited power
66A Tzar Asen Srt.
Sofia, Republic of Bulgary
Tel.(+34) 918 021 649
Fax. (+34) 917 750 542
info@upowerbatteries.com

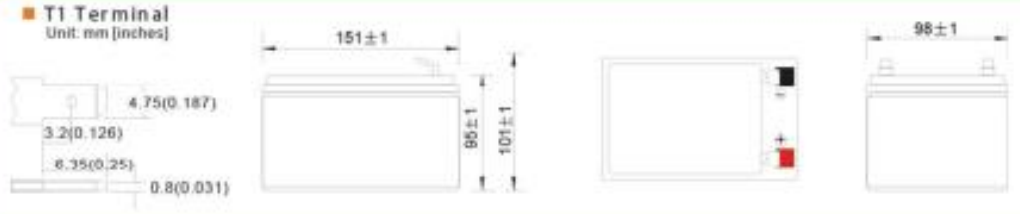


Upower
Unlimited power

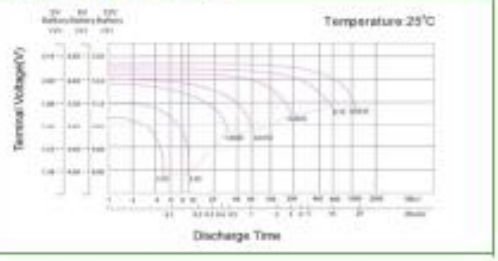
UP SERIES-AGM Battery

UP12-12

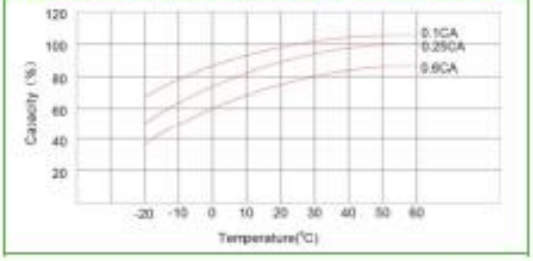
Dimensions



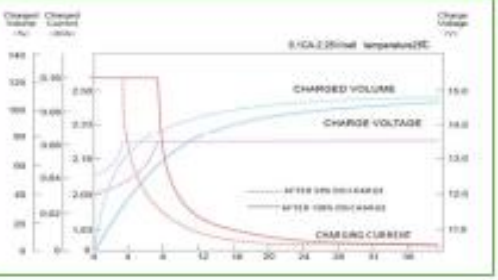
Discharge Characteristics



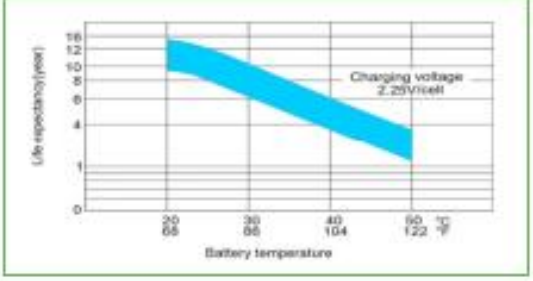
Temperature Effects in Relation to Battery Capacity



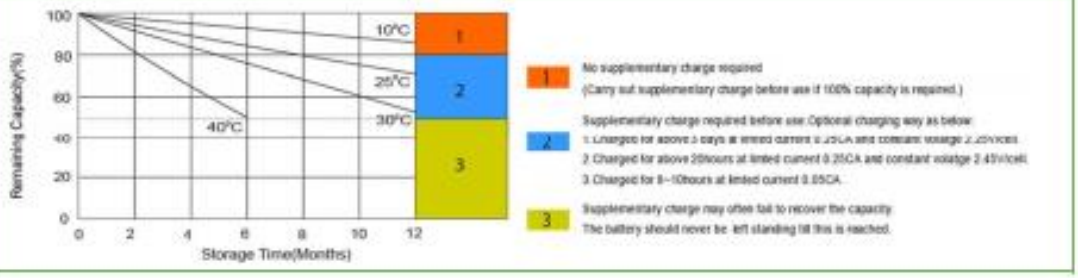
Float Charging Characteristics



Effect of Temperature on Long Term Float Life



Self Discharge Characteristics



Upower
Unlimited power
66A Tzar Asen Srt.
Sofia, Republic of Bulgary
Tel.(+34) 918 021 649
Fax. (+34) 917 750 542
info@upowerbatteries.com



ANEXO VI: Hoja de características de la Raspberry Pi 3B+

Overview

The Raspberry Pi 3 Model B+ is the latest product in the Raspberry Pi 3 range, boasting a 64-bit quad core processor running at 1.4GHz, dual-band 2.4GHz and 5GHz wireless LAN, Bluetooth 4.2/BLE, faster Ethernet, and PoE capability via a separate PoE HAT

The dual-band wireless LAN comes with modular compliance certification, allowing the board to be designed into end products with significantly reduced wireless LAN compliance testing, improving both cost and time to market.

The Raspberry Pi 3 Model B+ maintains the same mechanical footprint as both the Raspberry Pi 2 Model B and the Raspberry Pi 3 Model B.

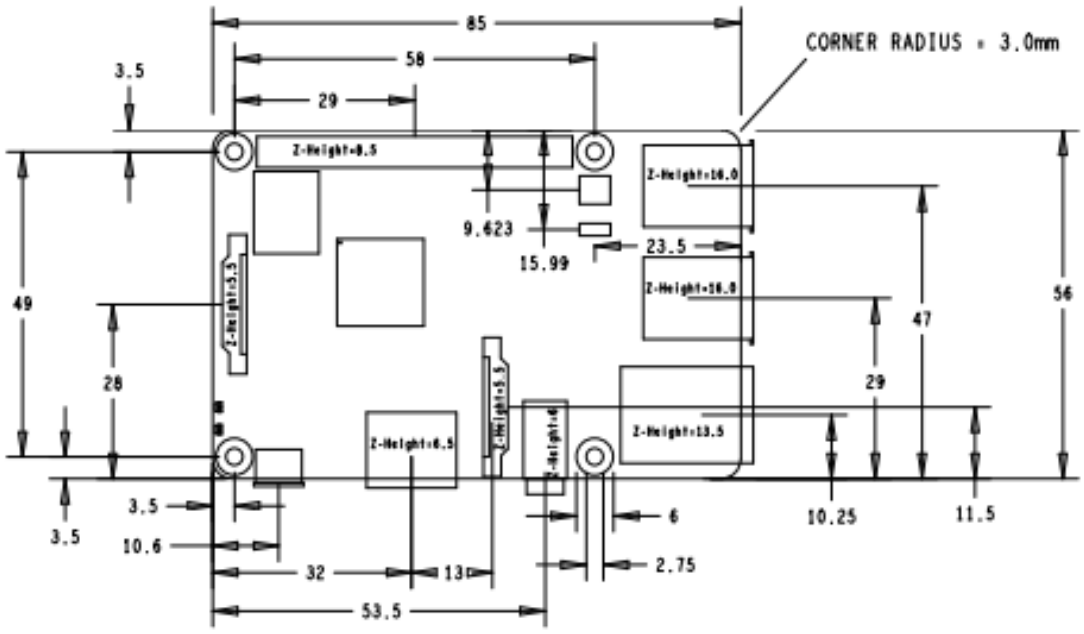
raspberrypi.org

Specifications

Processor:	Broadcom BCM2837B0, Cortex-A53 64-bit SoC @ 1.4GHz
Memory:	1GB LPDDR2 SDRAM
Connectivity:	■ 2.4GHz and 5GHz IEEE 802.11.b/g/n/ac wireless LAN, Bluetooth 4.2, BLE ■ Gigabit Ethernet over USB 2.0 (maximum throughput 300Mbps) ■ 4 × USB 2.0 ports
Access:	Extended 40-pin GPIO header
Video & sound:	■ 1 × full size HDMI ■ MIPI DSI display port ■ MIPI CSI camera port ■ 4 pole stereo output and composite video port
Multimedia:	H.264, MPEG-4 decode (1080p30); H.264 encode (1080p30); OpenGL ES 1.1, 2.0 graphics
SD card support:	Micro SD format for loading operating system and data storage
Input power:	■ 5V/2.5A DC via micro USB connector ■ 5V DC via GPIO header ■ Power over Ethernet (PoE)–enabled (requires separate PoE HAT)
Environment:	Operating temperature, 0–50°C
Compliance:	For a full list of local and regional product approvals, please visit www.raspberrypi.org/products/raspberry-pi-3-model-b+
Production lifetime:	The Raspberry Pi 3 Model B+ will remain in production until at least January 2023.

raspberrypi.org

Physical specifications



Warnings

- This product should only be connected to an external power supply rated at 5V/2.5A DC. Any external power supply used with the Raspberry Pi 3 Model B+ shall comply with relevant regulations and standards applicable in the country of intended use.
- This product should be operated in a well-ventilated environment and, if used inside a case, the case should not be covered.
- Whilst in use, this product should be placed on a stable, flat, non-conductive surface and should not be contacted by conductive items.
- The connection of incompatible devices to the GPIO connection may affect compliance, result in damage to the unit, and invalidate the warranty.
- All peripherals used with this product should comply with relevant standards for the country of use and be marked accordingly to ensure that safety and performance requirements are met. These articles include but are not limited to keyboards, monitors, and mice when used in conjunction with the Raspberry Pi.
- The cables and connectors of all peripherals used with this product must have adequate insulation so that relevant safety requirements are met.

Safety instructions

To avoid malfunction of or damage to this product, please observe the following:

- Do not expose to water or moisture, or place on a conductive surface whilst in operation.
- Do not expose to heat from any source; the Raspberry Pi 3 Model B+ is designed for reliable operation at normal ambient temperatures.
- Take care whilst handling to avoid mechanical or electrical damage to the printed circuit board and connectors.
- Whilst it is powered, avoid handling the printed circuit board, or only handle it by the edges to minimise the risk of electrostatic discharge damage.

raspberrypi.org

ANEXO VII: Hoja de características del regulador de tensión

TRACO POWER

DC/DC Converters

TEN 60 Series, 60 Watt

cULus
UL 60950-1

Features

- Highest power density: 60W in a 51x51x10mm (2"x2"x0.4") package
- Wide 2:1 input voltage range
- Very high efficiency up to 90%
- No minimum load required
- Over temperature protection
- Under voltage lock-out circuit
- Remote On/Off
- Shielded metal case with insulated baseplate
- Optional heatsink
- Lead free design - RoHS compliant
- 3-year product warranty

The TEN 60 series is a family of high performance 60W dc-dc converter modules with wide 2:1 input voltage ranges in a compact low profile case with industry-standard footprint. A very high efficiency allows an operating temperature range of –40°C to 85°C. Built-in filters for both input and output minimizes the need for external filtering. Further standard features include remote On/Off, output voltage trimming, over voltage protection, under voltage lockout and short circuit protection.
Typical applications for these products are battery operated equipment and distributed power architectures in communication and industrial electronics, everywhere where isolated, tightly regulated voltages are required and space is limited on the PCB.

Models

Order code	Input voltage range	Output voltage	Output current max.	Efficiency typ.
TEN 60-2410	18 – 36 VDC (24 VDC nominal)	3.3 VDC	14.0 A	89 %
TEN 60-2411		5.0 VDC	12.0 A	90 %
TEN 60-2412		12 VDC	5.0 A	90 %
TEN 60-2413		15 VDC	4.0 A	90 %
TEN 60-2415		24 VDC	2.5 A	89 %
TEN 60-4810	36 – 75 VDC (48 VDC nominal)	3.3 VDC	14.0 A	89 %
TEN 60-4811		5.0 VDC	12.0 A	90 %
TEN 60-4812		12 VDC	5.0 A	90 %
TEN 60-4813		15 VDC	4.0 A	90 %
TEN 60-4815		24 VDC	2.5 A	89 %

TRACO POWER

DC/DC Converters

TEN 60 — 60 Watt

Input Specifications

Input current at no load (nominal input 24/48 Vin)	3.3 V output models:	100 / 80 mA typ.
	5.0 V output models:	130 / 90 mA typ.
	12 V, 15 V & 24 V output models:	50 / 30 mA typ.
Input current at full load (nominal input 24/48 Vin)	3.3 V output models:	2260 / 1140 mA typ.
	5.0 V output models:	2940 / 1450 mA typ.
	12 V & 15 V output models:	2900 / 1450 mA typ.
	24 V output models:	2940 / 1470 mA typ.
Input voltage variation (dv/dt)		5 V/ms, max. (complies with ETS300 132 part 4.4)
Start-up voltage	24 Vin models:	17 VDC (or lower)
	48 Vin models:	34 VDC (or lower)
Under voltage shut down (lock-out circuit)	24 Vin models:	15 VDC typ.
	48 Vin models:	32 VDC typ.
Surge voltage (100 msec. max.)	24 Vin models:	50 V
	48 Vin models:	100 V
Conducted noise (input)		EN 55022 level A, FCC part 15, level A with external capacitor, see application note: www.tracopower.com/overview/ten60
ESD (input)		EN 61000-4-2, perf. criteria A
Fast transient (input)		EN 61000-4-4, perf. criteria A
Surge (input)		EN 61000-4-5, perf. criteria A

Output Specifications

Voltage set accuracy		±1 %
Output voltage adjustment		±10 %
Regulation	– Input variation Vin min. to Vin max.	0.2 % max.
	– Load variation 0 – 100 %	0.5 % max.
Temperature coefficient		±0.02 %/K max.
Ripple and noise (20 MHz Bandwidth)	3.3 V & 5 V output models:	75 mVpk-pk max.
	12 V & 15 V output models:	100 mVpk-pk max.
	24 V output models:	200 mVpk-pk max.
Start up time (nominal Vin and constant resistive load)		20 ms typ.
Transient response time (25% load change)		250 µs typ.
Short circuit protection		indefinite (automatic recovery)
Over load protection		150 % of Iout max typ.
Minimum Load		not required
Thermal shutdown		at +110°C typ
Over voltage protection	3.3 V output models:	3.7 V
	5 V output models:	5.6 V
	12 V output models:	13.8 V
	15 V output models:	16.8 V
	24 V output models:	30.0 V
Capacitive load	3.3 V output models:	36'000 µF
	5 V output models:	20'400 µF
	12 V output models:	3'550 µF
	15 V output models:	2'300 µF
	24 V output models:	885 µF

TRACO POWER

DC/DC Converters

TEN 60 — **60 Watt**

General Specifications

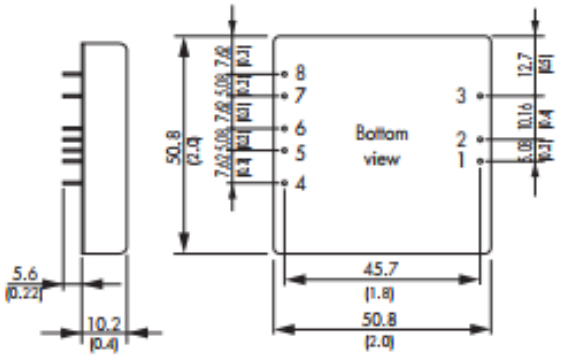
Temperature ranges	– Operating – Case temperature – Storage	–40°C to +85°C +110°C max. –55°C to +125°C
Derating		see application note: www.tracopower.com/overview/ten60
Humidity (non condensing)		95 % rel H max.
Reliability, calculated MTBF (MIL-HDBK-217F, at +70°C, ground benign)		>400'000 h
Isolation (Input/Output)	– Voltage – Capacitance – Resistance	1'600 VDC 1'500 pF max. >1'000 MOhm
Remote On/Off	– On: – Off: – Off idle current:	3.0 ... 12 VDC or open circuit. 0 ... 1.2 VDC or short circuit pin 3 and pin 2 3.0 mA max.
Switching frequency (fixed)		300 kHz typ. (Pulse width modulation PWM)
Vibration		10 – 55Hz, 10G, 30 minutes along X,Y,Z
Safety standards		UL 60950-1, IEC/EN 60950-1
Safety approvals	– Certification documents	www.tracopower.com/overview/ten60
Environmental compliance	– Reach – RoHS	www.tracopower.com/products/reach-declaration.pdf RoHS directive 2011/65/EU

Physical Specifications

Casing material	copper, nickel plated
Baseplate material	none conductive FR4
Potting material	epoxy (UL 94V-0-rated)
Weight	60 g (2.1oz)
Soldering temperature	max. 265°C / 10 sec.

Supporting documents: www.tracopower.com/overview/ten60

Outline Dimensions



Pin-Out	
Pin	
1	+Vin (Vcc)
2	–Vin (GND)
3	Remote On/Off
4	–Sense
5	+Sense
6	+Vout
7	–Vout
8	Trim

Dimensions in [mm], () = Inch
Pin diameter: 1.0 ±0.05 (0.02 ±0.002)
Pin pitch tolerances: ±0.35 (±0.014)
Case tolerances: ±0.5 (±0.02)

All specifications valid at nominal input voltage, full load and +25°C after warm-up time unless otherwise stated.

ANEXO VIII: Hoja de características del kit de evaluación BB-163X

MODULE FOR **STEPPER MOTORS**

MODULE

V 1.0

HARDWARE MANUAL

BB-1630

Baseboard
for TMCM-1630 and TMCM-160
10A / 48V DC
RS232, CAN, RS485, and USB
Encoder interface

TRINAMIC Motion Control GmbH & Co. KG
Hamburg, Germany

www.trinamic.com

TRINAMIC
MOTION CONTROL

Table of contents

1 Life support policy

TRINAMIC Motion Control GmbH & Co. KG does not authorize or warrant any of its products for use in life support systems, without the specific written consent of TRINAMIC Motion Control GmbH & Co. KG.

Life support systems are equipment intended to support or sustain life, and whose failure to perform, when properly used in accordance with instructions provided, can be reasonably expected to result in personal injury or death.

© TRINAMIC Motion Control GmbH & Co. KG 2012

Information given in this data sheet is believed to be accurate and reliable. However neither responsibility is assumed for the consequences of its use nor for any infringement of patents or other rights of third parties, which may result from its use.

Specifications are subject to change without notice.

2 Features

The BB-1630 is the baseboard for the TMCM-1630 and TMCM-160 plug-in modules. The baseboard has been primarily designed for evaluation of the plug-in boards.

Applications
- Baseboard / Evaluation board for TMCM-1630 and TMCM-160

Electrical data
- Supply voltage: +24V DC or +48V DC nominal (+12... +55V DC max.)
- Motor current: up to 10A RMS peak

Interfaces
- The interface connectors refer to the assembly options of the TMCM-1630 and TMCM-160:
  - RS232
  - CAN (2.0B up to 1Mbit/s)
  - RS485
  - USB (High-speed 12Mbit/s)

  The baseboard integrates all communication connectors. There are no assembly options.

  Please note: Functionality of communication interfaces depends on assembly options of TMCM-1630 and TMCM-160 modules

Other
- Separate connectors for interfaces, encoder, power supply, motor, and hall sensors
- Potentiometers for analog inputs plus push-button for digital input (function depending on firmware)
- ROHS compliant
- Size: 120x90mm²

3 Order codes

Order code	Description	Dimensions [mm]
BB-1630	Baseboard for TMCM-1630 and TMCM-160	120 x 90 x 38

Table **3.1**: Order codes

4 Mechanical and electrical interfacing

4.1 Dimensions of the module

The module has a size of 120mm x 90mm. Maximum overall height of the module including connectors and screws: approx. 38mm.

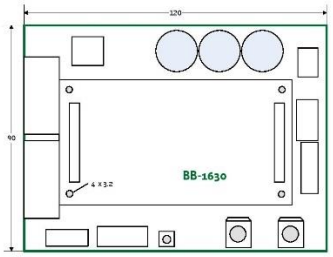


Figure 4.1: Size of module

5 Connectors

The baseboard offers two double row 2.54mm pitch standard connectors (female) for connecting the TMCM-1630 (I/O-connector and supply / motor connector)

Figure 5.1: Connectors of BB-1630

Domain	Connector type	Mating connector
I/O connector	SSW series, socket strip, 2x13 poles, double row, 2.54mm pitch, Samtec	TSM-113-03-L-DV-K-A, 2x13 poles, double row, 2.54mm pitch, SMD vertical, Samtec
Power / motor connector	SSW series, socket strip, 2x13 poles, double row, 2.54mm pitch, Samtec	TSM-113-03-L-DV-K-A, 2x13 poles, double row, 2.54mm pitch, SMD vertical, Samtec
USB	USB, type B, 4 pol., vertical, female	USB, type B, 4 poles, male
RS232	D-sub, 9 poles, female VS-09-BU-D-SUB/10-MPT-0,5	D-sub, 9 poles, male
CAN	D-sub, 9 poles, male VS-09-ST-DSUB/10-MPT-0,5	D-sub, 9 poles, male
RS485	RIA Connect 31383104, 4 poles, grid dimension 3.81mm	RIA Connect 31114104, 4 poles, grid dimension 3.81mm
Motor	RIA Connect 31220103, 3 poles, grid dimension 5.08mm	RIA Connect 31249103, 3 poles grid dimension 5.08mm
Encoder	low profile box header without locking bar, 2.54mm pitch	Female connector, 10 poles, 2.54mm pitch
Power supply	RIA Connect 31220102, 2 poles, grid dimension 5.08mm	RIA Connect 31249102, 2 poles, grid dimension 5.08mm
Hall	RIA Connect 31383105, 5 poles, grid dimension 3.81mm	RIA Connect 31114105, 5 poles, grid dimension 3.81mm

Table 5.1: Connectors of the BB-1630

Since the two connectors of the TMCM-1630 are similar be careful not to plug-in the module the other way round. Also, be sure to place the connectors exactly to their mating opponents. Not following these guidelines might cause permanent damage to the module when turning power supply on.

5.1 Power / motor connector

A double row 26 pin socket strip with 2.54mm pitch is used for connecting all motor related signals and module power supply.

Pin	Label	Description	Pin	Label	Description
1	W	Motor coil W	2	W	Motor coil W
3	W	Motor coil W	4	W	Motor coil W
5	V	Motor coil V	6	V	Motor coil V
7	V	Motor coil V	8	V	Motor coil V
9	U	Motor coil U	10	U	Motor coil U
11	U	Motor coil U	12	U	Motor coil U
13	VM	Module driver supply voltage	14	VM	Module driver supply voltage
15	VM	Module driver supply voltage	16	VM	Module driver supply voltage
17	GND	Module ground (power supply and signal ground)	18	GND	Module ground (power supply and signal ground)
19	GND	Module ground (power supply and signal ground)	20	GND	Module ground (power supply and signal ground)
21	GND	Module ground (power supply and signal ground)	22	GND	Module ground (power supply and signal ground)
23	+5V	+5V output (100mA max.) for encoder and/or hall sensor supply	24	HALL3	Hall sensor 3 signal input
25	HALL1	Hall sensor 1 signal input	26	HALL2	Hall sensor 2 signal input

Table 5.2: Connector for motor related signals and power supply

5.1.1.1 Power supply requirements

The power supply should be designed in a way, that it supplies the nominal motor voltage at the desired maximum motor current. In no case shall the supply voltage exceed the upper or lower voltage limits. To be able to cope with high voltage spikes which might be caused by energy fed back from the motor during deceleration, a sufficient power supply capacitor should be added on the baseboard closed to the module. Depending on the motor and expected motor current please use a 4700µF or larger capacitor with suitable voltage rating. Additionally, a suitable suppressor (zener-) diode might be useful.

5.2 I/O connector

A double row 26 pin socket strip with 2.54mm pitch is used for connecting all communication and GPIO signals.

Pin	Label	Description	Pin	Label	Description
1	+5V	5V analog reference as used by the internal DAC. Max. load 0.5mA	2	Velocity	Used for velocity control in standalone operation by supplying external 0 - 10V signal
3	Torque	Used for max. motor current / torque control in standalone operation by supplying external 0-10V signal	4	GND	Module ground (power supply and signal ground)
5	Dir_IN	5V TTL input. Tie to GND to inverse motor direction, leave open or tie to 5V otherwise.	6	Tacho	This pin outputs a tacho impulse, i.e. toggles on each hall sensor change

Pin	Label	Description	Pin	Label	Description
7	Stop_IN	Emergency stop. Tie this pin to GND to stop the motor (same as the Motor Off switch on PCB). The motor can be restarted via the interface, or by cycling the power supply	8	LED-Temp	5V TTL output: Toggling with 3Hz when temperature pre-warning threshold is exceeded, high when module shut down due to overtemperature
9	LED-Curlim	High, when module goes into current limiting mode	10	+5V	5V output as reference for external purpose
11	GND	GND reference	12	GND	GND reference
13	Enc_A+	Encoder A+ channel	14	Enc_A-	Encoder A- channel
15	Enc_B+	Encoder B+ channel	16	Enc_B-	Encoder B- channel
17	Enc_N+	Encoder N+ channel	18	Enc_N-	Encoder N- channel
19	CANL/USBD-	CAN low / USB D- bus line	20	RXD/ 485-	RXD signal for RS232 / inverting signal for RS485
21	CANH/USBD+	CAN high / USB D+ bus line	22	TXD/ 485+	TXD signal for RS232 / non inverting signal for RS485
23	USB_+VB	Use to detect availability of attached host system (e.g. PC)	24	n.c.	
25	GND	GND reference	26	GND	GND reference

Table 5.3: Connector for communication and GPIOs

5.3 Power connector

Pin	Label	Description
1	GND	Module ground (power supply and signal ground)
2	VDD	Power supply input

Table 5.4: Power connector

5.4 Motor connector

Pin	Label	Description
1	U	Motor coil U
2	V	Motor coil V
3	W	Motor coil W

Table 5.5: Motor connector

5.5 Hall connector

Pin	Label	Description
1	Hall1	Hall sensor 1 signal input
2	Hall2	Hall sensor 2 signal input
3	Hall3	Hall sensor 3 signal input
4	GND	Signal and module ground
5	+5V_Hall	+5V output (100mA max.) for encoder and/or hall sensor supply

Table 5.6: Hall connector

5.6 Encoder connector

Pin	Label	Description
1, 2	GND	Module and signal ground
3	Encoder_N+	Encoder channel N+
4	Encoder_N-	Encoder channel N-
5	Encoder_A+	Encoder channel A+
6	Encoder_A-	Encoder channel A-
7, 8	+5V_output	+5V output
9	Encoder_B+	Encoder channel B+
10	Encoder_B-	Encoder channel B-

Table **5.7**: Encoder connector

5.7 Interface connectors of BB-1630

5.7.1 USB interface

Pin	Label	Description
1	USB_+VB	Board is self-powered – just use to detect availability of attached host system (e.g. PC)
2	USB-	Differential USB bus
3	USB+	Differential USB bus
4	GND	System / module ground

Table **5.8**: USB connector

Note: the USB interface can only be used if the RS485 termination jumper is not set.

5.7.2 CAN interface

Pin	Label	Description
2	CAN_L	CAN differential bus
7	CAN_H	CAN differential bus
3	GND	System / board ground
1, 4, 5, 6, 8, 9	n.c.	Pins not used / not connected

Table **5.9**: CAN connector

5.7.3 RS232 interface

Pin	Label	Description
2	RS232_TxD	RS232 transmit serial data
3	RS232_RxD	RS232 receive serial data
5	GND	System / board ground
1, 4, 6, 7, 8, 9	n.c.	Pins not used / not connected

Table **5.10**: RS232 connector

5.7.4 RS485 interface

Pin	Label	Description
1	RS485-	RS485 differential bus (connected to pin 3)
2	RS485+	RS485 differential bus (connected to pin 4)
3	RS485-	RS485 differential bus (connected to pin 1)
4	RS485+	RS485 differential bus (connected to pin 2)

Table **5.11**: RS485 connector

6 Jumpers

There are six jumpers on the Baseboard BB-1630 for activating/deactivating special functions.
Further, it is possible to access signals via the jumper strips.

Note: Pin1 is always a square solder pad.

SIGNALS

Section	Pin 1	Pin 2	Pin 3
CAN	CANH	CANL	N/A
RS485	485-	485+	N/A
Stop	Stop	GND	N/A
Direction	Direction	GND	N/A
Torque	GND	Torque	+5V to GND[1]

Table **6.1**: Signals on jumper strips

[1]) Actual voltage depends on potentiometer position.

FUNCTIONS WITH/WITHOUT JUMPERS

Section	Jumper set/unset	Description
CAN	Jumper set	Termination resistor 120Ω applied
	Jumper unset	Resistor not applied
RS485	Jumper set	Termination resistor 120Ω applied
	Jumper unset	Resistor not applied
Stop	Jumper set	Permanent stop signal
	Jumper unset	No stop signal
Direction	Jumper set	Direction signal is GND
	Jumper unset	Direction signal is set by potentiometer
Torque	Jumper set pin1 - pin2	Torque signal is GND
	Jumper set pin2 - pin3	Torque signal is set by potentiometer
	Jumper not set	No external signal applied
Velocity	Jumper set pin1 - pin2	Velocity signal is GND
	Jumper set pin2 - Pin3	Velocity signal is set by potentiometer
	Jumper not set	No external signal applied

Table **6.2**: Jumper setting

It is necessary to remove the RS485 termination jumper before using the USB interface!

7 LEDs

The BB-1630 module has 4 on-board LEDs:

- Power
- Current overload
- temperature warning
- GPIOs

Please note: function of LEDs might depend on firmware version

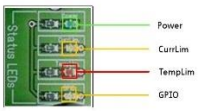


Table **7.1**: Status LEDs of BB-1630

8 Operational ratings

The operational ratings show the intended/the characteristic range for the values and should be used as design values. An operation within the limiting values is possible, but shall not be used for extended periods, because the unit life time may be shortened. In no case shall the limiting values be exceeded.

Symbol	Parameter	Min	Typ	Max	Unit
V_S	Power supply voltage for operation	12	24, 48	55	V
I_S	Power supply current			10	A

Table **8.1**: Operational ratings

9 Revision history

9.1 Document revision

Version	Date	Author	Description
1.00	2011-JUN-02	SD	Initial version
1.01	2011-NOV-03	SD	Minor changes
1.02	2012-JUN-11	SD	Information for USB interface added

Table **9.1**: Document revision

9.2 Hardware revision

Version	Date	Description
1.00	2011-MAR-31	First version for TMCM-1630 and TMCM-160

Table **9.2**: Hardware revision

10 References

[TMCM-1630] TMCM-1630 Hardware Manual
[TMCM-1630] TMCM-1630 TMCL™ Firmware Manual
[TMCM-160] TMCM-160 Hardware Manual
[TMCM-160] TMCM-160 Firmware Manual
[TMCL-IDE] TMCL-IDE User Manual

Please refer to www.trinamic.com.

ANEXO IX: Hoja de características del driver TMC-1630

2 Features

The TMCM-1630 is a highly integrated single axis BLDC servo controller module with several interface-options. The highly integrated module (size: 50mm x 92.5 mm) has been designed in order to be plugged onto a baseboard. It integrates velocity and position control and offers hall sensor and incremental encoder (a/b/n) inputs. The module can be used in stand alone operation or remote controlled.

Applications
- Demanding single and multi-axis BLDC motor solutions

Electrical data
- Supply voltage: +24V DC or +48V DC nominal (+15_ +55V DC max.)
- Motor current: up to 10A RMS (programmable) peak

Integrated motion controller
- High performance ARM Cortex™-M3 microcontroller for system control and communication protocol handling

Integrated motor driver
- High performance integrated pre-driver (TMC603A)
- High-efficient operation, low power dissipation (MOSFETs with low $R_{DS(ON)}$)
- Dynamic current control
- Integrated protection
- On the fly alteration of motion parameters (e.g. position, velocity, acceleration)

Interfaces
- Two standard assembly options:
  - RS232 and CAN (2.0B up to 1Mbit/s)
  - RS485 and USB (High-speed 12Mbit/s)
- 2 analogue and 2 digital inputs
- 3 open drain outputs

Motor type
- Block commutated 3 phase BLDC motors with optional hall sensors / optional encoder
- Motor power from a few Watts to nearly 500W
- Motor velocity up to 100,000 RPM (electrical field)
- Common supply voltages of 12V DC, 24V DC, 36V DC and 48V DC supported
- Coil current up to 10A peak

Software
- TMCL™ stand-alone operation or remote controlled operation
- TMCL™ program memory (non volatile) for up to 2048 TMCL™ commands
- TMCL™ PC-based application development software TMCL-IDE and TMCL-BLDC available for free
- CANopen: for CANopen support please consider using the TMCM-1633

Other
- Two double-row 2.54mm connectors
- ROHS compliant
- Size: 50x92.5mm²

Please see separate TMCL™ Firmware Manual for additional information

5 Connectors

The module offers two double row 2.54mm pitch standard connectors, one at each end of the board.

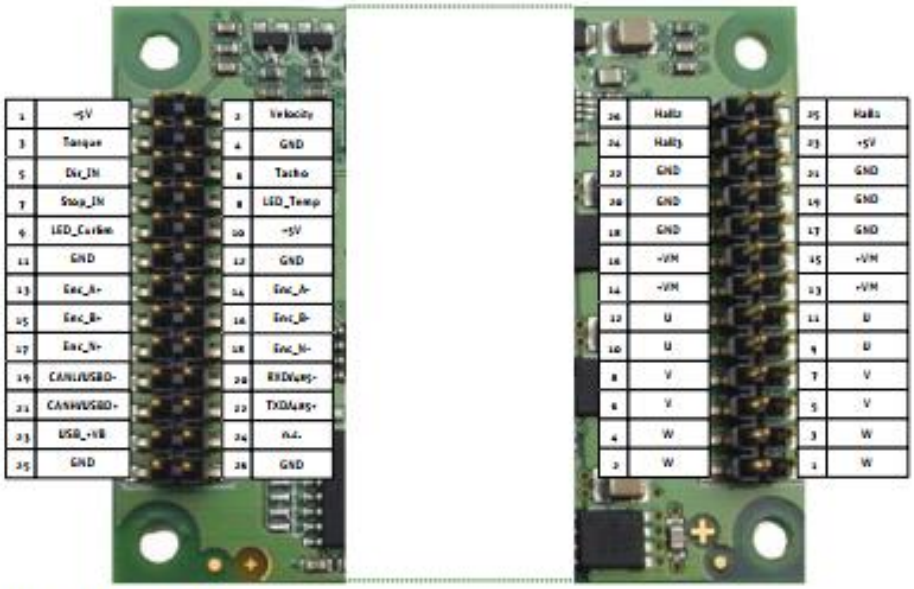


Figure 5.1: Connectors

Domain	Connector type	Mating connector type
I/Os, interfaces, encoder	TSM-113-03-L-DV-K-A, 2x13 poles, double row, 2.54mm pitch, SMD vertical, Samtec	SSW, SSQ, SSM, BSW, ESW, ESQ, BCS, SLW, CES, HLE , IDSS and IDSD series, Samtec
Power, motor	TSM-113-03-L-DV-K-A, 2x13 poles, double row, 2.54mm pitch, SMD vertical, Samtec	SSW, SSQ, SSM, BSW, ESW, ESQ, BCS, SLW, CES, HLE , IDSS and IDSD series, Samtec

Table 5.1: Connector type and mating connector of the TMCM-1630

Since the two connectors of the TMCM-1630 are similar be careful not to plug-in the module the other way round. Also, be sure to place the connectors exactly to their mating opponents. Not following these guidelines might cause permanent damage to the module when turning power supply on.

5.2 Communication, GPIO, and encoder connector

A double row 26 pin header with 2.54mm pitch is used for connecting all communication and GPIO signals.

Pin	Label	Description	Pin	Label	Description
1	+5V	5V analog reference as used by the internal DAC. Max. load 0.5mA	2	Velocity	Used for velocity control in stand alone operation by supplying external 0 – 10V signal
3	Torque	Used for max. motor current / torque control in stand alone operation by supplying external 0-10V signal	4	GND	Module ground (power supply and signal ground)
5	Dir_IN	5V TTL input. Tie to GND to inverse motor direction, leave open or tie to 5V otherwise.	6	Tacho	This pin outputs a tacho impulse, i.e. toggles on each hall sensor change
7	Stop_IN	Emergency stop. Tie this pin to GND to stop the motor (same as the *Motor Off* switch on PCB). The motor can be restarted via the interface, or by cycling the power supply	8	LED-Temp	5V TTL output: Toggling with 3Hz when temperature pre-warning threshold is exceeded, high when module shut down due to overtemperature
9	LED-Curlim	High, when module goes into current limiting mode	10	+5V	5V output as reference for external purpose
11	GND	GND reference	12	GND	GND reference
13	Enc_A+	Encoder A+ channel	14	Enc_A-	Encoder A- channel
15	Enc_B+	Encoder B+ channel	16	Enc_B-	Encoder B- channel
17	Enc_N+	Encoder N+ channel	18	Enc_N-	Encoder N- channel
19	CANL/USBD-	CAN low / USB D- bus line	20	RXD/ 485-	RXD signal for RS232 / inverting signal for RS485
21	CANH/USBD+	CAN high / USB D+ bus line	22	TXD/ 485+	TXD signal for RS232 / non inverting signal for RS485
23	USB_+VB	Use to detect availability of attached host system (e.g. PC)	24	n.c.	
25	GND	GND reference	26	GND	GND reference

Table 5.4: Connector for communication and GPIOs

5.2.1 Reset the module to factory defaults

Interface	Description
RS232	Short RxD and TxD for resetting the module.
USB	Use your USB interface for resetting the module with the functions of TMCL-IDE. Please refer to the TMCL™ Firmware Manual.

Table 5.5: Reset the module to factory defaults

7 Operational ratings

The operational ratings show the intended/the characteristic range for the values and should be used as design values. An operation within the limiting values is possible, but shall not be used for extended periods, because the unit life time may be shortened. In no case shall the limiting values be exceeded.

Symbol	Parameter	Min	Typ	Max	Unit
V_S	Power supply voltage for operation	15	24, 48	55	V
I_S	Power supply current	0.04		I_{MOT}	A
P_{ID}	Module idle power consumption		1.2		W
V_5	5 Volt (+-8%) output external load (hall sensors plus other load)			100	mA
I_{MC}	Continuous Motor current at V_{MS}		0 – 8	10	A
I_{MP}	Short time Motor current in acceleration periods		0 – 10		A
V_I	Logic input voltage on digital / hall sensor inputs	-0.3		V_{CC}+ 0.3	V
I_D	Sink current on digital outputs (open-drain outputs)			1	A
V_{IA}	Analog input voltage	-24	0 – 10	24	V
f_{CHOP}	Chopper frequency		20		kHz
E_A	Exactness of voltage and current measurement	-8		+8	%
T_{SL}	Motor output slope (U, V, W)		100		ns
T_O	Environment temperature operating	-25		+70	°C
T_{OF}	Environment temperature for operation at full specified current (air flow required, depending upon motor / voltage)	-25		+60	°C
T_{board}	Temperature of the module, as measured by the integrated sensor.		<100	125	°C

Table 7.1: Operational ratings

8 Functional description

In figure 8.1 the main parts of the TMCM-1630 module are shown. The module mainly consists of the Cortex™-M3 CPU, TRINAMICs highly integrated TMC603 BLDC motor pre-driver, the MOSFET driver stage, different interfaces (depends up-on which option you have chosen), the inputs, and the outputs (open drain).

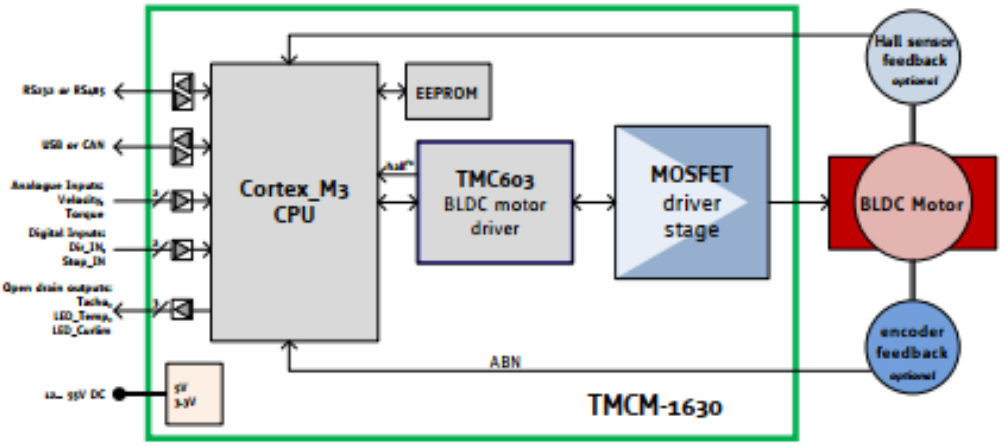


Figure 8.1: Main parts of the TMCM-1630

8.1 System architecture

The TMCM-1630 integrates a microcontroller with the TMCL™ Firmware or CANopen. The motion control real-time tasks are realized by the TMC603A.

8.1.1 Microcontroller

On this module, the ARM Cortex-M3™ CPU 32-bit processor is used to run the TMCL™ operating system and to control the TMC603A. The flash memory of the microcontroller holds the TMCL™ operating system. The EEPROM memory is used to permanently store configuration data. The microcontroller runs the TMCL™ or CANopen operating system which makes it possible to execute commands that are sent to the module from the host via the interface. The microcontroller interprets the commands and controls the TMC603A.

The TMCL™ operating system can be updated via the host interface. Please use the latest version of the TMCL-IDE to do this.

8.1.2 TMC603A pre-driver

The TMC603A is a three phase motor driver for highly compact and energy efficient drive solutions. It contains all power and analog circuitry required for a high performance BLDC motor system. The TMC603A is designed to provide the frontend for a microcontroller doing motor commutation and control algorithms. Protection and diagnostic features as well as a step down switching regulator reduce system cost and increase reliability.

ANEXO X: Planos

Listado de planos

1.0.0 Plataforma omnidireccional. Dimensiones principales.

1.0.1 Plataforma omnidireccional. Plano de conjunto explosionado

1.1.0 Montaje de la rueda. Vista explosionada

1.1.1 Soporte motores.

1.1.2 Acoplamiento motor.

1.2.0 Montaje baterías. Vista explosionada.

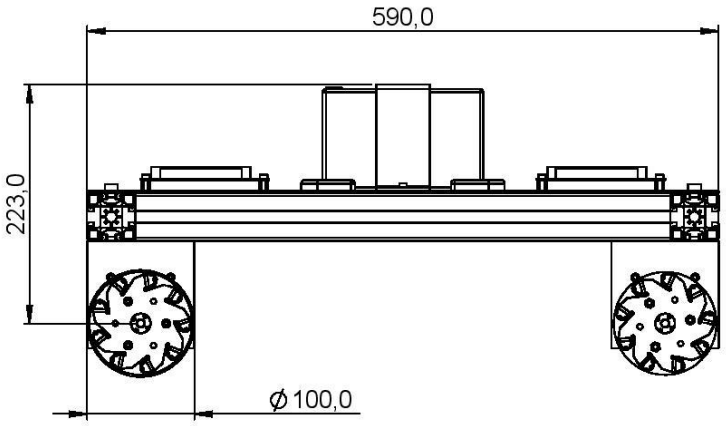
1.2.1 Soporte vertical baterías.

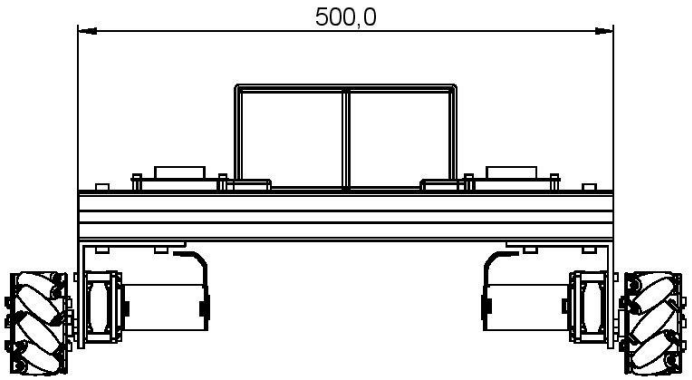
1.2.2 Soporte horizontal baterías.

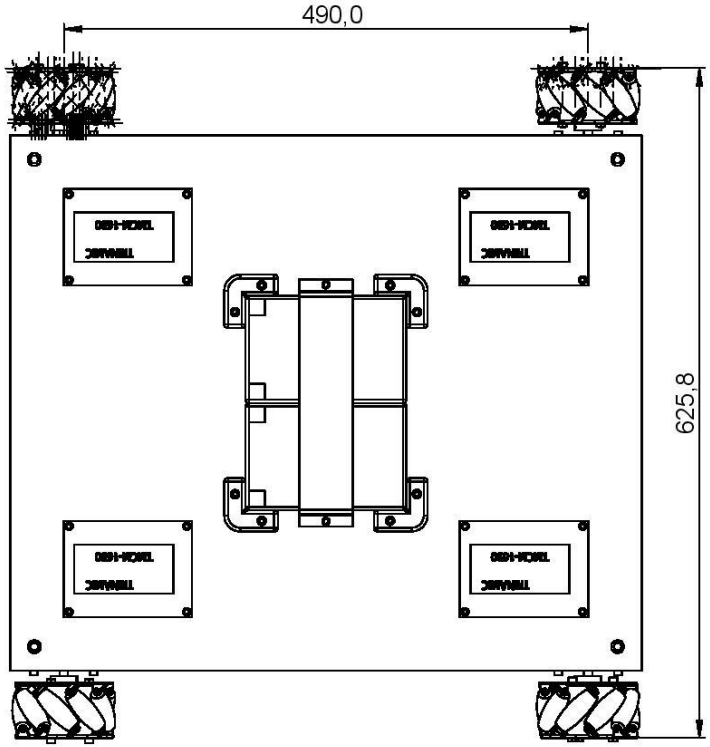
1.3.0 Montaje bastidor. Dimensiones principales.

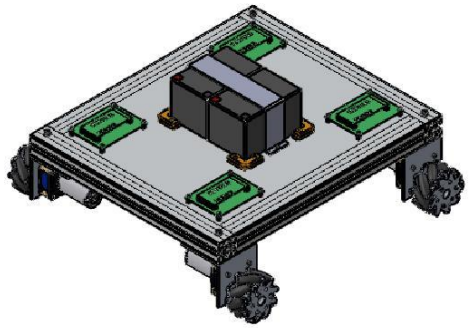
1.3.1 Montaje bastidor. Vista explosionada.

1.3.2 Chapa bastidor









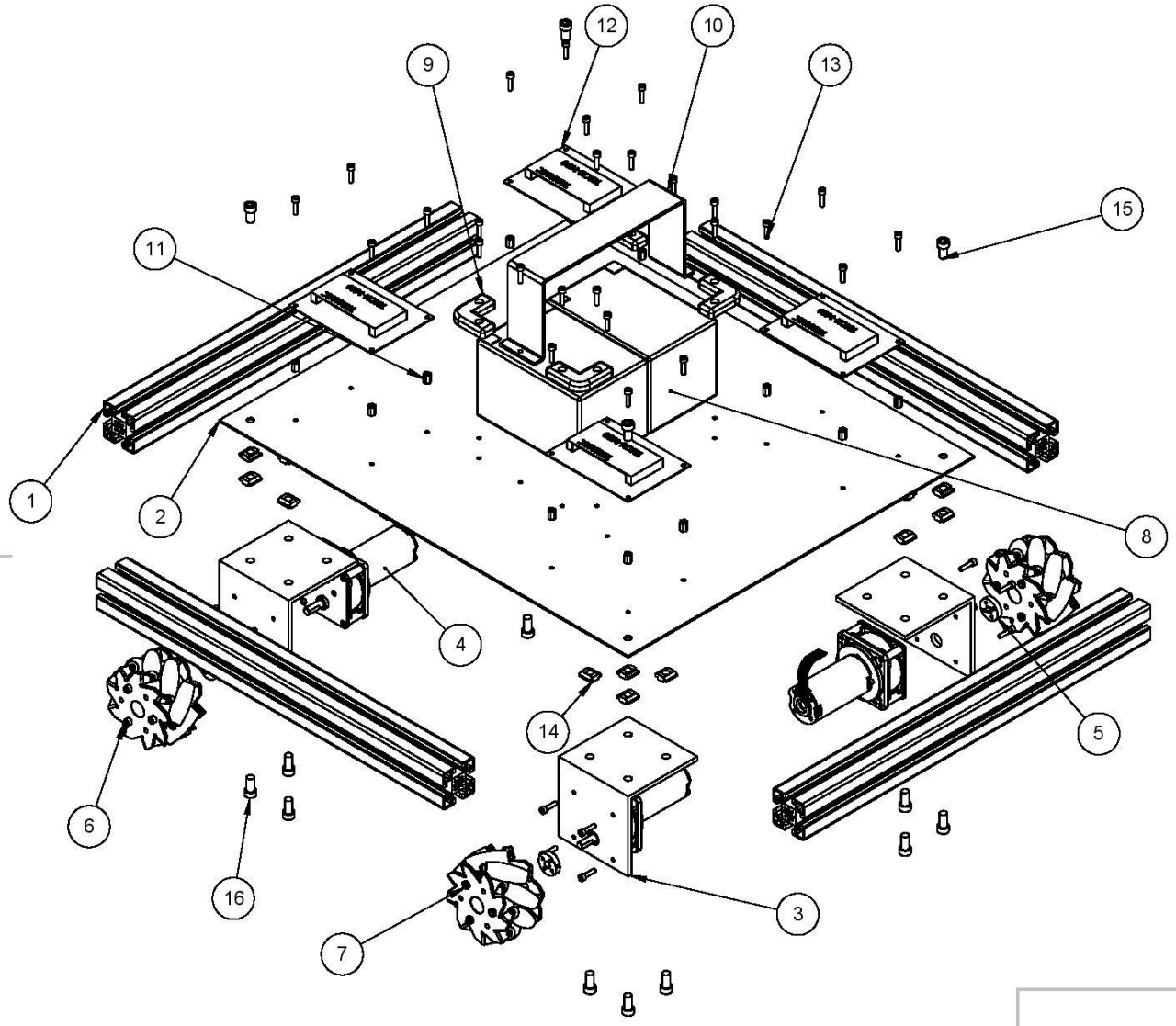
MÁSTER EN INGENIERÍA MECATRÓNICA
-TRABAJO FIN DE MÁSTER-

	FECHA	NOMBRE	FIRMA
Proyectado	28/06/19	Fernando Peña	Fernando
Dibujado	28/06/19	Fernando Peña	Fernando
Comprobado	28/06/19	Fernando Peña	Fernando

ESCALA:1:5

Plataforma omnidireccional.
Dimensiones Principales.

PLANO Nº: 1.0.0
Sustituye a:
Sustituido por:



N.º DE ELEMENTO	N.º DE PIEZA	CANTIDAD
1	Perfil aluminio 45x45x500	4
2	Chapa 590x500	1
3	Soporte motores perfil	4
4	Motor KF65 DKM BG42x30	4
5	Acoplamiento mecanum	4
6	Mecanum derecha	2
7	Mecanum Izquierda	2
8	Bateria12V	2
9	Fijación batería L	4
10	Fijacion vertical	1
11	Separador hexagonal	16
12	Trinamic Board	4
13	M4x16	42
14	Tuerca T M8	16
15	M8x14	4
16	M8x16	12

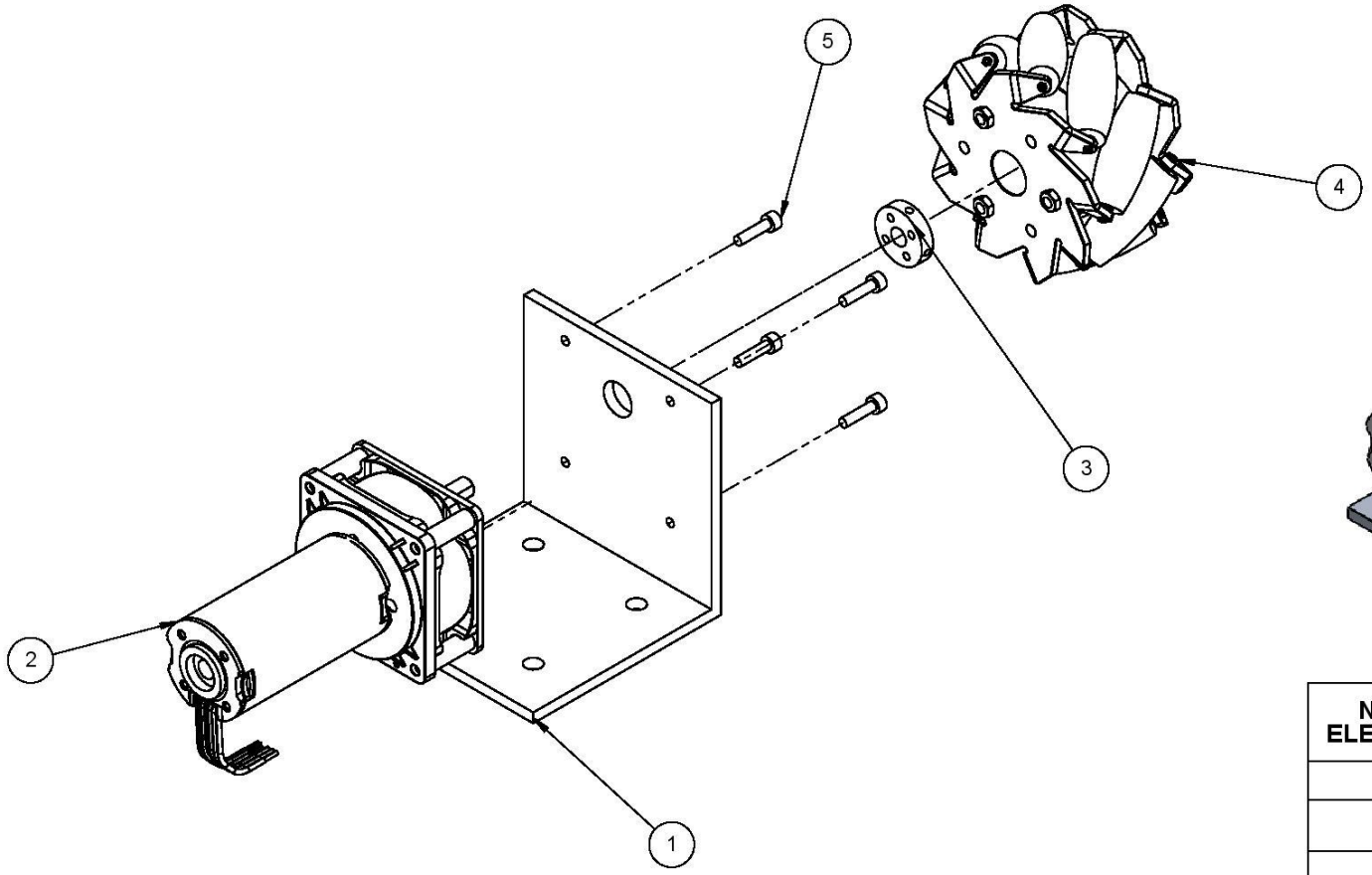
MÁSTER EN INGENIERÍA MECATRÓNICA
-TRABAJO FIN DE MÁSTER-

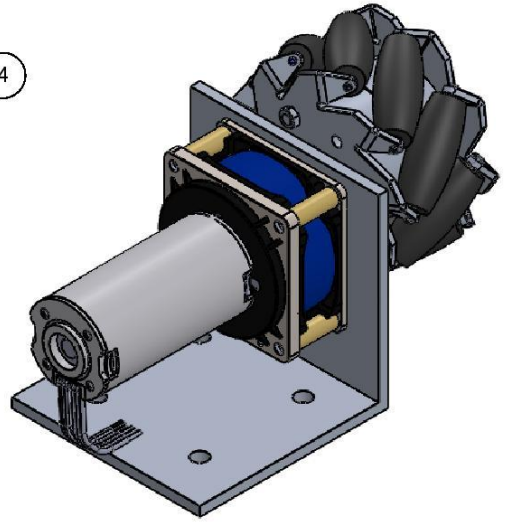
	FECHA	NOMBRE	FIRMA
Proyectado	30/06/19	Fernando Peña	Fernando
Dibujado	30/06/19	Fernando Peña	Fernando
Comprobado	30/06/19	Fernando Peña	Fernando

ESCALA:1:5

Plataforma omnidireccional.
Plano de conjunto explosionado.

PLANO Nº: 1.0.1
Sustituye a:
Sustituido por:





N.º DE ELEMENTO	N.º DE PIEZA	CANTIDAD
1	Soporte motores perfil	1
2	Motor KF65 DKM BG42x30	1
3	Acoplamiento mecanum	1
4	Mecanum derecha	1
5	M4x16	4

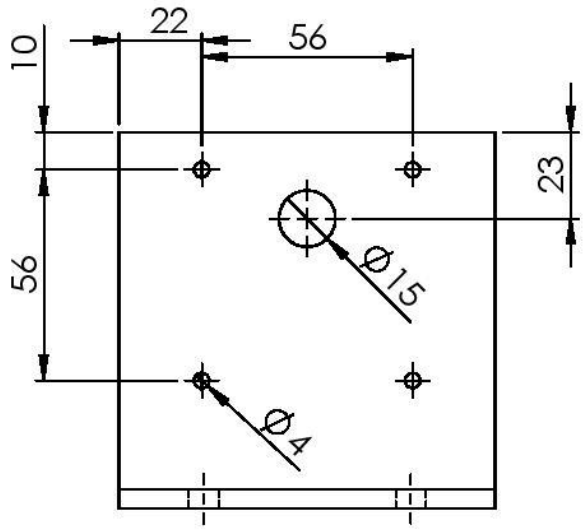
MÁSTER EN INGENIERÍA MECATRÓNICA
-TRABAJO FIN DE MÁSTER-

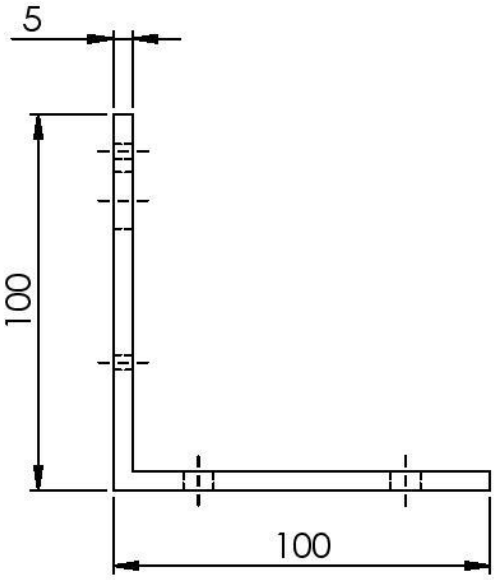
	FECHA	NOMBRE	FIRMA
Proyectado	15/06/19	Fernando Peña	Fernando
Dibujado	15/06/19	Fernando Peña	Fernando
Comprobado	15/06/19	Fernando Peña	Fernando

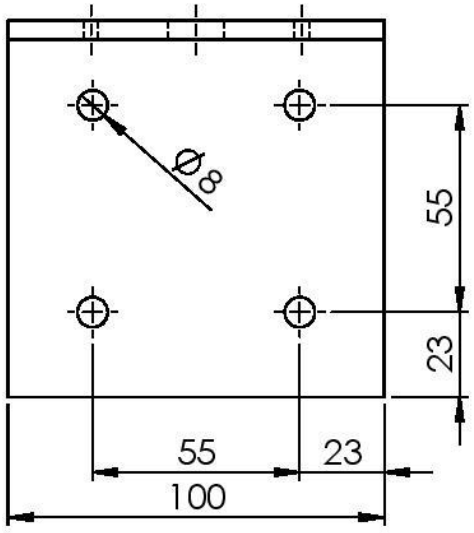
ESCALA:1:5

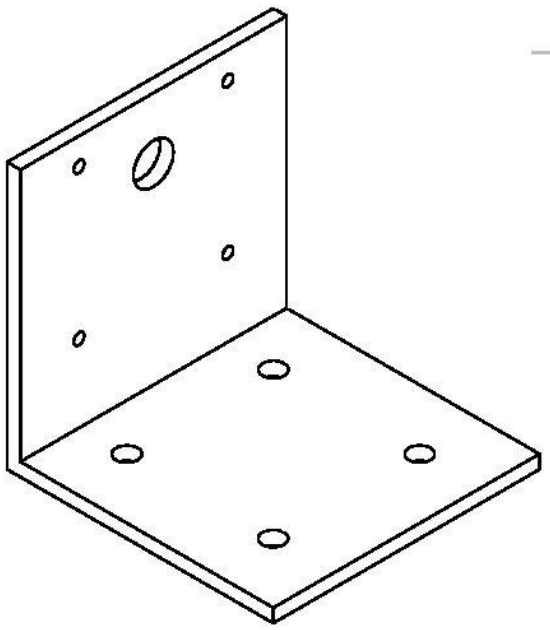
Montaje de la rueda
Vista explosionada

PLANO Nº: 1.1.0
Sustituye a:
Sustituido por:







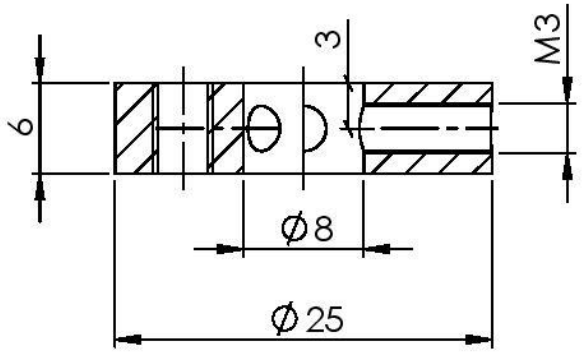


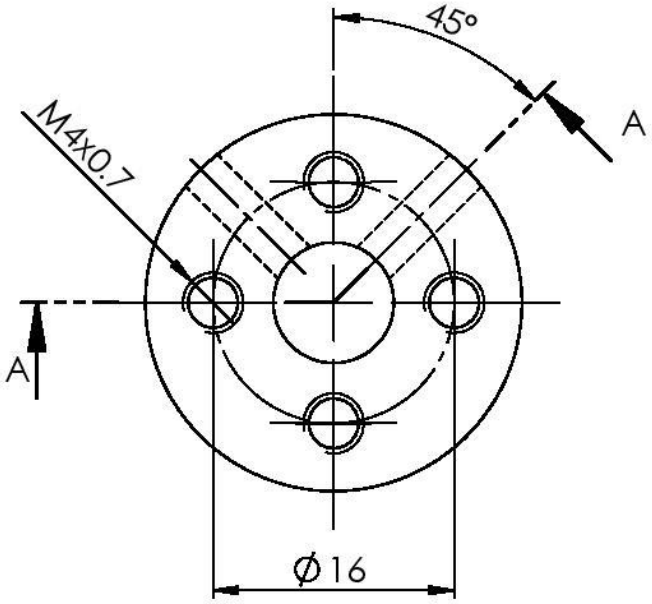
MÁSTER EN INGENIERÍA MECATRÓNICA
-TRABAJO FIN DE MÁSTER-

	FECHA	NOMBRE	FIRMA
Proyectado	15/06/19	Fernando Peña	Fernando
Dibujado	15/06/19	Fernando Peña	Fernando
Comprobado	15/06/19	Fernando Peña	Fernando

ESCALA:1:2	Soporte motores	PLANO Nº: 1.1.1
		Sustituye a:
		Sustituido por:

SECCIÓN A-A

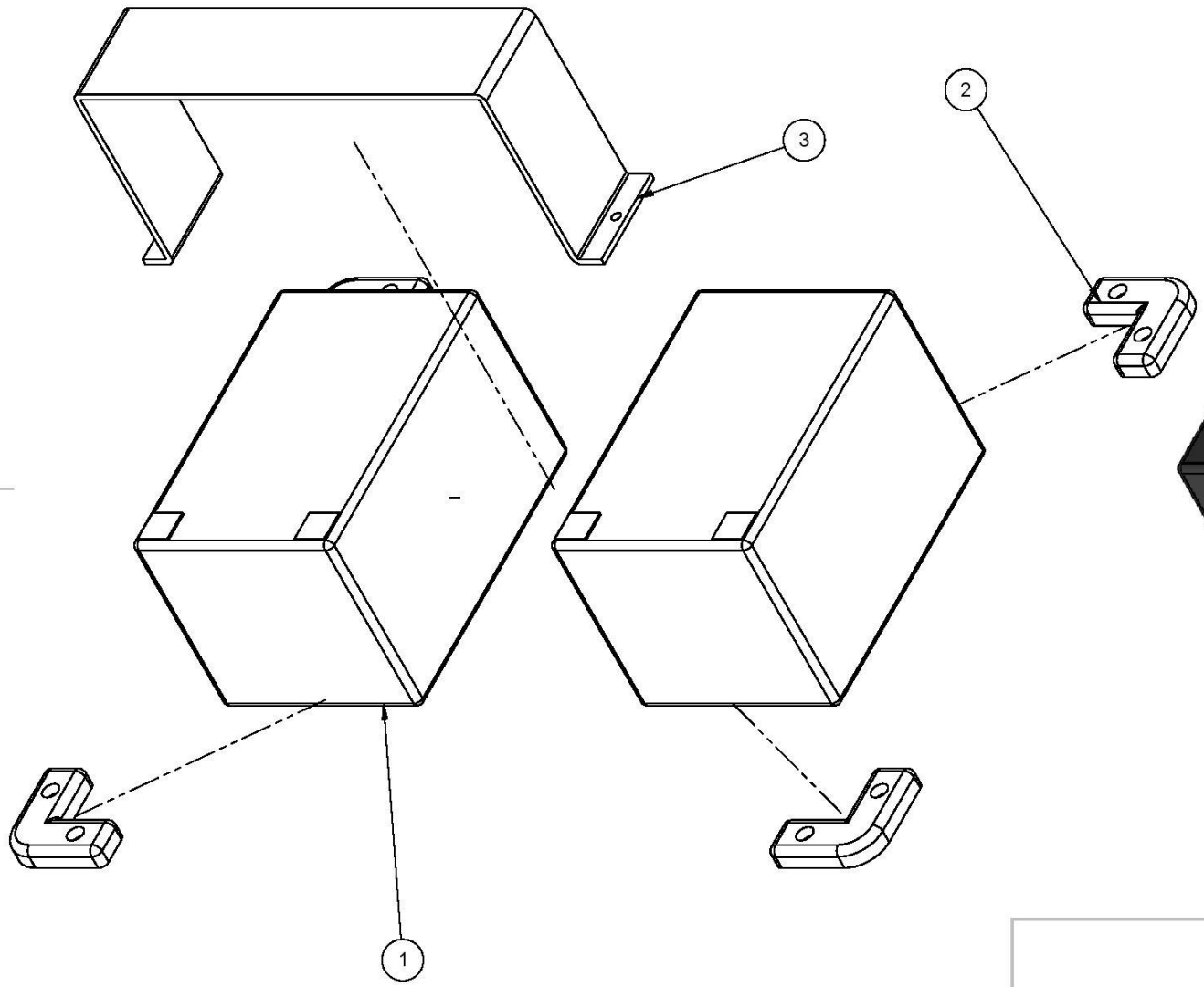


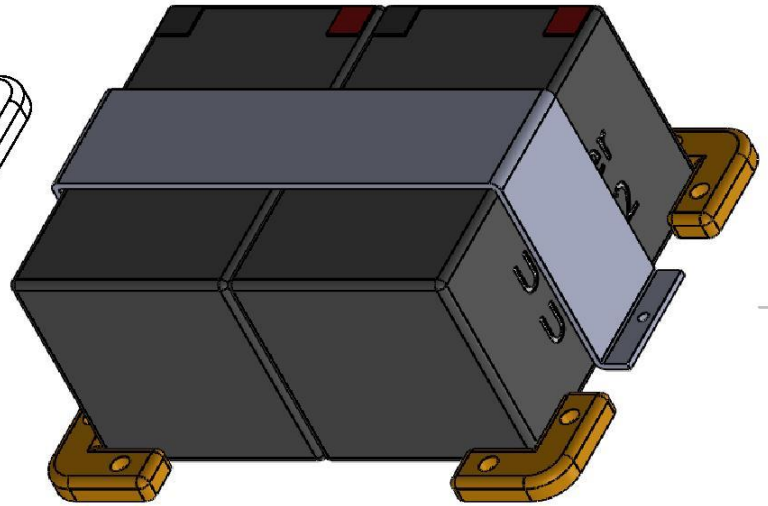


MÁSTER EN INGENIERÍA MECATRÓNICA
-TRABAJO FIN DE MÁSTER-

	FECHA	NOMBRE	FIRMA
Proyectado	15/06/19	Fernando Peña	Fernando
Dibujado	15/06/19	Fernando Peña	Fernando
Comprobado	15/06/19	Fernando Peña	Fernando

ESCALA:2:1	Acoplamiento del motor	PLANO Nº: 1.1.2
		Sustituye a:
		Sustituido por:





N.º DE ELEMENTO	N.º DE PIEZA	CANTIDAD
1	Bateria12V	2
2	Fijación batería L	4
3	Fijacion vertical	1

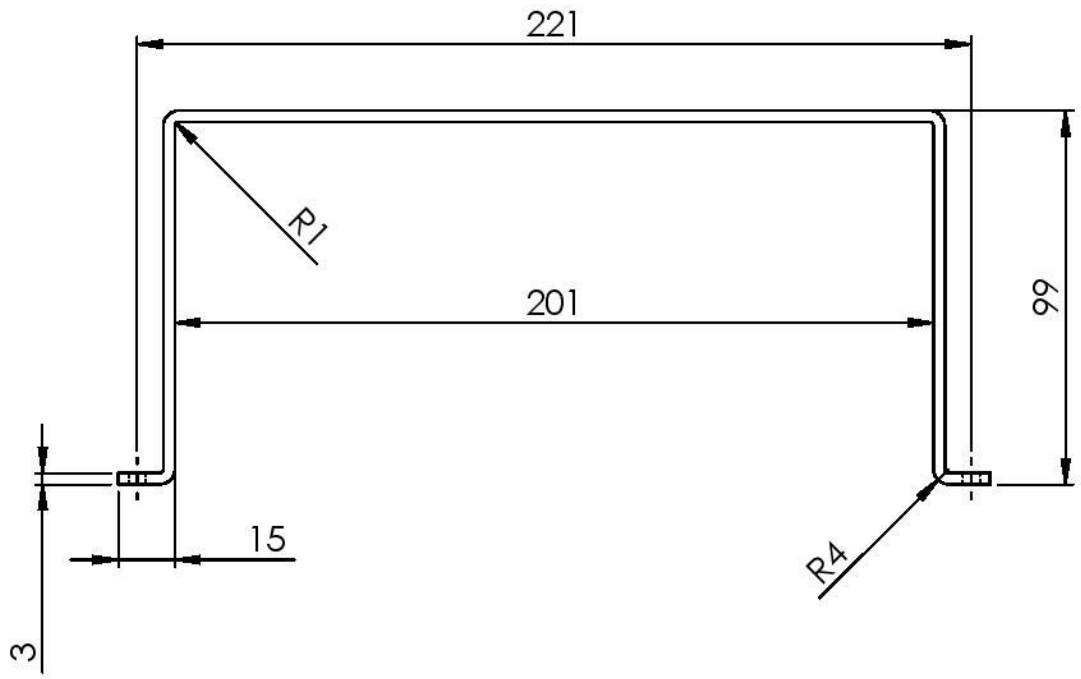
MÁSTER EN INGENIERÍA MECATRÓNICA
-TRABAJO FIN DE MÁSTER-

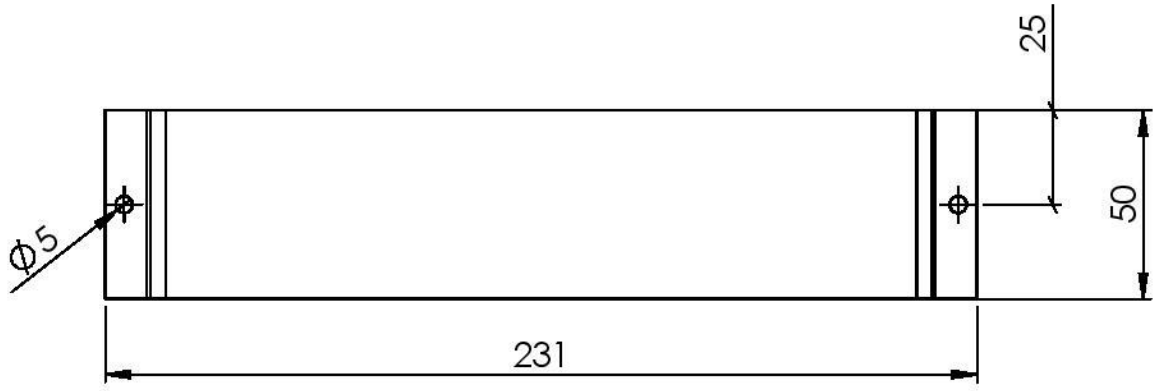
	FECHA	NOMBRE	FIRMA
Proyectado	15/06/19	Fernando Peña	Fernando
Dibujado	15/06/19	Fernando Peña	Fernando
Comprobado	15/06/19	Fernando Peña	Fernando

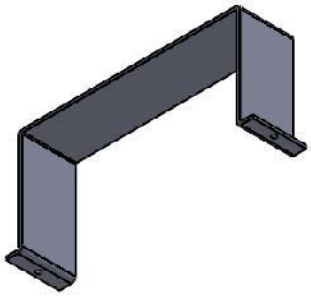
ESCALA:1:5

Montaje baterías
Vista explosionada

PLANO Nº: 1.2.0
Sustituye a:
Sustituido por:



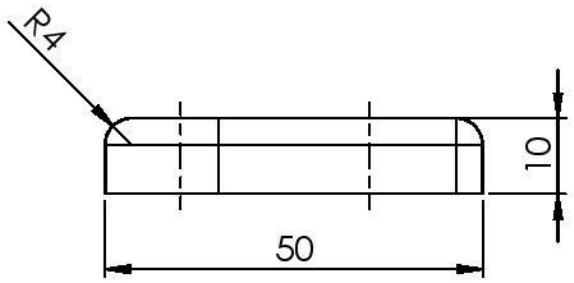


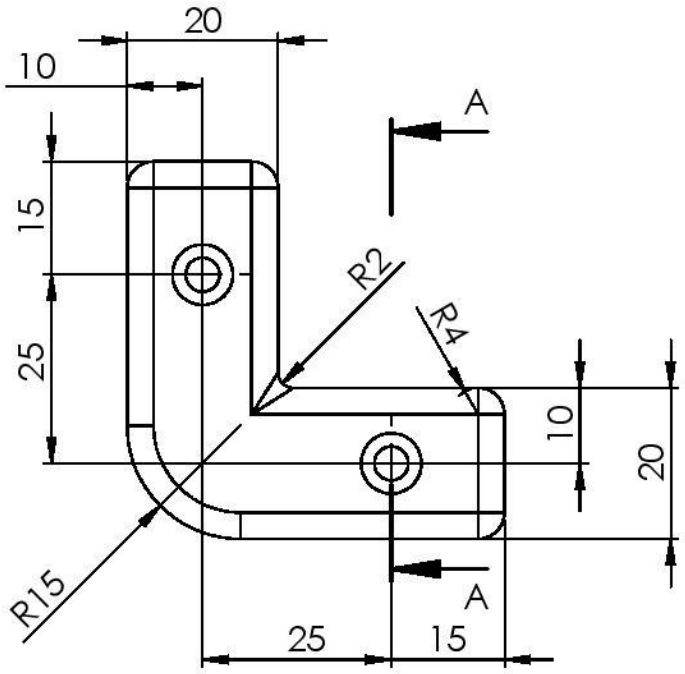


MÁSTER EN INGENIERÍA MECATRÓNICA
-TRABAJO FIN DE MÁSTER-

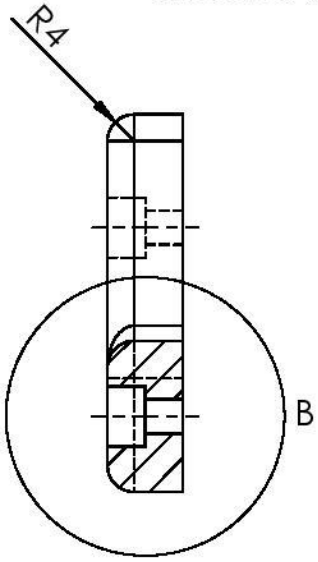
	FECHA	NOMBRE	FIRMA
Proyectado	15/06/19	Fernando Peña	Fernando
Dibujado	15/06/19	Fernando Peña	Fernando
Comprobado	15/06/19	Fernando Peña	Fernando

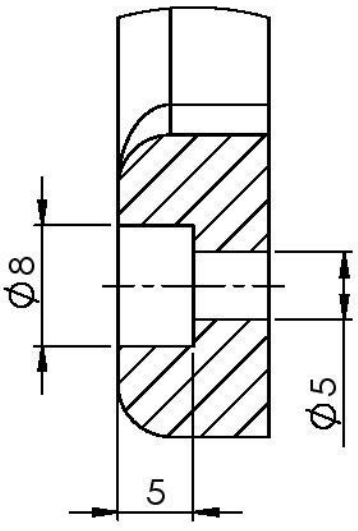
ESCALA:1:5	Soporte vertical batería	PLANO Nº: 1.2.1
		Sustituye a:
		Sustituido por:





SECCIÓN A-A



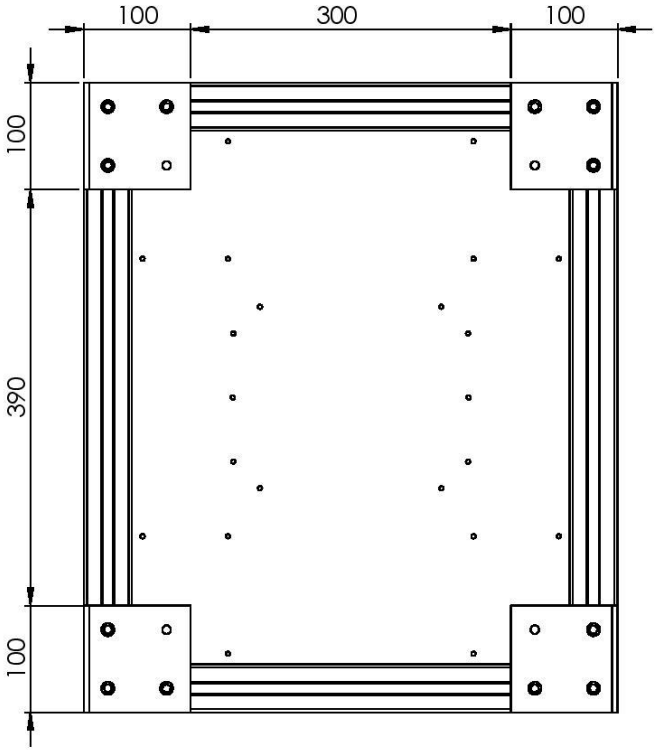


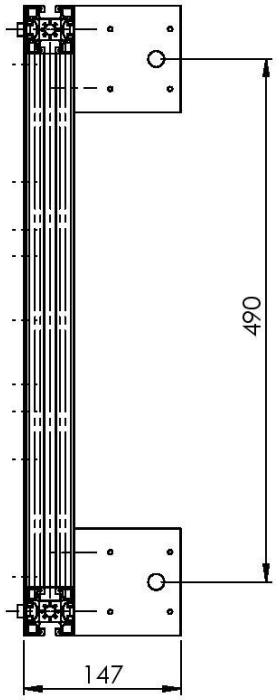
DETALLE B
ESCALA 2 : 1

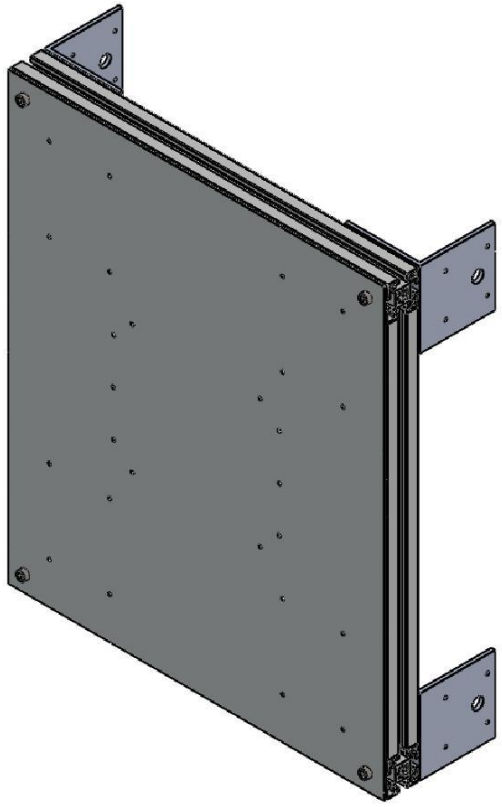
MÁSTER EN INGENIERÍA MECATRÓNICA
-TRABAJO FIN DE MÁSTER-

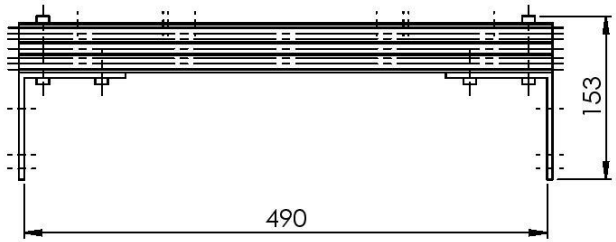
	FECHA	NOMBRE	FIRMA
Proyectado	15/06/19	Fernando Peña	Fernando
Dibujado	15/06/19	Fernando Peña	Fernando
Comprobado	15/06/19	Fernando Peña	Fernando

ESCALA:1:1	Soporte horizontal batería	PLANO Nº: 1.2.2
		Sustituye a:
		Sustituido por:









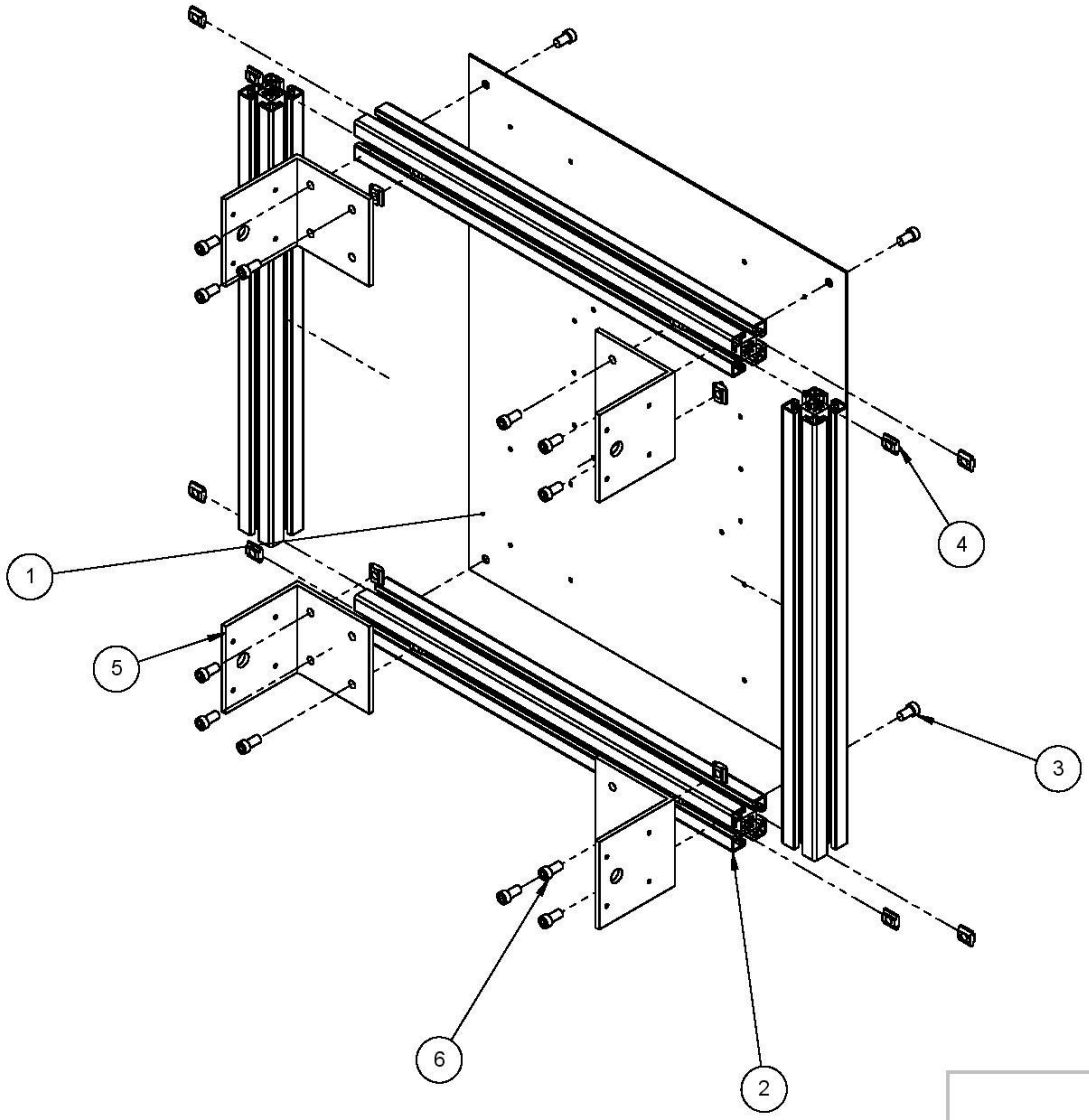
MÁSTER EN INGENIERÍA MECATRÓNICA
-TRABAJO FIN DE MÁSTER-

	FECHA	NOMBRE	FIRMA
Proyectado	15/06/19	Fernando Peña	Fernando
Dibujado	15/06/19	Fernando Peña	Fernando
Comprobado	15/06/19	Fernando Peña	Fernando

ESCALA:1:5

Montaje bastidor
Dimensiones principales

PLANO Nº: 1.3.0
Sustituye a:
Sustituido por:



N.º DE ELEMENTO	N.º DE PIEZA	CANTIDAD
1	Chapa 590x500	1
2	Perfil aluminio 45x45x500	4
3	M8x14	4
4	Tuerca T M8	16
5	Soporte motores perfil	4
6	M8x16	12

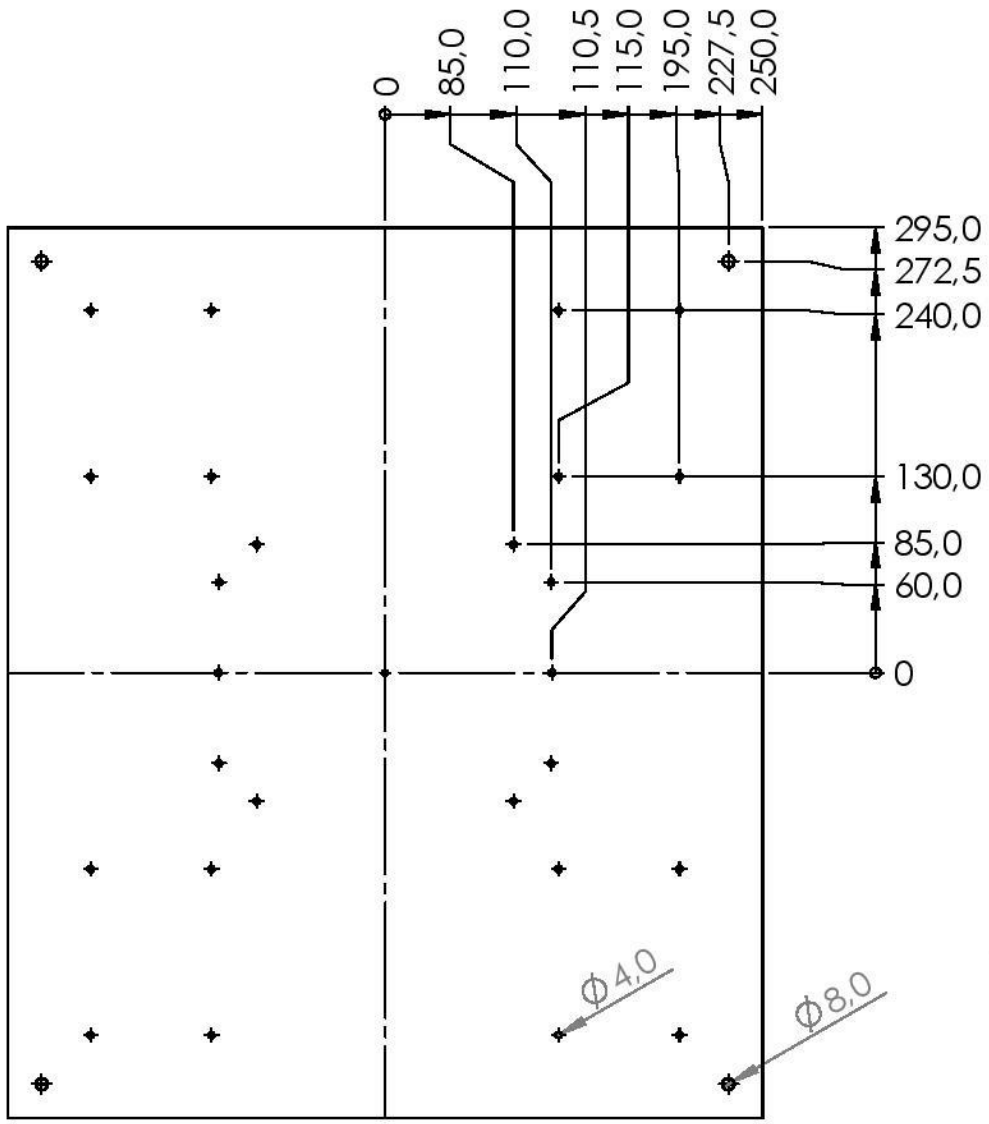
MÁSTER EN INGENIERÍA MECATRÓNICA
-TRABAJO FIN DE MÁSTER-

	FECHA	NOMBRE	FIRMA
Proyectado	15/06/19	Fernando Peña	Fernando
Dibujado	15/06/19	Fernando Peña	Fernando
Comprobado	15/06/19	Fernando Peña	Fernando

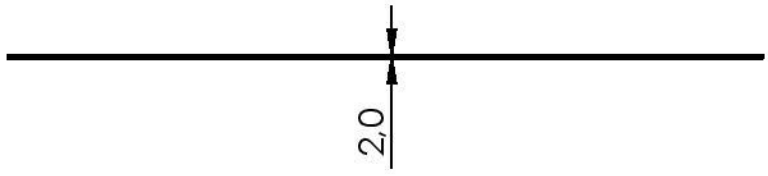
ESCALA:1:5

Montaje bastidor
Vista explosionada

PLANO Nº: 1.3.1
Sustituye a:
Sustituido por:



Nota: Los cuatro cuadrantes tienen taladros simétricos respecto de los ejes de simetría centrales



MÁSTER EN INGENIERÍA MECATRÓNICA
-TRABAJO FIN DE MÁSTER-

	FECHA	NOMBRE	FIRMA
Proyectado	15/06/19	Fernando Peña	Fernando
Dibujado	15/06/19	Fernando Peña	Fernando
Comprobado	15/06/19	Fernando Peña	Fernando

ESCALA:1:5	Chapa bastidor	PLANO Nº: 1.3.2
		Sustituye a:
		Sustituido por: