UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Dpto. de Ingeniería de Sistemas y Automática

Montaje, puesta en marcha y control de trayectorias de un robot móvil para entornos agrícolas

Trabajo Fin de Máster

Máster Universitario en Automática e Informática Industrial

AUTOR/A: Raigal Carbonell, Hipòlit

Tutor/a: Salt Llobregat, Julián José

CURSO ACADÉMICO: 2023/2024

TRABAJO DE FIN DE MÁSTER

Máster Universitario en Automática e Informática Industrial

MONTAJE, PUESTA EN MARCHA Y CONTROL DE TRAYECTORIAS DE UN ROBOT MÓVIL PARA ENTORNOS AGRÍCOLAS

AUTOR: Hipòlit Raigal Carbonell

TUTOR: Julián José Salt Llobregat

Curso 2023/2024

Resumen

En el presente trabajo se documenta el montaje, control y puesta en marcha de un robot móvil para uso en entornos agrícolas.

El robot consiste en un modelo a escala 1:5 del famoso vehículo *Curiosity*, el *rover* encargado de la misión *Mars Science Laboratory* (abreviada MSL). En este caso, en lugar de explorar Marte y recoger muestras de suelo para su posterior análisis, el objetivo del proyecto es proporcionar un vehículo capaz de recorrer una trayectoria predeterminada en un terreno accidentado y con obstáculos presentes como lo es un entorno agrícola, y así servir de base móvil para futuros proyectos.

Para ello se ha construido un prototipo desde cero con componentes económicos y ligeros, mientras que algunas piezas, como las uniones entre barras y elementos, se han impreso en 3D para facilitar el montaje. Concretamente, el robot cuenta con seis ruedas impulsadas mediante motores DC, cuatro de ellas directrices gracias a servos de posicionamiento. Todo ello está integrado y unido al cuerpo principal mediante un mecanismo conocido como *rocker-bogie*, que proporciona la estabilidad y suspensión al conjunto. El control para el seguimiento de trayectorias se ha realizado mediante una placa *Arduino Due*, que se encarga tanto del control de todos los componentes como del cálculo de la ruta a seguir.

Finalmente, se han realizado una serie de pruebas en un entorno agrícola real, con el objetivo de comprobar realmente la funcionalidad del prototipo frente a un terreno accidentado y con multitud de obstáculos presentes.

Resum

En el present treball es documenta el muntatge, control i posada en marxa d'un robot mòbil per a ús en entorns agrícoles.

El robot consisteix en un model a escala 1:5 del famós vehicle *Curiosity*, el *rover* encarregat de la missió *Mars Science Laboratory* (abreujada MSL). En aquest cas, en lloc d'explorar Mart i arreplegar mostres de sòl per a la seua posterior anàlisi, l'objectiu del projecte és proporcionar un vehicle capaç de recórrer una trajectòria predeterminada en un terreny accidentat i amb obstacles presents com ho és un entorn agrícola, i així servir de base mòbil per a futurs projectes.

Per a això s'ha construït un prototip des de zero amb components econòmics i lleugers, mentre que algunes peces, com les unions entre barres i elements, s'han imprés en 3D per a facilitar el muntatge. Concretament, el robot compta amb sis rodes impulsades mitjançant motors DC, quatre d'elles directrius gràcies a servos de posicionament. Tot això està integrat i unit al cos principal mitjançant un mecanisme conegut com *rocker-bogie*, que proporciona l'estabilitat i suspensió al conjunt. El control per al seguiment de trajectòries s'ha realitzat mitjançant una placa *Arduino Due*, que s'encarrega tant del control de tots els components com del càlcul de la ruta a seguir.

Finalment, s'han realitzat una sèrie de proves en un entorn agrícola real, amb l'objectiu de comprovar realment la funcionalitat del prototip enfront d'un terreny accidentat i amb multitud d'obstacles presents.

Abstract

This work documents the assembly, control, and commissioning of a mobile robot designed for use in agricultural environments.

The robot consists in a 1:5 scale model of the famous Curiosity rover, also known as the Mars Science Laboratory (MSL) rover. In this case, instead of exploring Mars and collecting soil samples for further analysis, the goal of this project is to create a vehicle capable of traveling on a predetermined path through rough terrain and obstacles typical of an agricultural environment, also serving as a mobile base for future projects.

To achieve this, a prototype has been built from scratch using economical and lightweight components while some parts, such as the joints between bars and elements, have been 3D-printed to facilitate assembly. Specifically, the robot has six wheels driven by DC motors, with four steering wheels guided by positioning servos. These are integrated into the main body through a mechanism known as "rocker-bogie," which provides stability and suspension to the whole. Control of the path tracking is managed by an Arduino Due, which handles both the control of all components and the calculation of the route to follow.

Finally, a series of tests were conducted in a real agricultural environment to evaluate the functionality of the prototype against rough terrain and numerous obstacles.

Índice general

Índice de figuras

Índice de cuadros

Parte I

Memoria

1 Objeto

En este proyecto se muestra el proceso completo de selección de componentes, montaje y puesta en marcha de un robot móvil para entornos agrícolas, consistente en un modelo a escala 1:5 del vehículo espacial *Curiosity*. Además, se realizará un control de trayectorias para dicho robot, con el objetivo de seguir una ruta predeterminada a través de un terreno accidentado, así como una modalidad adicional de control remoto desde un dispositivo móvil mediante Bluetooth.

1.1. Alcance

El trabajo consistirá en los siguientes aspectos:

- Contextualización del proyecto: estado del uso de nuevas tecnologías en el sector agrario, uso de robots móviles en entornos rurales y descripción general de los vehículos tipo *rover* y su cometido.
- Selección de componentes y montaje del sistema móvil.
- Programación e implementación del control de trayectorias y el control remoto.
- Pruebas y resultados de su puesta en marcha en un entorno agrícola real.

Por otro lado, quedan fuera del alcance del proyecto los siguientes aspectos:

- Diseño de las piezas en 3D y cálculo mecánico del robot.

2 Antecedentes

El presente trabajo surge de la idea de desarrollar un robot móvil capaz de desenvolverse en terrenos poco favorables y realizar tareas dentro de un entorno agrícola, como por ejemplo la inspección de cultivos o la eliminación de malas hierbas. Sin embargo, la realización de un proyecto con dichas funcionalidades desde cero requiere de mucho más tiempo y recursos de los que se disponen, por lo que la solución adoptada es la de desarrollar una plataforma robótica móvil basada en el astro vehículo *Curiosity* que pueda servir como base para futuros proyectos.

A continuación se desarrollarán algunos de los aspectos más importantes a la hora de entender tanto el contexto respecto al tipo de vehículo utilizado como el entorno donde se sitúa el trabajo.

2.1. La problemática del uso de nuevas tecnologías en el sector agrícola

Las últimas décadas han estado caracterizadas principalmente por un vertiginoso desarrollo tecnológico, el cual está presente en todos los sectores y aspectos de nuestras vidas. Algunos sectores, como el sector industria o el sector servicios, representan la mayor parte de este crecimiento, mientras que otros sectores parecen estar encontrando más dificultades a la hora de adoptar las nuevas tecnologías.

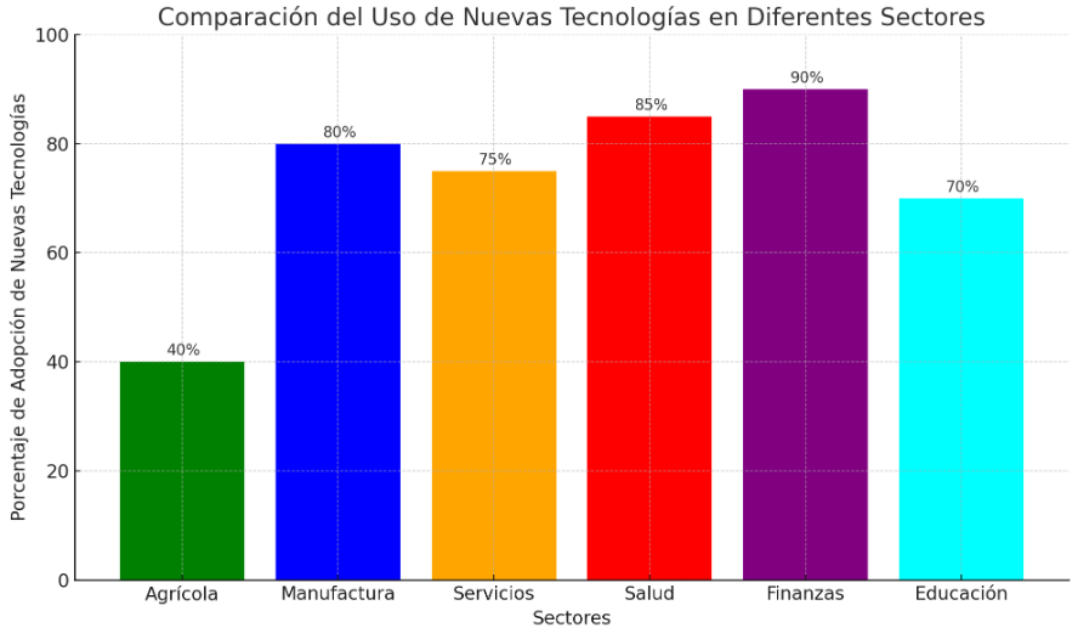


Figura 1: Gráfica comparativa del uso de nuevas tecnologías en distintos sectores

Concretamente, el sector agrario es uno de los que se muestran más reticentes a la hora de actualizarse e incorporar nuevas tecnologías, así como se muestra en la anterior figura. Algunos de los impedimentos más grandes a la hora de implementar soluciones tecnológicas en el campo son el precio, la falta de conectividad e infraestructura y la falta de capacitación en el sector. El agrario se trata de un sector más bien precario, y si a eso se le suma la creciente inestabilidad económica que atraviesa el mismo se imposibilita la implementación de nuevas tecnologías.

No obstante, esto no implica que no se estén desarrollando nuevas herramientas ni técnicas para aplicar en los campos, más bien todo lo contrario. Con el objetivo de solucionar los problemas anteriormente mencionados, se están llevando a cabo multitud de trabajos de innovación en los campos de la robótica y la automatización que buscan impulsar el sector y darle una nueva vida a nivel mundial.

(a) Uso de brazos robóticos para manipulación de cultivos

(b) Uso de drones para inspección y supervisión

Figura 2: Ejemplos de uso de nuevas tecnologías en el secor agrario

2.2. Estado del campo de la robótica móvil aplicada a la agricultura

Así como se ha mencionado en el apartado anterior, aunque el sector agrícola se encuentra más estancado que otros en lo que a digitalización se refiere, eso no implica que no se haya estado avanzando significativamente en los últimos años.

Uno de los mejores ejemplos para ilustrar esta idea, más acorde al tema que ocupa este proyecto, es la robótica móvil. Este campo de la robótica ha experimentado un gran crecimiento durante los últimos años, ofreciendo cada vez soluciones más innovadoras para realizar todo tipo de tareas como recolección, siembra o deshierbe.

Figura 3: Robot *Autonomous LaserWeeder*, de la empresa Carbon Robotics

Un buen ejemplo de ello es el robot *Autonomous LaserWeeder*, de la compañía estadounidense Carbon Robotics. El *LaserWeeder* consiste en un vehículo automatizado de gran capacidad, con diversos sensores *LiDAR* para conocer el entorno y, lo más importante, un conjunto de láseres de CO_2 que le permiten deshacerse de malas hierbas a una velocidad muy superior y de manera más eficiente de lo que lo haría un humano normalmente.

Por otro lado, la valenciana Robotnik cuenta con *BACCHUS*, un proyecto consistente en una nueva versión de su robot *RB-VOGUI-XL* con dos brazos colaborativos integrados. El tamaño más compacto del vehículo, junto con los dos brazos montados sobre el robot, permiten realizar tareas de recolección y manipulación en un entorno más cerrado como puede ser un viñedo.

Figura 4: Proyecto *BACCHUS*, de la empresa Robotnik

Estos son solamente dos ejemplos de la inmensa cantidad de robots móviles que ya existen o que se están desarrollando con fines agrícolas. Tal y como se ha mencionado al inicio del capítulo, desarrollar desde cero un robot de dichas características implica una gran inversión de tiempo y dinero, por lo que para este proyecto se ha optado por un modelo a escala 1:5 del vehículo *Curiosity* que pueda desenvolverse por terrenos poco favorables y así servir como prototipo para futuros proyectos.

2.3. El *rover Curiosity*, misión y características principales

El vehículo *Curiosity*, también conocido como el *Mars Science Laboratory* (MSL), es un *rover* de la NASA que aterrizó en Marte el 6 de agosto de 2012. Su misión principal es explorar el cráter Gale en Marte para investigar si el planeta alguna vez tuvo condiciones ambientales favorables para la vida microbiana y para estudiar el clima y la geología del planeta.

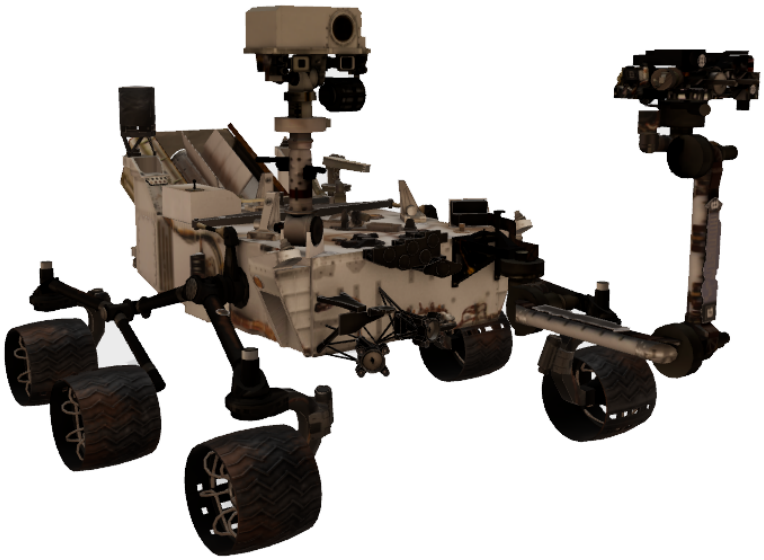


Figura 5: Modelo 3D del *rover Curiosity*

Algunas de sus características más relevantes son las siguientes:

- Dimensiones de aproximadamente 3 m de largo, 2.7 m de ancho y 2.2 m de alto, con un peso aproximado de 900 kg.
- Seis ruedas de aluminio con un diámetro de 50 cm cada una, con una velocidad media aproximada de 30 metros por hora.
- Sistema de suspensión *Rocker-Bogie*, que aporta capacidad para moverse sobre terrenos accidentados manteniendo la estabilidad.
- Multitud de sensores e instrumentos científicos con diversas funcionalidades, desde toma de imágenes y vídeos hasta equipo para análisis de muestras rocosas.

En definitiva, un vehículo como el *Curiosity* es el idóneo para realizar trabajos en entornos poco favorables, y por ello se ha tomado como referencia este proyecto tal y como se desarrollará en apartados posteriores.

2.4. Programas informáticos utilizados

2.4.1. Diseño de mapas y esquemas eléctricos con AutoCAD

AutoCAD es un *software* de diseño asistido por computador utilizado mayoritariamente para dibujo 2D y modelaje 3D, perteneciente a la empresa Autodesk. Se trata de un programa reconocido a nivel internacional por sus amplias capacidades de edición y dibujo, y es una de las principales herramientas para ingenieros, arquitectos y diseñadores. En la industria se utiliza mayoritariamente para dibujar planos de piezas, máquinas o incluso plantas enteras.

Figura 6: Logotipo del programa AutoCAD

En el proyecto se ha utilizado AutoCAD tanto para la creación del mapa del terreno donde se desenvolverá el vehículo como para la obtención de trayectorias dentro del mismo. Para ello, se ha creado un *script* que permite obtener una serie de coordenadas X e Y de los n puntos que forman una línea, devolviendo el resultado en un archivo de texto. El código descrito se encuentra en el apartado de Anexos. Por otro lado, también se ha utilizado el programa para el diseño del esquema eléctrico del robot, situado en el apartado correspondiente a los planos.

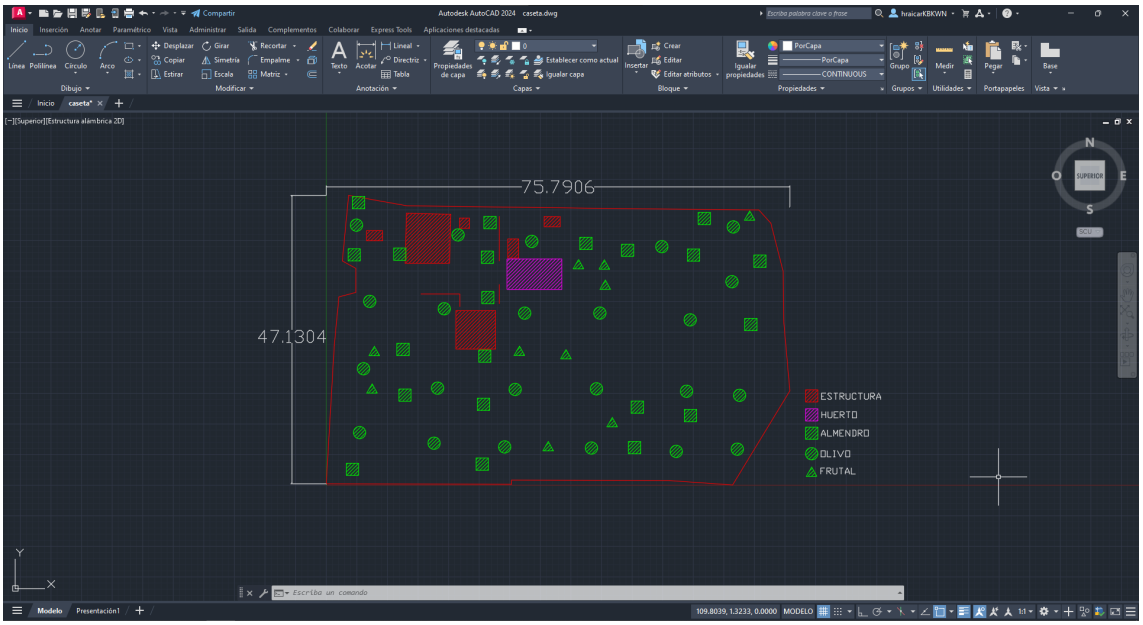


Figura 7: Entorno del programa AutoCAD

2.4.2. Programación del robot con Arduino IDE

Arduino IDE es el editor que ofrece la propia empresa Arduino para trabajar con sus productos. Se trata de un entorno de programación simple y rápido, pero que a su vez presenta múltiples funcionalidades como auto completado, navegación por el código y un *debugger* propio, todo ello completamente integrado en su ecosistema.

Figura 8: Logotipo del programa Arduino IDE

Dado que el control del robot desarrollado se realiza mediante una placa *Arduino Due*, todo el código relacionado tanto con el movimiento del vehículo como con el control de trayectorias se ha llevado a cabo dentro de este entorno de programación. Para ello, primero se han desarrollado varios códigos con el objetivo de probar los componentes del robot individualmente. Finalmente, una vez comprobados todos los elementos, se ha implementado el propio código para el seguimiento de trayectorias, así como un código para el control remoto del vehículo mediante Bluetooth.

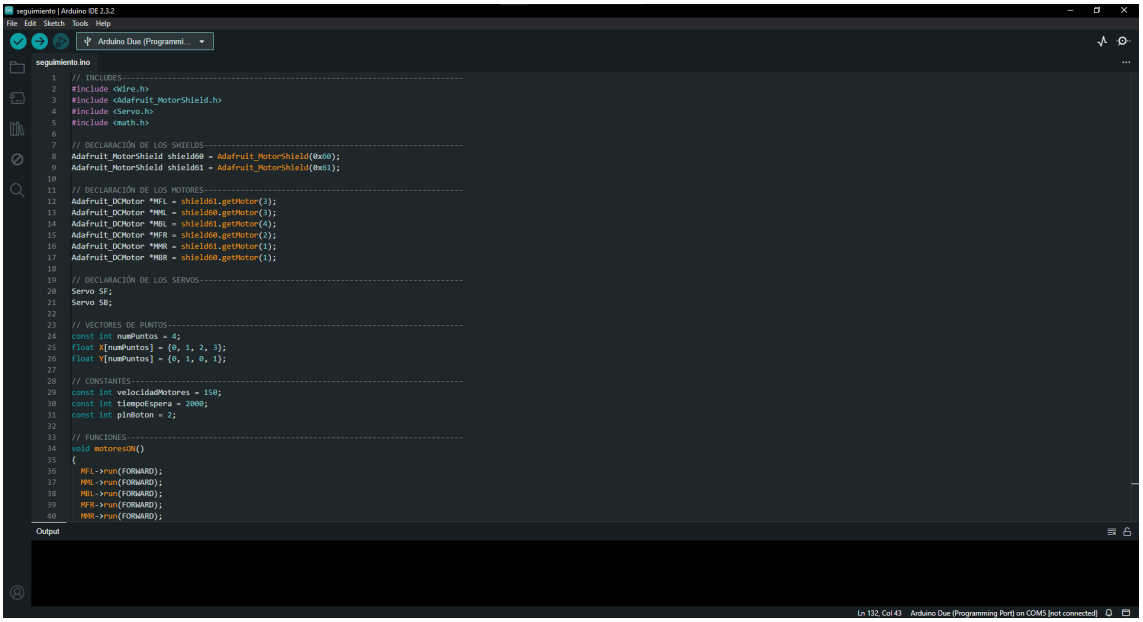


Figura 9: Entorno del programa Arduino IDE

3 Factores a considerar: necesidades, limitaciones y condicionamientos

En el presente apartado se tratarán los aspectos relacionados con las especificaciones que debe cumplir el robot diseñado. Se analizarán las especificaciones técnicas y de funcionamiento, además de las diversas limitaciones con las que se ha trabajado.

3.1. Especificaciones técnicas y de funcionamiento

Tal y como se ha mencionado anteriormente, este trabajo surge de la idea de desarrollar desde cero una plataforma robótica móvil capaz de desenvolverse en un entorno agrícola para así servir de base en futuros proyectos.

Partiendo de esta idea, las especificaciones que se han tenido en cuenta a lo largo de la realización del trabajo son:

- Diseño que permita desplazarse por terrenos accidentados, así como para sortear posibles obstáculos desviándose de la trayectoria original lo menos posible.
- Tamaño moderado (entre 500 y 600 mm), lo suficientemente grande como para adaptarse con facilidad al entorno pero que no presente problemas a la hora de desplazarse entre cultivos estrechos.
- Autonomía suficiente para la realización de varias trayectorias completas sin necesidad de recarga.
- Control de trayectorias que permita al robot seguir una ruta previamente programada, así como control remoto mediante Bluetooth.

3.2. Limitaciones y condicionamientos

No obstante, un proyecto de este tipo también presenta una serie de limitaciones y condicionamientos que hay que tener en cuenta, ya puedan ser de tiempo, físicas o económicas.

La más importante es que desarrollar un robot de estas características desde cero es un proceso que requiere grandes cantidades de tiempo y dinero, por lo que se ha tratado de limitar las funcionalidades para que pueda cumplir con su propósito empleando el menor número de recursos posibles. Además, el hecho de que se trate de un prototipo y no de un producto final también condiciona los resultados obtenidos, por lo que en todo momento se tendrá en cuenta a la hora de analizar los datos finales.

4 Soluciones alternativas y justificación de la solución adoptada

En esta sección se realizará un breve repaso de las diferentes alternativas que se contemplaron a la hora de diseñar el robot móvil agrícola, así como la solución finalmente adoptada.

4.1. Soluciones alternativas contempladas

Algunas de las principales alternativas que se tuvieron en cuenta a la hora de diseñar el vehículo fueron las siguientes:

- **Robot móvil tipo Ackerman con cuatro ruedas *Mecanum*.** Una de las ideas que se tuvieron a la hora de diseñar el vehículo fue emplear ruedas de tipo *Mecanum* con tal de aumentar el rango de movimientos que el robot puede realizar. Utilizar este modelo de rueda permitiría al robot desplazarse lateralmente sin necesidad de realizar grandes giros, lo que sería muy conveniente a la hora de desenvolverse entre líneas de cultivos estrechas. No obstante, esta alternativa se descartó rápidamente debido a que, según la propia naturaleza de las ruedas, no podría moverse a través de terrenos accidentados ni adaptarse a los obstáculos del entorno.
- **Robot móvil tipo Ackerman con cuatro ruedas convencionales.** Mismo concepto que en el caso anterior, solo que esta vez se cambian las ruedas *Mecanum* por ruedas convencionales para aportar adaptabilidad al terreno. Sin embargo, dado las especificaciones del robot se optó por descartar también esta idea, debido a que el peso completo del robot recaería sobre cuatro ruedas sencillas y la estructura se volvería demasiado frágil.

4.2. Solución finalmente adoptada

Debido a que el motivo común de descarte de todas las soluciones anteriores era la escasa adaptación a terrenos poco favorables, se ha optado por centrar la solución final en ese aspecto en concreto.

Para ello se ha adoptado la configuración de un *rover*, en este caso el *Curiosity*, como modelo de referencia para el diseño del robot. Este tipo de vehículos cuenta con seis ruedas motrices, cuatro de ellas también directrices, distribuidas en dos mecanismos independientes conocidos como *Rocker-Bogie*. Este mecanismo otorga al robot una mejor adaptación al terreno, además de mayor capacidad para sortear obstáculos sin perder la estabilidad general del vehículo.

5 Descripción detallada de la solución adoptada

Una vez las alternativas no viables han sido descartadas, se realizará una descripción más detallada de la solución finalmente adoptada, así como del método de control elegido para el seguimiento de trayectorias.

Para ello, se va a dividir el contenido de esta sección en tres apartados; mecánica, electrónica y control, tal como se muestra en el siguiente organigrama:

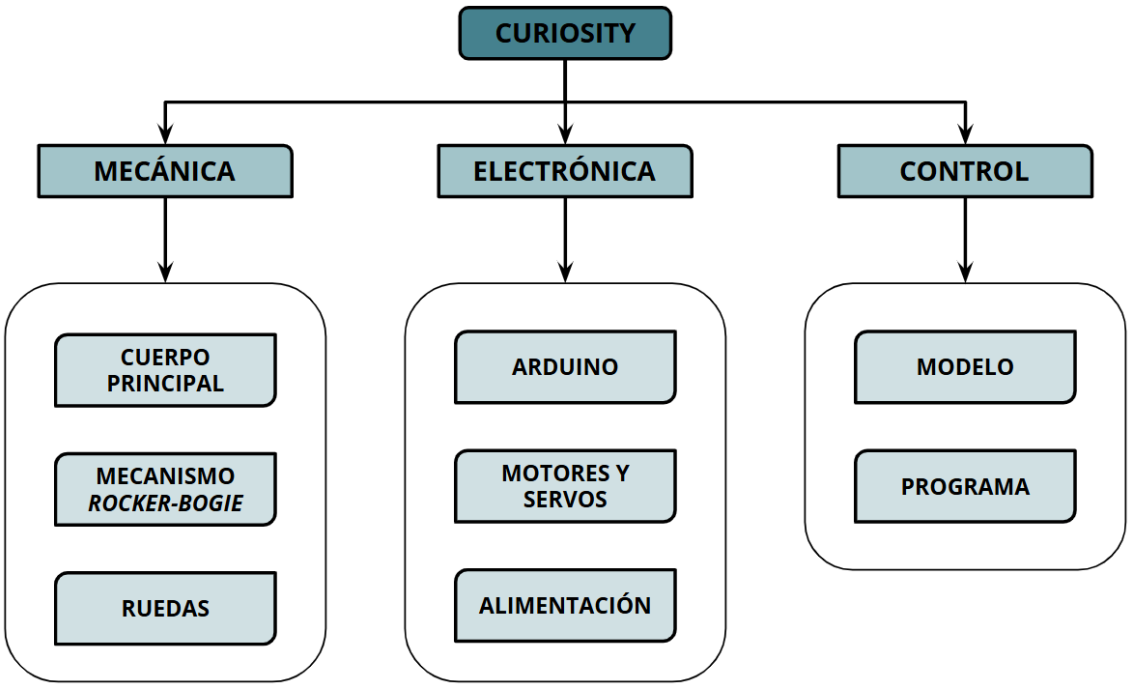


Figura 10: Organigrama del proyecto

Esta clasificación será utilizada a partir de este punto durante todo el trabajo, ya que es la que mejor refleja la estructura del proyecto y permite diferenciar de manera clara los diversos subsistemas.

A continuación se profundizará en cada uno de los subsistemas y subconjuntos, con el objetivo de ofrecer una visión clara sobre la función de cada elemento dentro del conjunto total del proyecto.

5.1. Mecánica

La mecánica, correspondiente a la parte estructural del robot, está formada por el cuerpo principal, el mecanismo de suspensión *Rocker-Bogie* y las ruedas.

Figura 11: Esqueleto que conforma el cuerpo del robot

5.1.1. Cuerpo principal

El cuerpo principal del robot consiste en una sencilla estructura formada por planchas de metacrilato unidas entre sí.

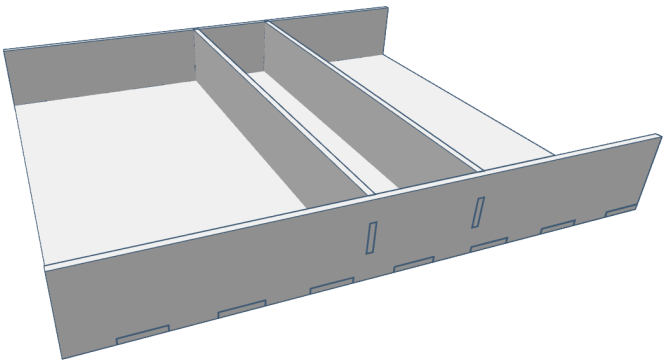


Figura 12: Plataforma formada por planchas de metacrilato

Esta plataforma sirve tanto de unión mecánica entre ambos balancines como superficie de montaje para los componentes electrónicos que forman el sistema, por lo que debe estar estable en todo momento. Para ello, cuenta con una barra móvil que conecta ambos lados del mecanismo *Rocker-Bogie*, asegurando que se encuentren compensados en todo momento. Además, ambos balancines también se encuentran unidos mediante una varilla metálica, otorgando rigidez adicional al conjunto.

5.1.2. Mecanismo *Rocker-Bogie*

El mecanismo conocido como *Rocker-Bogie* es una de las piezas fundamentales del proyecto. Consiste en un sistema de suspensión y tracción diseñado para vehículos de exploración espacial, debido principalmente a la estabilidad, tracción y robustez que aporta a este tipo de vehículos. En este proyecto, se ha decidido utilizar un sistema de estas características debido a las similitudes entre un entorno agrícola y la superficie de un planeta como Marte.

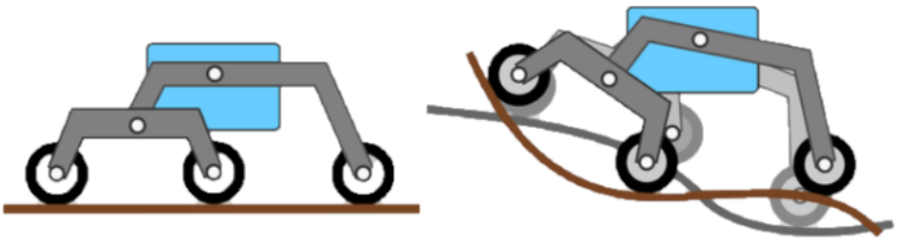


Figura 13: Mecanismo *Rocker-Bogie* sobre distintos terrenos

La estructura básica está formada por los siguientes elementos:

- ***Rocker:*** Brazo oscilante que va unido al chasis del vehículo mediante un mecanismo diferencial, que permite que ambos brazos compensen el movimiento el uno del otro. Cuenta con una rueda motriz a un extremo de la barra.
- ***Bogie:*** Segundo brazo del conjunto; va unido al otro extremo del *Rocker*, y cuenta con dos ruedas motrices que permiten adaptarse y ejercer tracción según las inclinaciones del terreno.

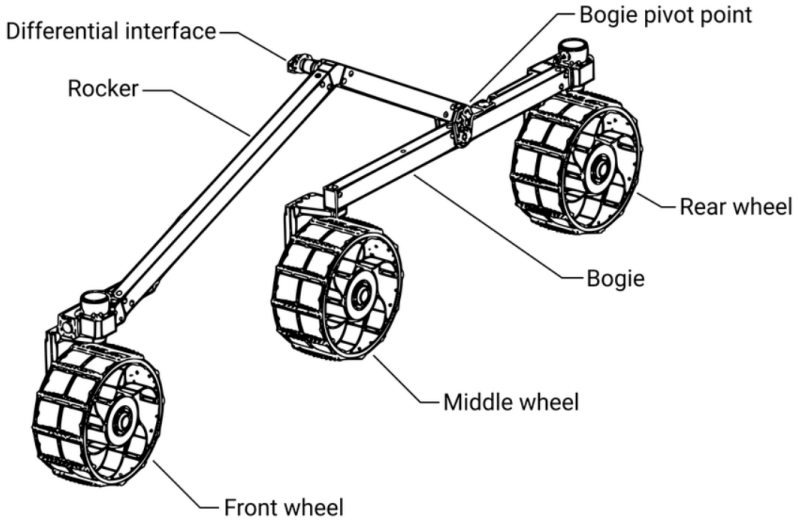


Figura 14: Partes del mecanismo *Rocker-Bogie*

Tal y como se ha mencionado, el uso del mecanismo *Rocker-Bogie* aporta muchas ventajas a la hora de desplazarse a través de terrenos accidentados o poco favorables, por lo que es la elección idónea para este proyecto.

Para replicar este mecanismo en el robot desarrollado, se han impreso una serie de piezas en 3D que harán de uniones entre las barras, mientras que para las barras se han utilizado perfiles de aluminio de distintas medidas. A continuación se muestra el resultado de la implementación del mecanismo con y sin ruedas:

(a) Mecanismo sin ruedas (b) Mecanismo con ruedas

Figura 15: Implementación del mecanismo *Rocker-Bogie*

5.1.3. Ruedas

Por último, para las ruedas del vehículo se ha tratado de replicar el modelo original de las ruedas del vehículo *Curiosity*. Al igual que las uniones entre barras del mecanismo *Rocker-Bogie*, se han impreso en 3D a una escala reducida al modelo original y con ligeras variaciones, pero manteniendo el diseño general.

(a) Rueda original (b) Rueda replicada

Figura 16: Comparativa entre las ruedas originales del *Curiosity* y la réplica a escala

El objetivo es contar con unas ruedas tipo todoterreno capaces de desplazarse por un entorno agrícola. Por ello el diseño de la rueda presenta una ligera curvatura en el centro, además de una serie de extrusiones a modo de palas para adaptarse mejor al terreno y a posibles obstáculos.

5.2. Electrónica

La electrónica del robot está formada por el hardware de control, que ejecuta los algoritmos necesarios para la ejecución de tareas, y el sistema de potencia, encargado de mover y direccionar el vehículo. Todo ello funciona gracias a un sistema de alimentación acoplado que suministra energía al conjunto.

5.2.1. Arduino

Para este proyecto, el hardware de control consiste principalmente en una placa Arduino Due acompañada de dos Adafruit Motor Shield V2 en cascada, además de un módulo Bluetooth para el control remoto.

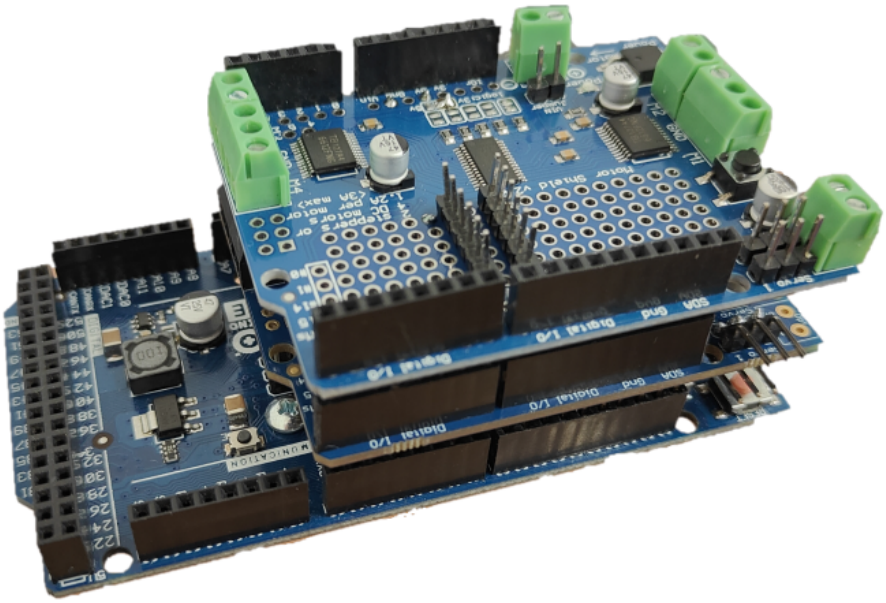


Figura 17: Conjunto de placas Arduino-Adafruit

Se ha considerado el uso de la placa Arduino Due por encima de otras de la familia debido a la gran cantidad de pines entrada/salida de los que dispone, tanto digitales como analógicos, además de su mayor velocidad de procesamiento y precisión de control. Sin embargo, se trata de una placa que opera lógicamente a 3.3 V, por lo que habrá que tenerlo en cuenta para más adelante.

Por otro lado, la placa Adafruit Motor Shield V2 se trata de una *shield* compatible con Arduino que facilita el control mediante I2C de motores DC y servomotores. Cada placa debe alimentarse independientemente, y puede controlar hasta 4 motores DC y 2 servomotores. Por ello, para controlar los 6 motores DC y los 4 servomotores será necesario contar con 2 *shield* apiladas sobre el Arduino. Por último, para trabajar junto con el Arduino Due las placas también deberán configurarse para operar a 3.3 V.

5.2.2. Motores y servos

Los motores y servomotores son los encargados de llevar a cabo el movimiento del vehículo. Así como se ha comentado durante los apartados anteriores, el robot cuenta con un total de seis ruedas motrices, cuatro de ellas también directrices. Esto se traduce en que todas las ruedas cuentan con un motor DC independiente, mientras que cuatro de ellas, situadas en las cuatro esquinas del vehículo, también cuentan con un servomotor que se encargará de dar la dirección al robot.

Figura 18: Conjunto de rueda con motor DC y servomotor

La alimentación y control de motores y servos es llevada a cabo desde las placas Adafruit Motor Shield V2, por lo que todas las conexiones se realizan en un mismo sitio. Por último, los conjuntos rueda-motor-servomotor consisten en una unión mecánica que funciona gracias a rodamientos, permitiendo al servomotor ejercer un movimiento de giro suave a la hora de establecer la dirección o ángulo del conjunto.

5.2.3. Alimentación

Finalmente, el sistema de alimentación es el encargado de proporcionar la energía necesaria para el funcionamiento del conjunto entero. Está formado por una batería Li-Po, con capacidad suficiente para unas horas de funcionamiento, y un conversor DC-DC que se encarga de disminuir el voltaje con el que se alimentarán los servomotores para evitar fallos por sobretensión.

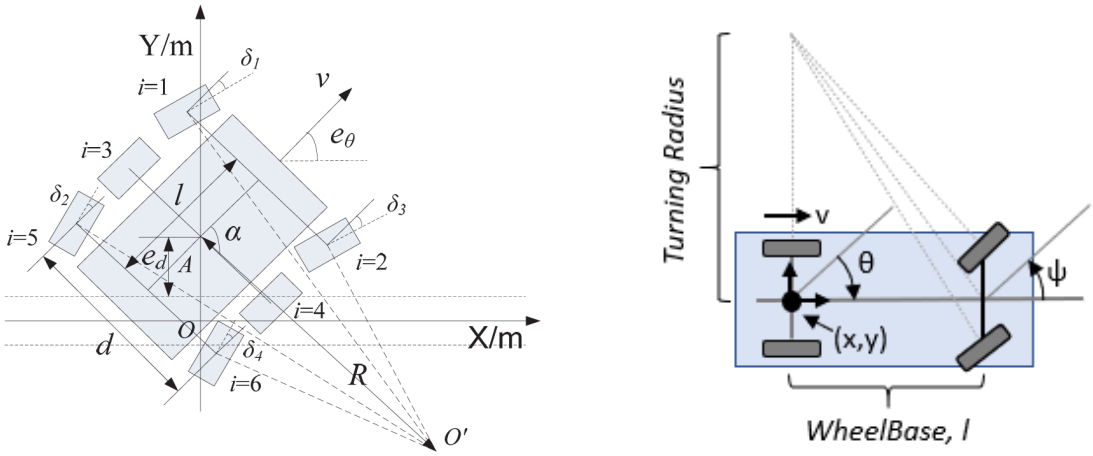
5.3. Control

Por último, el subsistema de control es el encargado de ejecutar el algoritmo de seguimiento de trayectorias del vehículo y dar las órdenes a los diferentes componentes del sistema. A continuación se estudiará el modelo utilizado para desarrollar el algoritmo, así como se comentará brevemente el programa que llevará a cabo el control.

5.3.1. Modelo

El modelo del robot desarrollado está basado en aquel diseñado por la NASA para su vehículo de exploración espacial, el cuál se ha replicado para este proyecto.

Dicho modelo es muy similar a un típico modelo Ackermann, presente en la mayoría de automóviles y vehículos convencionales. Solamente se diferencian en dos aspectos fundamentales; la presencia de dos motores intermedios a cada lado y el hecho de que las dos ruedas traseras sean también directrices.



(a) Modelo del vehículo *Curiosity*

(b) Modelo de vehículo Ackermann

Figura 19: Comparativa entre distintos modelos de vehículos

El control del robot se ha realizado tomando como referencia estos dos modelos, aunque se han realizado una serie de simplificaciones con el objetivo de facilitar la programación del vehículo:

- Los seis motores DC se mueven siempre a la misma velocidad constante.
- Tanto la pareja de ruedas delanteras como la de ruedas traseras siempre girarán el mismo ángulo.
- Las parejas de ruedas delanteras y traseras siempre formarán ángulos suplementarios, es decir, la suma de ambas será de 180º.

Estas simplificaciones permiten traducir el sistema a otro tipo de modelo más sencillo, el modelo de un vehículo tipo bicicleta. Este modelo consiste en un cuerpo rígido, con dos ejes situados a los extremos del mismo (el delantero móvil, y el trasero fijo) y separados por una determinada distancia.

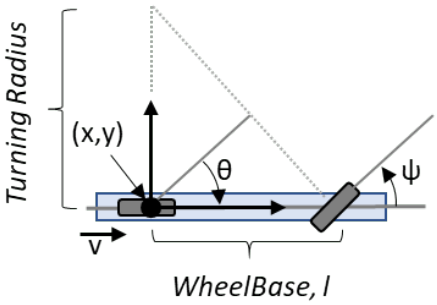


Figura 20: Modelo de vehículo bicicleta

Sin embargo, en este caso trabajaremos asumiendo que ambos ejes son móviles, lo que nos permite realizar giros más cerrados. Por tanto, se programará el control de trayectorias teniendo en cuenta las siguientes consideraciones:

- Todos los motores DC se mueven a una velocidad única y constante, arrancando al inicio de la trayectoria y deteniéndose al final de la misma.
- Los servomotores actúan en conjuntos de dos en dos; los dos delanteros forman una pareja, mientras que los dos traseros forman otra. Esto se traduce en que cada pareja girará un mismo ángulo.
- El ángulo de giro de los servomotores delanteros y traseros es suplementario. Esto facilita la programación, dado que con solamente calcular el ángulo de giro de un conjunto se obtiene el otro.

5.3.2. Programa

Tomando en cuenta todas las consideraciones anteriores, se ha desarrollado un código para Arduino que consiste en el seguimiento punto a punto de una trayectoria proporcionada. El funcionamiento general es el siguiente:

1. Se inicializan todos los componentes, situando los servomotores a 90º y asignando a los motores DC la velocidad deseada.
2. A partir de dos vectores de puntos (coordenadas X e Y), se calculan la distancia a recorrer y el tiempo en base a la velocidad.
3. Se mueven los servos a la posición necesaria y se inician los motores DC, deteniéndose una vez se haya cumplido el tiempo de desplazamiento.
4. Se repite el procedimiento según el número de puntos que tenga la trayectoria.

A continuación se muestra el diagrama de flujo correspondiente al programa desarrollado. El programa completo se encuentra en el apartado del Anexo correspondiente a los códigos.

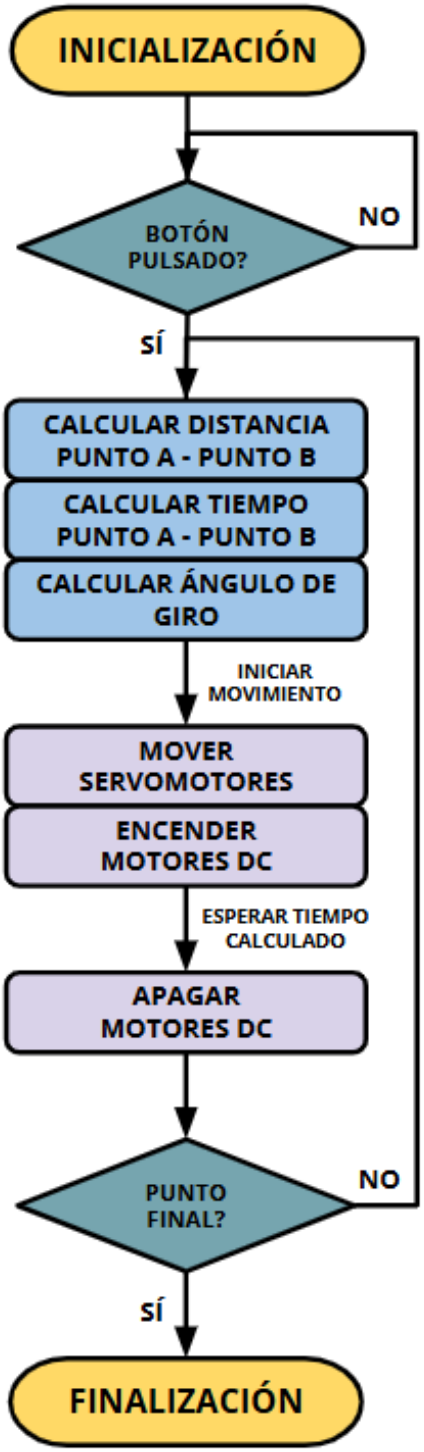


Figura 21: Diagrama de flujo del seguimiento de trayectorias

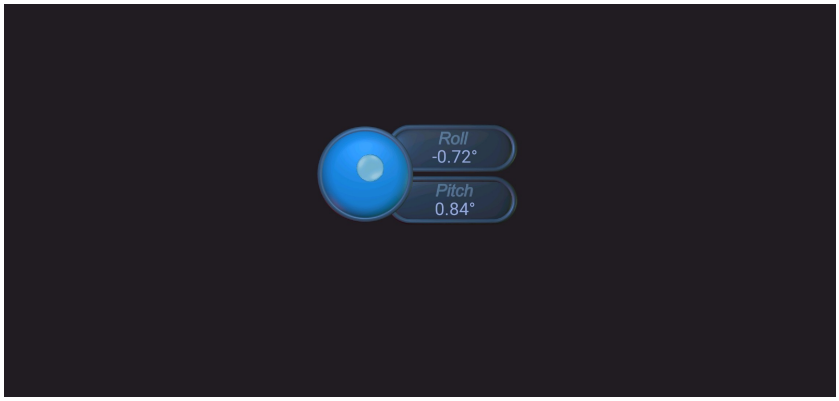
Para complementar el programa del seguimiento de trayectorias, así como ofrecer otra alternativa para el control del vehículo, se ha implementado la funcionalidad de control remoto desde un dispositivo móvil mediante Bluetooth, empleando el módulo para Arduino HC-06.

Concretamente, se han implementado dos tipos de 'control' o mando distintos:

- **Control mediante *joysticks*:** El vehículo se controla mediante dos palancas, una para la velocidad y otra para la dirección. La primera de ellas, correspondiente a la velocidad del vehículo, solamente tendrá en cuenta cambios en el eje Y, mientras que la segunda, correspondiente a la dirección, se limitará a los movimientos en el eje X.
- **Control mediante acelerómetro:** El vehículo se controla mediante el acelerómetro incorporado del dispositivo móvil. La velocidad del robot se define en base al *pitch*, mientras que la dirección se controla mediante el *roll*.

A continuación se muestran las dos pantallas o mandos implementados:

(a) Control mediante *joysticks*



(b) Control mediante acelerómetro

Figura 22: Pantallas de mando del control remoto

6 Justificación detallada de la selección y dimensionamiento de los elementos

El objetivo de este apartado es presentar, solamente en aquellos casos donde sea necesario, las consideraciones técnicas a tener en cuenta a la hora de dimensionar los elementos descritos en el apartado anterior.

6.1. Mecánica

Tanto el cálculo mecánico del robot como el diseño de las piezas 3D que forman parte del mecanismo *Rocker-Bogie* no forman parte del alcance del proyecto, por lo que en este apartado se limitará a ofrecer algunos datos técnicos que puedan ser de relevancia.

6.1.1. Cuerpo principal

Se ha elegido el metacrilato para el cuerpo principal del robot debido a que se trata de un material rígido pero ligero, dos cualidades que se buscarán para el resto de elementos del chasis del vehículo.

Concretamente, se ha elegido metacrilato de 3 *mm* de grosor, suficiente para soportar la electrónica sobre el mismo pero al mismo tiempo no suponer una gran carga de peso sobre las ruedas.

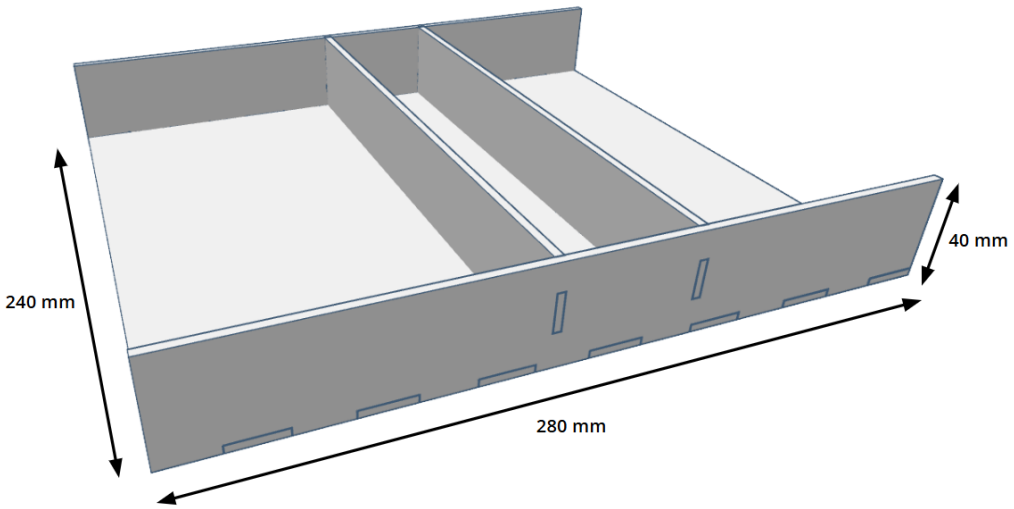


Figura 23: Medidas del cuerpo de metacrilato

6.1.2. Mecanismo *Rocker-Bogie*

Ambos mecanismos *Rocker-Bogie* están formados por los siguientes elementos:

- Elementos de unión entre barras impresos en 3D en polvo de nylon. Este material ofrece la rigidez y ligereza buscadas, además de permitir cierto grado de flexibilidad beneficioso en elementos móviles.

Figura 24: Elementos de unión impresos en 3D

- Barras de unión consistentes en perfiles de aluminio cuadrados de 10x10x1 *mm* de distintas medidas de longitud (especificadas en el pliego de condiciones). Presenta las mismas cualidades que el resto de elementos.

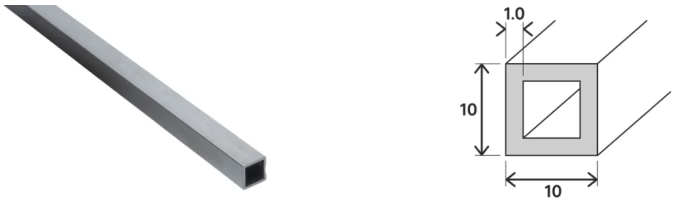


Figura 25: Perfil de aluminio 10x10 *mm*

6.1.3. Ruedas

Las ruedas han sido igualmente impresas en 3D, aunque para este elemento se ha utilizado filamento PLA en lugar del polvo de nylon empleado anteriormente. Cuentan con unas dimensiones de 70 *mm* de ancho por 85 *mm* de diámetro.

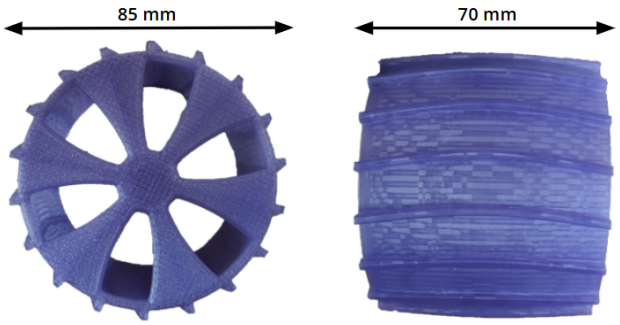


Figura 26: Medidas de las ruedas del vehículo

6.2. Electrónica

Del subsistema electrónico forman parte el conjunto del Arduino, los elementos móviles y la alimentación del vehículo. En este caso se analizarán uno a uno todos los componentes, presentando los cálculos realizados en caso de que sea necesario.

6.2.1. Arduino

El conjunto encargado del control está formado por una placa Arduino Due sobre la que se encuentran apiladas dos placas Adafruit Motor Shield V2. El Arduino se encarga del control del sistema completo, mientras que cada *shield* manejará tres motores DC y dos servomotores.

6.2.1.1. Arduino Due

Tal y como se ha explicado en el apartado anterior, se ha elegido el Arduino Due frente a otros controladores de la familia debido a la gran cantidad de pines de los que dispone, además de su mayor velocidad de procesamiento y precisión de control. Adicionalmente, la placa también presenta las siguientes características:

- Microcontrolador de 32-bits basado en el Atmel SAM3X8E ARM Cortex-M3.
- Voltaje de operación de 3.3 V, voltaje de alimentación de 7 a 12 V.
- 54 pines de entrada/salida digitales, 12 entradas analógicas.
- Comunicación por I2C, que será de utilidad para direccionar las *shields*.

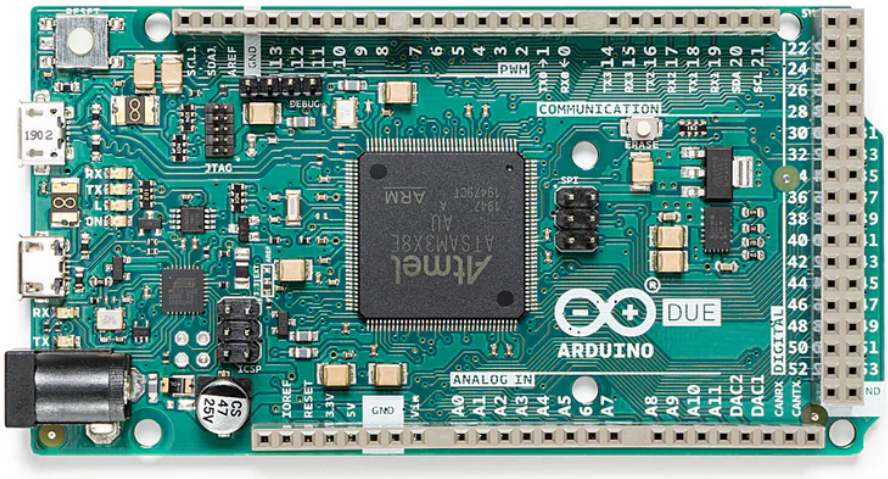


Figura 27: Arduino Due

6.2.1.2. Adafruit Motor Shield V2

Las placas Adafruit Motor Shield V2 son las encargadas de la alimentación y el control del conjunto móvil. Cada una de las placas puede manejar hasta 4 motores DC y 2 servomotores, por lo que para el proyecto se han utilizado un total de dos *shields*. Cada una de ellas presenta las siguientes características:

- Control de motores mediante el chip de puente en H TB6612.
- Proporciona 1.2 A por motor hasta un máximo de 3 A. La alimentación se realiza de manera independiente, permitiendo separar lógica de potencia.
- Comunicación por bus I2C con la placa sobre la que se apile.
- Posibilidad de configuración de lógica interna a 3.3 V.

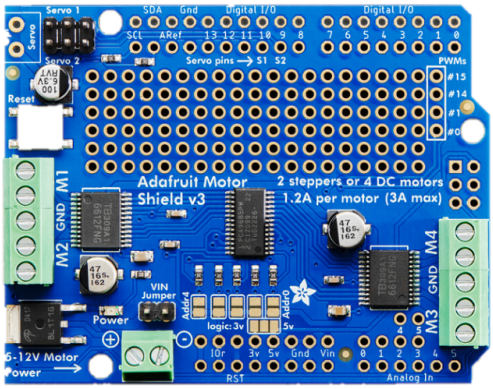


Figura 28: Adafruit Motor Shield V2

6.2.1.3. Módulo Bluetooth HC-06

El módulo Bluetooth HC-06 es el responsable de recibir las señales de mando lanzadas desde el dispositivo móvil, así como transmitirlas a la placa Arduino mediante comunicación serie.

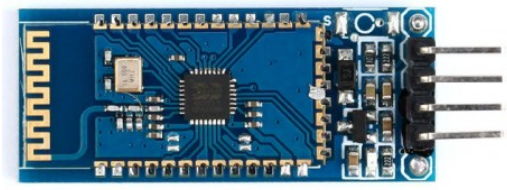


Figura 29: Módulo Bluetooth HC-06

6.2.2. Motores y servos

6.2.2.1. Motores DC

Para el proyecto se ha utilizado un conjunto de seis motores DC de la marca Pololu, concretamente el modelo *37D Metal Gearmotors*. Se trata de unos motores de capacidad mayor a los motores DC convencionales, lo que permite que el robot sea capaz de desplazarse con un mayor peso por terrenos poco favorables. Algunas de las características que presenta este modelo son las siguientes:

- Diámetro de 37 *mm* con reductor 70:1 incorporado.
- Voltaje de alimentación de 12 *V*.
- Encoder incorporado en el propio conjunto del motor.

Figura 30: Motor DC *37D Metal Gearmotors*, de la marca Pololu

Así como se ha mencionado, este modelo de motor DC cuenta con su propio encoder incorporado, lo que permite conocer el número de vueltas que ha dado el motor y, en consecuencia, la posición actual del vehículo respecto al punto inicial. La conexión tanto del motor como del encoder se realizará siguiendo las especificaciones proporcionadas por el *datasheet*:

Lead Color	Function
Red	Motor power
Black	Motor power
Green	Encoder ground
Blue	Encoder Vcc (3.5 V to 20 V)
Yellow	Encoder A output
White	Encoder B output

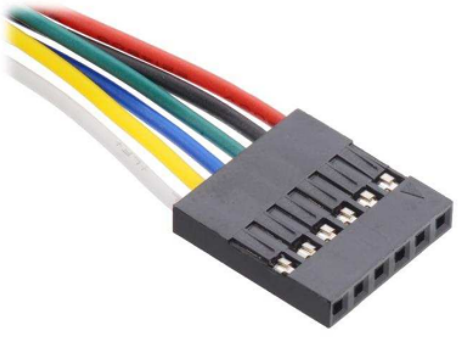


Figura 31: Esquema de conexión de los motores DC

6.2.2.2. Servomotores

El otro elemento del conjunto móvil son los servomotores, encargados de la dirección del vehículo. En este caso se han utilizado cuatro servomotores *FS5115M* de la marca FEETECH. Al igual que los motores DC, se han buscado unos servomotores capaces de mover una mayor carga que un modelo convencional. Las características que presentan son:

- Voltaje de funcionamiento de entre 4.8 y 6 V.
- Rango de ancho del pulso de 500 a 2500 μsec (0^{o} a 180^{o}).
- Desplazamiento de carga de hasta 15 kg a capacidad máxima.
- Pin de *feedback* incorporado.

Figura 32: Servomotor *FS5115M*, de la marca FEETECH

Al igual que los motores DC, los servomotores utilizados cuentan con sus propias herramientas para conocer la posición del motor. En este caso, cada servomotor cuenta con un pin de *feedback*, que permite obtener la posición del servo y utilizar el dato para determinar el ángulo de giro del robot. La conexión de los servomotores se realizará siguiendo las indicaciones a continuación:

Lead Color	Function
Brown	Servo ground
Red	Servo power
Orange	Servo signal
Green	Feedback pin

Figura 33: Esquema de conexión de los servomotores

6.2.3. Alimentación

Por último, el subsistema correspondiente a la alimentación del robot está formado por la batería y un conversor DC-DC, además del conjunto de cableado, regletas de conexión y conectores que los acompañan.

6.2.3.1. Batería Li-Po

Para la alimentación del vehículo se ha elegido una batería de polímero de litio (Li-Po) de gran capacidad, ya que el conjunto cuenta con muchos elementos y así permitirá realizar pruebas de mayor duración. Las características técnicas son las siguientes:

- Voltaje de 11.1 V, divididos en tres celdas iguales.
- Capacidad de 3500 mAh.
- Conector tipo XT60.

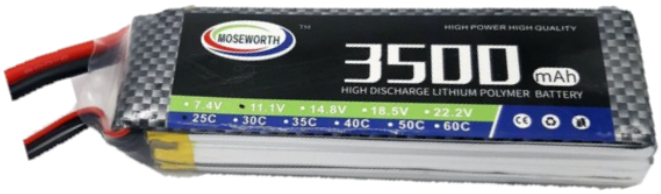


Figura 34: Batería Li-Po de 3500 mAh

6.2.3.2. Conversor DC-DC

Debido a que la tensión de alimentación de los servos es menor a la del resto del conjunto, se ha empleado un conversor DC-DC ajustable con voltímetro incorporado, fijado a una tensión de salida de unos 6 V aproximadamente.

Figura 35: Conversor DC-DC ajustable XL4015

7 Pruebas realizadas

En esta sección se explicarán las diferentes pruebas realizadas sobre el robot, añadiendo observaciones y comentarios sobre cada una de ellas.

7.1. Pruebas de motores y servomotores

Los motores DC y servomotores han sido testeados tanto individualmente como en conjunto. Las pruebas realizadas según el tipo de elemento son las siguientes:

- **Motores DC:** Para comprobar el funcionamiento de los motores DC, se ha ejecutado un programa simple que consiste en arrancar un solo motor durante un tiempo pequeño. Con esto simplemente se comprueba si el motor arranca o no, y se puede probar con distintas velocidades.

  Una vez todos los motores han sido comprobados, se ha pasado a comprobar el conjunto en sí. Para ello se han arrancado todos los motores a la vez, haciendo que el robot avance en línea recta.

- **Servomotores:** Los servomotores están conectados de dos en dos, es decir, los dos servomotores delanteros actúan como uno solo y idem con los traseros. Por ello, para probarlos simplemente basta con configurar el programa como si se estuviera trabajando con dos servomotores. La prueba consiste en mover el eje a distintas posiciones, y comprobar si funcionan como es esperado.

Estas pruebas son principalmente para comprobar que todos los componentes funcionan correctamente, por lo que no se pueden extraer apenas conclusiones.

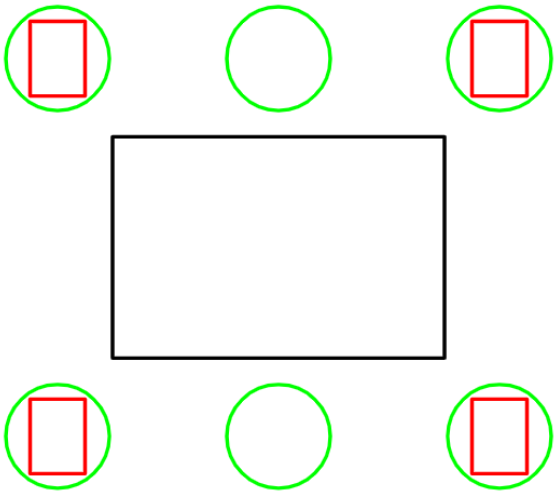


Figura 36: Disposición de motores y servomotores

7.2. Pruebas generales del conjunto

Las pruebas generales del conjunto consisten en probar un desplazamiento empleando a la vez motores DC y servomotores. Para ello, se ha desarrollado un programa más elaborado que consiste en una secuencia de movimientos que deberá realizar el robot:

1. Los servos se inicializan a 90^{o}, y se asigna a los motores una velocidad baja.
2. Los motores arrancan y avanzan en línea recta durante 3 segundos hasta detenerse para ajustar los servomotores a un nuevo valor.
3. Los servos delanteros se situan a 50^{o} y los traseros a 150^{o}.
4. Los motores vuelven a arrancar, y el conjunto avanza en círculo durante 3 segundos hasta detenerse finalmente.

Los movimientos descritos son arbitrarios y se pueden modificar según parezca, así como añadir más movimientos a la secuencia.

Durante la realización de la prueba se ha podido comprobar el funcionamiento del sistema completo, y ha sido de ayuda a la hora de detectar errores y a la vez implementar algunas mejoras. Algunos de los errores detectados son los siguientes:

- Se ha detectado una mala conexión entre las *shields* y la Arduino Due. Según cómo de separadas se encuentren entre ellas, se interrumpe o no la conexión por I2C. Esto imposibilitaba a veces el control de los motores DC.
- La placa Arduino Due no se alimenta correctamente desde el puerto barril tipo *jack* del que dispone. Para solucionarlo, se ha alimentado la placa a través de una *power bank* externa para móviles.
- Una de las ruedas desliza en lugar de estar firmemente sujeta al servomotor. Esto causa que dicha rueda no siempre se situe en la posición deseada, causando que el vehículo entero se desvíe.

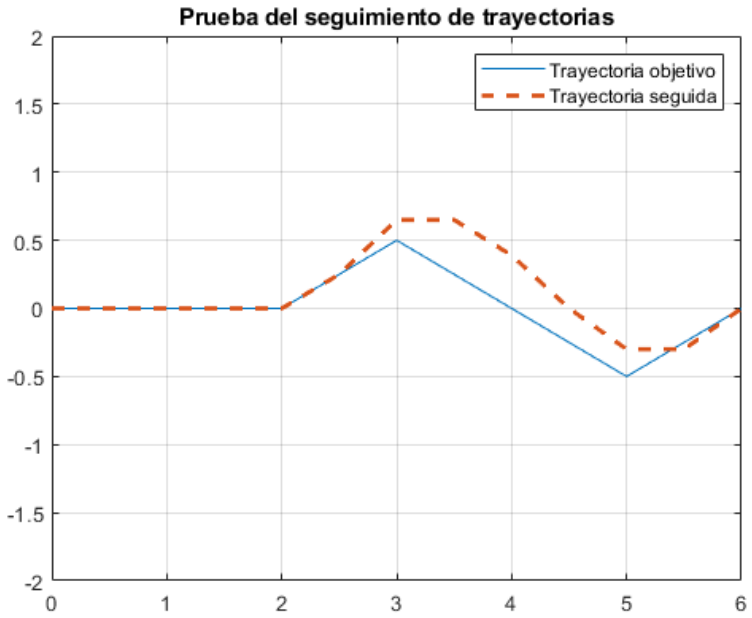
Por otro lado, la prueba también ha servido para implementar las siguientes mejoras:

- Se ha mejorado el código del seguimiento de trayectorias, añadiendo el cálculo de la velocidad lineal en base a las hojas de especificaciones de los motores. Con ello, se podrá obtener una lectura aproximada de la posición en la que se debería encontrar el robot en cada momento.
- Se han sustituido algunas de las piezas que presentaban problemas, logrando una estructura más sólida.
- Se ha añadido un pulsador de inicio, que sirve para arrancar el programa a voluntad tras haber sido cargado desde el PC. Esto permite poder desplazar el robot manualmente al punto de inicio y, una vez allí, empezar con la prueba sin depender de la ubicación del ordenador.

7.3. Pruebas del seguimiento de trayectorias

A diferencia de las pruebas realizadas anteriormente, en este caso los movimientos a seguir ya no han sido definidos uno a uno. Se ha empleado el código de seguimiento de trayectorias, que a partir de una serie de puntos que forman una trayectoria, calcula mediante iteraciones sucesivas la dirección y tiempo que deberá tomar el robot para seguir una ruta preestablecida. A continuación se muestra un ejemplo de trayectoria:

(a) Prueba de una trayectoria



(b) Resultados del seguimiento

Figura 37: Pruebas del seguimiento de trayectorias

7.4. Pruebas del control remoto

Finalmente, el control remoto se ha probado sobre terreno agrícola para ver realmente la resistencia del vehículo. En este caso no hay una trayectoria predefinida, por lo que la prueba ha consistido más bien en sortear obstáculos y atravesar zonas de difícil maniobrabilidad, como pueden ser zonas de abundante vegetación y a diferentes superficies de diferente elevación (escalones, montículos, etc). Para ello se han empleado ambos controles diseñados, tanto el control mediante *joysticks* como el control mediante acelerómetro.

A continuación se muestran algunas imágenes de los diferentes obstáculos o terrenos que ha conseguido sortear el vehículo:

(a) Descenso de escalones

(b) Desplazamiento entre vegetación abundante

Figura 38: Pruebas del control remoto

8 Conclusiones

Finalmente se valorará de manera general el proyecto, y al mismo tiempo se presentarán tanto observaciones como posibles aspectos a mejorar de cara a futuras revisiones del prototipo. Además, se extraerán las conclusiones a las que se ha llegado durante y tras la finalización del trabajo.

Por un lado, los resultados obtenidos por el robot móvil para entornos agrícolas han sido satisfactorios. No solamente se ha conseguido montar y poner en marcha un prototipo de un robot móvil completamente funcional, sino que además se ha logrado que el vehículo se desenvuelva bien por un entorno accidentado o poco favorable, lo que era uno de los objetivos principales del proyecto. Además, se han desarrollado dos programas correspondientes a dos funcionalidades distintas; el seguimiento de trayectorias y el control remoto.

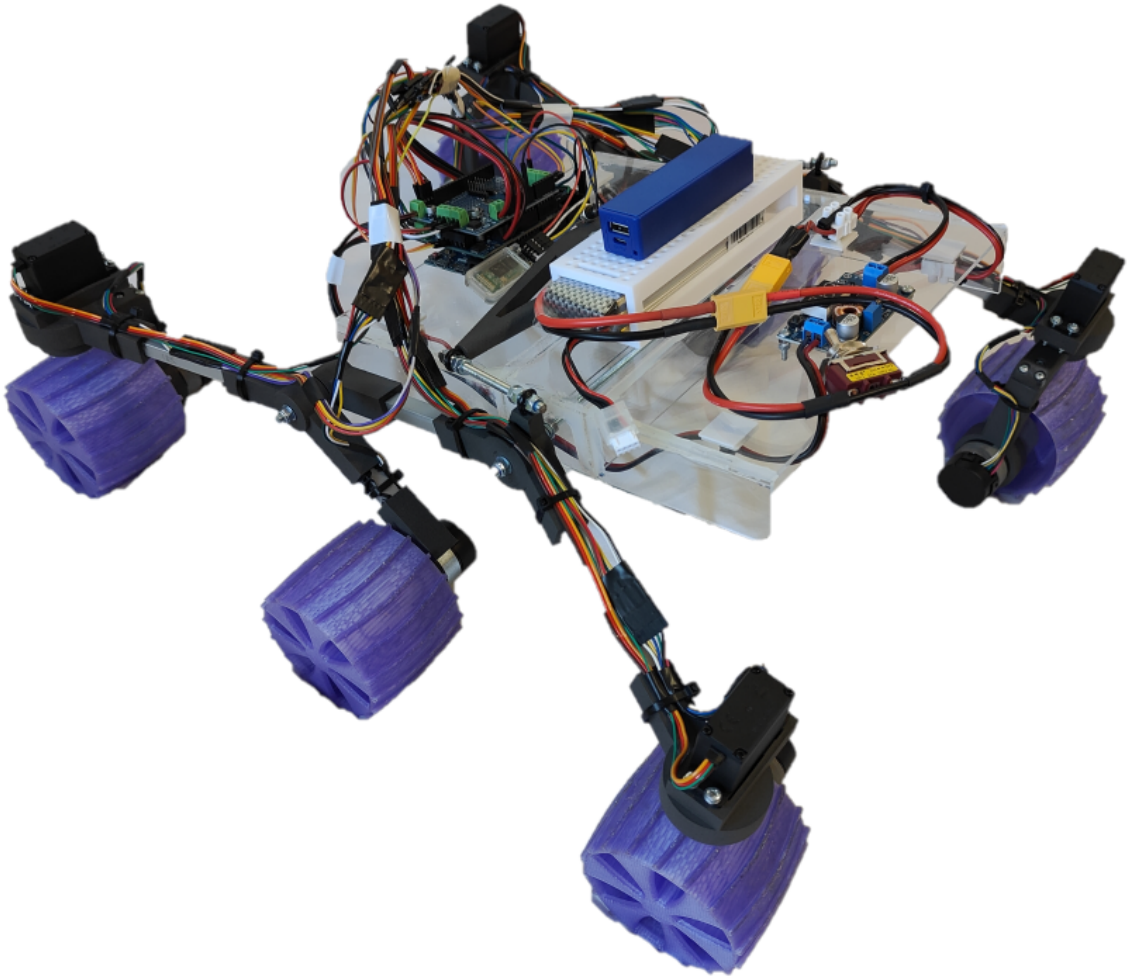


Figura 39: Montaje final del robot móvil

Por otro lado, debido entre otros aspectos a tratarse de un prototipo, hay algunos aspectos del trabajo que podrían estar sujetos a futuras mejoras. Uno de ellos es la propia estructura del robot, que pese a haber resistido correctamente todas las pruebas que se han realizado, podría fabricarse en materiales más resistentes y de mejor calidad. Del mismo modo, debido a problemas con los encoders incorporados de los motores DC, el control del seguimiento de trayectorias se ha tenido que realizar en bucle abierto, por lo que en algunas de las pruebas el vehículo se ha desviado de la trayectoria original debido a obstáculos del terreno. Con el objetivo de solucionar el problema, se propone el uso de GPS o similares para realizar correctamente el control en bucle cerrado, permitiendo al robot volver a la trayectoria objetivo en caso de desvío.

Finalmente, destacar que en varios puntos de este trabajo se ha descrito el robot diseñado como una 'base móvil para futuros proyectos'. Esto se debe a que, como se ha comentado anteriormente, este proyecto consiste en un prototipo que pueda servir como punto de partida en futuros proyectos. Algunas de las posibles funciones que podrían ser implementadas más adelante son las siguientes:

- Sistema de cámaras para inspección y monitorización de cultivos mediante visión artificial.
- Mantenimiento del estado del campo mediante tareas de deshierbe, aplicación de fertilizantes o pesticidas.
- Vigilancia y seguridad mediante sensores de movimiento y cámaras.
- Implementación de paneles para alimentación del sistema mediante energía solar.

Como conclusión, pese a las limitaciones de tiempo y presupuesto con las que se ha trabajado, así como los problemas que han podido surgir durante la realización del proyecto, se considera que se han cumplido satisfactoriamente todos los objetivos propuestos en el trabajo.

9 Bibliografía

A continuación se muestran por orden de aparición los enlaces a los distintos sitios web de donde se ha extraído tanto información como algunas de las imágenes utilizadas durante el proyecto, así como los artículos consultados.

9.1. Enlaces a sitios web

- Curiosity BTL (*última modificación 25/06/2022*)

https://bricolabs.cc/wiki/proyectos/curiosity_btl

- Portal Carbon Robotics

https://carbonrobotics.com/

- Portal Robotnik

https://robotnik.eu/es/

- Portal NASA - Curiosity

https://science.nasa.gov/mission/msl-curiosity/

- Portal Autodesk

https://www.autodesk.com/es

- Portal Arduino

https://www.arduino.cc/

- Información mecanismo *Rocker-Bogie* (*última modificación 07/05/2024*)

https://es.wikipedia.org/wiki/Rocker-bogie

- Imagen mecanismo *Rocker-Bogie*

https://docs.sunfounder.com/projects/galaxy-rvr/en/latest/lesson2_rocker_bogie.html

- Imagen partes *Rocker-Bogie*

https://www.researchgate.net/figure/The-rocker-bogie-suspension_fig1_372248374

- Diagrama del modelo de Ackermann

https://es.mathworks.com/help/robotics/ref/ackermannkinematicmodel.html

- Diagrama del modelo bicicleta

https://es.mathworks.com/help/robotics/ref/bicyclekinematics_es.html

- Portal Adafruit

https://www.adafruit.com/

9.2. Artículos consultados

- Lin-hui Li, Jing Lian, Bai-chao Chen, Jing Chang, Hai-yang Huang (2014)
  - *1398. Trajectory tracking and traction coordinating controller design for lunar rover based on dynamics and kinematics analysis*

Parte II

Planos

1 Planos, *layouts* y esquemas

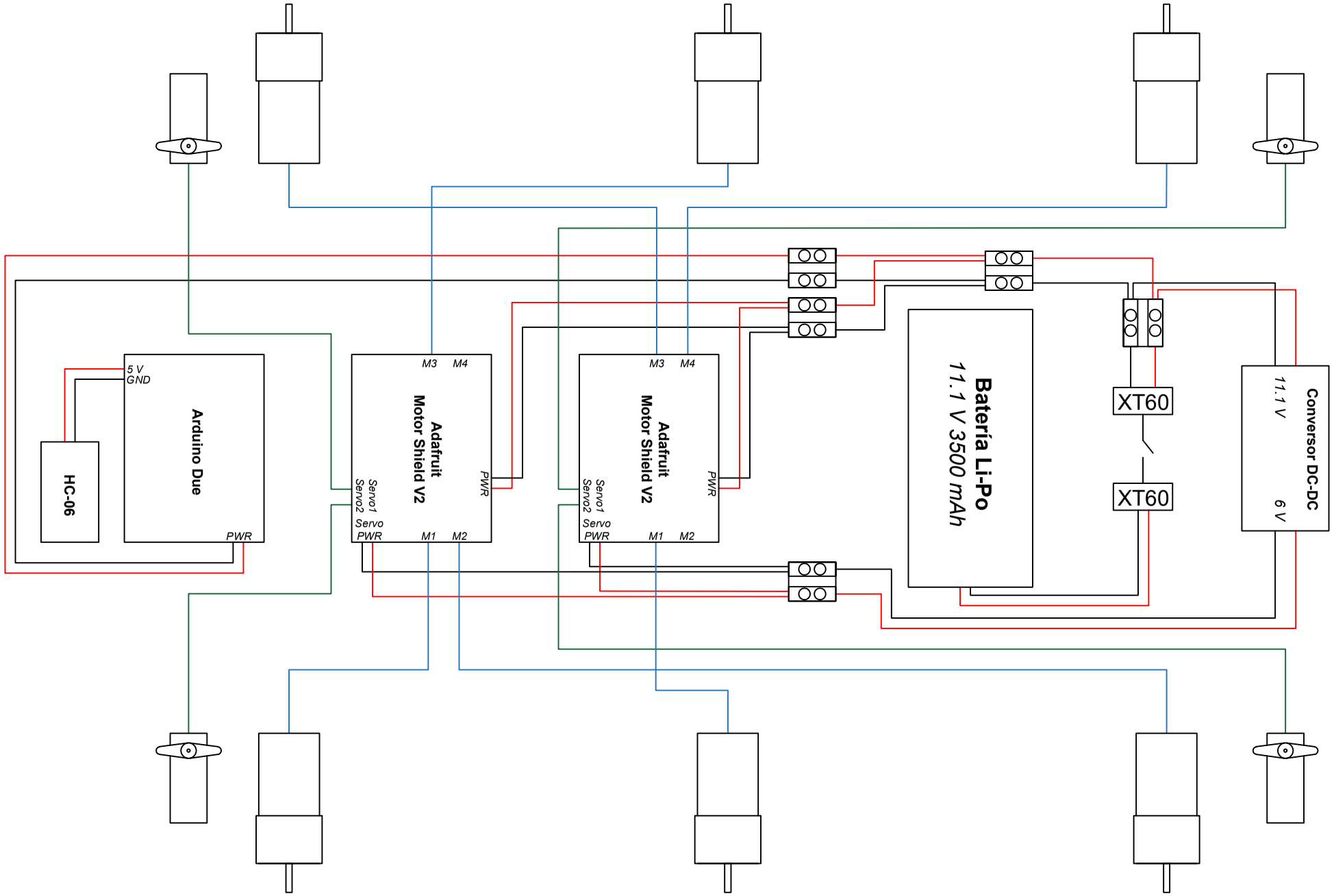
En este apartado se encuentran adjuntos los planos, *layouts* y esquemas eléctricos realizados a lo largo del proyecto.

El diseño de piezas en 3D no forma parte del alcance del proyecto, por lo que los planos adjuntos se limitarán a aquellos descritos a continuación:

- Esquema eléctrico del robot
- *Layout* del terreno de pruebas

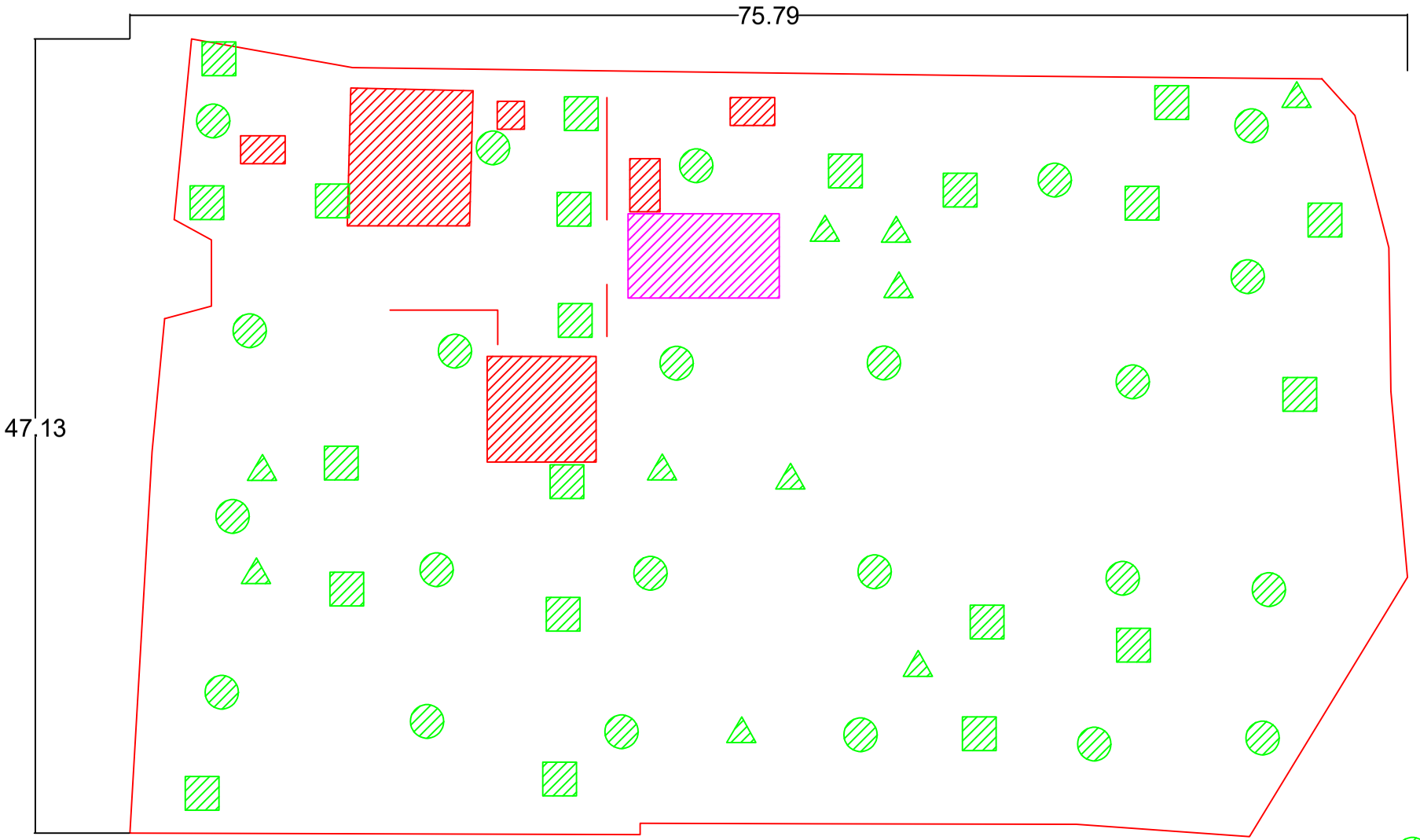
Ambos planos han sido realizados con AutoCAD, tal y como se ha mencionado en el apartado de antecedentes del documento Memoria.

1.1. Esquema eléctrico del robot



isa	CUSTOMER:	DRAWING DATA:	CREATION DATE:	Next:
			LAST MODIFICATION:	Anterior:
	PROJECT DATA: MONTAJE, PUESTA EN MARCHA Y CONTROL DE TRAYECTORIAS DE UN ROBOT MÓVIL PARA ENTORNOS AGRÍCOLAS	ESQUEMA ELÉCTRICO	PROJECT: MONTAJE, PUESTA EN MARCHA Y CONTROL DE TRAYECTORIAS DE UN ROBOT MÓVIL PARA ENTORNOS AGRÍCOLAS	Total:
			DRAWN: H.RAIGAL / REVIEWED:	SCALE:

1.2. *Layout* del terreno de pruebas



isa	CUSTOMER:	DRAWING DATA:	CREATION DATE:	Next:
			LAST MODIFICATION:	Anterior:
	PROJECT DATA: MONTAJE, PUESTA EN MARCHA Y CONTROL DE TRAYECTORIAS DE UN ROBOT MÓVIL PARA ENTORNOS AGRÍCOLAS	LAYOUT TERRENO DE PRUEBAS	PROJECT: MONTAJE, PUESTA EN MARCHA Y CONTROL DE TRAYECTORIAS DE UN ROBOT MÓVIL PARA ENTORNOS AGRÍCOLAS	Total:
				SCALE:
			DRAWN: H.RAIGAL / REVIEWED:	1:500

Parte III

Pliego de condiciones

1 Objeto

Las presentes especificaciones técnicas corresponden al diseño y montaje de un robot móvil para entornos agrícolas. En el apartado se describirán, por tanto, las condiciones de los materiales, de la ejecución, y las pruebas de servicio a realizar durante el proyecto.

Quedan excluidos del documento, y por tanto del alcance del trabajo, los aspectos descritos a continuación:

- Diseño y cálculo mecánico de los mecanismos *Rocker-Bogie*.
- Diseño y cálculo mecánico de las piezas 3D que forman parte del mecanismo.

2 Condiciones de los materiales

En el presente apartado se describirán las condiciones que deben cumplir los diversos materiales y componentes utilizados para la realización del proyecto.

Con el objetivo de mejorar la lectura, durante todo el apartado se seguirá la misma distribución presentada en el organigrama del proyecto, ubicado en el documento correspondiente a la memoria.

2.1. Mecánica

2.1.1. Cuerpo principal

- **Planchas de metacrilato**
  - Metacrilato de grosor 3 *mm*.
  - Dimensiones y cantidades:

DIMENSIONES METACRILATO		
Pieza	Uds.	Dimensiones
Plancha superior	1	240x280 *mm*
Plancha lateral	2	40x280 *mm*
Plancha transversal	2	40x240 *mm*

Cuadro 1: Lista de planchas de metacrilato

2.1.2. Mecanismo *Rocker-Bogie*

- **Elementos de unión en 3D**
  - Impresas en polvo de nylon.

ELEMENTOS 3D *ROCKER-BOGIE*			
Pieza	Uds.	Pieza	Uds.
Barra central	1	Unión en I	2
Unión en T	1	Unión en T (simétrica)	2
Unión en L	1	Unión en L (simétrica)	1
Soporte servo delantero	1	Soporte servo delantero (simétrica)	1
Soporte servo trasero	1	Soporte servo trasero (simétrica)	1
Soporte motor central	1	Soporte motor central (simétrica)	1
Soporte motor esquina	4	Unión servo-motor	4

Cuadro 2: lista de elementos de unión en 3D

- **Barra de unión**
  - Perfil de aluminio cuadrado 10x10 *mm*.
  - Grosor 1 *mm*.
  - Longitudes:

LONGITUDES PERFIL ALUMINIO	
Uds.	Dimensiones
2	135 *mm*
2	125 *mm*
2	115 *mm*
2	100 *mm*

Cuadro 3: Lista de las barras de unión

2.1.3. Ruedas

- **Ruedas**
  - 6 ruedas impresas en filamento PLA morado.
  - Dimensiones de 70 *mm* de ancho por 85 *mm* de diámetro.

2.2. Electrónica

2.2.1. Arduino

- **Arduino Due**
  - Fabricante Arduino.
  - Basada en microcontrolador de 32-bit con núcleo ARM.
  - 54 pines E/S digitales (12 de ellas salidas PWM), 12 entradas analógicas, 4 UARTs.
  - Voltaje de operación lógica de 3.3 *V*.
  - Voltaje de alimentación de entre 7 y 12 *V*.
  - Comunicación por I2C.

- **Adafruit Motor Shield V2**
  - Fabricante Adafruit.
  - Control de hasta 4 motores DC o dos motores paso a paso.
  - Control de hasta 2 servomotores, con posibilidad de alimentación independiente.
  - Integrado TB6612 MOSFET, proporciona 1.2 A por motor (hasta 3 A).
  - Dirección I2C configurable.
  - Posibilidad de configuración de lógica interna a 3.3 V.
  - Diseño apilable.
- **Módulo Bluetooth HC-06**
  - Voltaje de alimentación de 3.3 a 6 V.
  - Protocolo Bluetooth v1.1 / 2.0.
  - Comunicación serie TX/RX.

2.2.2. Motores y servos

- **Motores DC**
  - Fabricante Pololu (Modelo 37D Metal Gearmotors).
  - Diámetro 37 mm.
  - Voltaje de alimentación 12 V.
  - Corriente 0.2 A sin carga, 0.68 A en carga máxima.
  - Reductor 70:1 incorporado.
  - Encoder incorporado.
  - Voltaje de alimentación del encoder de 3.5 a 20 V.
  - Consumición máxima del encoder 10 mA.
- **Servomotores**
  - Fabricante FEETECH (Modelo FS5115M).
  - Voltaje de alimentación entre 4 y 8.4 V.
  - Voltaje de funcionamiento entre 4.8 y 6 V.
  - Consumo de 1.3 a 1.7 A.
  - Carga de 15 kg a capacidad máxima.
  - Rango de ancho de pulso de 500 a 2500 μsec (0^{o} a 180^{o}).
  - Pin de *feedback* incorporado.

2.2.3. Alimentación

- **Batería Li-Po**
  - Batería de polímero de litio de 3 celdas.
  - Voltaje 11.1 V.
  - Capacidad 3500 mAh.
  - Conector XT60.
  - Necesario cargador Balanceador IMAX B6 de 80 W.
- **Conversor DC-DC**
  - Modelo XL4015.
  - Conversor Buck (*step-down*).
  - Voltaje de entrada de 4 a 38 V, salida regulable de 1.25 a 36 V.
  - Corriente máxima de salida 5 A.
  - Voltímetro digital integrado.

COMPONENTES ELECTRÓNICA	
Componente	Uds.
ARDUINO	
Arduino Due	1
Adafruit Motor Shield V2	2
Módulo Bluetooth HC-06	1
MOTORES Y SERVOS	
Motores DC	6
Servomotores	4
ALIMENTACIÓN	
Batería Li-Po	1
Conversor DC-DC	1

Cuadro 4: Lista de componentes electrónicos

2.3. Otros elementos

En esta sección se incluyen otros elementos tales como ferretería, cableado o elementos de conexión.

OTROS ELEMENTOS			
Elemento	Uds.	Elemento	Uds.
FERRETERÍA			
Tornillo M5x50 *mm*	3	Varilla roscada M5 1 *m*	1
Tuerca M5	14	Arandela M5	17
Tuerca Autoblocante M5	9	Tronillo M3x30 *mm*	4
Tornillo M3x16 *mm*	24	Tuerca M3	12
Arandela M3	12	Tornillo M2.5x20 *mm*	20
Tuerca M2.5	20	Arandela M2.5	8
Alcayata cerrada 16x6x2.8 *mm*	4	Rodamiento 15x35x11 *mm*	4
ELECTRICIDAD Y CABLEADO			
Cable Rojo/Negro 1.5 mm^2 5 *m*	1	Conector XT60 Macho	1
Regleta de conexión	1	Conector alimentación Arduino	1
Cable Dupont MM 40 *cm*	1	Cable Dupont MH 40 *cm*	1
Cable Dupont MM 20 *cm*	1	Cable Dupont MH 20 *cm*	1
Interruptor XT60	1		
*Nota: El cable tipo Dupont se vende en packs de 40 uds., por lo que no se contabiliza unitariamente.			
SUJECIÓN			
Bridas		Sujeta-bridas adhesivo	6
Cinta adhesiva aislante	1	Pegamento universal	1
*Nota: Las bridas no se comercializan por unidades individuales			

Cuadro 5: Lista de elementos varios

3 Condiciones de ejecución

A continuación se describen, en caso de que sea necesario, las condiciones de la ejecución y el montaje de los diferentes subsistemas que forman el proyecto.

3.1. Mecánica

3.1.1. Cuerpo principal

Las planchas de metacrilato que forman el cuerpo principal se unirán mediante pegamento universal de fijación fuerte. Para facilitar la unión, las planchas poseen un perfil en forma de 'almena', además de ranuras para insertar las planchas transversales sobre las laterales. Para más solidez, una vez ensamblado el cuerpo se podrá reforzar la unión mediante silicona o similares.

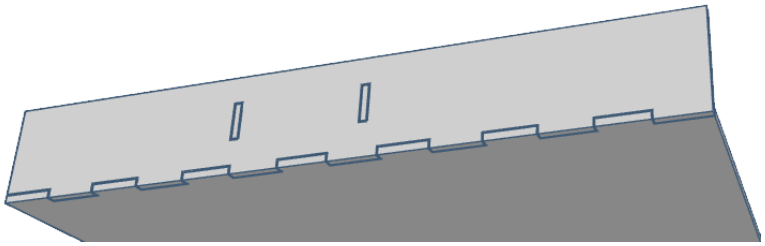


Figura 40: Unión entre planchas de metacrilato

Previamente a realizar cualquier unión, se deberán taladrar los agujeros correspondientes a la barra central, la varilla de unión de ambos mecanismos *Rocker-Bogie* (M5) y los componentes electrónicos que lo requieran (M3).

3.1.2. Mecanismo *Rocker-Bogie*

Las uniones móviles entre elementos se realizarán con tornillería M5, mientras que para las uniones fijas se empleará el perfil de aluminio cortado a diferentes medidas.

- **Uniones móviles:** Es necesario emplear arandelas para evitar la fricción entre elementos y mejorar la movilidad.
- **Uniones fijas:** El perfil de aluminio va insertado directamente en la ranura de la pieza, por lo que puede quedar poco sujeto. Para solucionarlo, se puede emplear cinta adhesiva alrededor del perfil para aumentar la superficie de agarre.

Ambos mecanismos se unirán a través del cuerpo principal mediante una varilla roscada de M5, sujeta mediante tuercas para evitar que se desplace.

3.1.3. Ruedas

Las ruedas han sido diseñadas a medida para los motores utilizados, por lo que solamente habrá que insertar el eje del motor en la ranura correspondiente de la rueda. No es necesario realizar ninguna sujeción adicional.

3.2. Electrónica

3.2.1. Arduino

La placa Arduino Due se fijará sobre el cuerpo de metacrilato mediante tornillos y tuercas M3. Las placas Adafruit Motor Shield V2, en cambio, van apiladas sobre la propia placa Arduino. Además, es necesario realizar algunas modificaciones sobre las mismas para su correcto funcionamiento:

1. Se debe cambiar el voltaje de operación de ambas placas a 3.3 V, en lugar de los 5 V por defecto. Para ello, se deberá soldar el *jumper* correspondiente a 3.3 V en la placa.

2. Se debe cambiar la dirección I2C de una de las dos placas para evitar conflicto. Para ello, se deberá soldar uno de los *jumpers* asignados para la tarea.

Además, sobre ambas placas se han realizado algunas modificaciones adicionales para facilitar la conexión de todos los elementos del proyecto.

3.2.2. Motores y servos

Los motores DC y servomotores van atornillados a los elementos 3D mediante tornillería variada de M2.5 y M3, dependiendo del elemento al que van unidos. Es necesario realizar las uniones lo más fuerte posible, ya que el robot está sometido a muchas vibraciones y se podrían aflojar fácilmente.

Por otro lado, tanto motores DC como servomotores se conectan al conjunto Arduino mediante cables Dupont de longitudes variadas, dependiendo de la distancia a la que se encuentre el elemento del lugar de conexión.

3.2.3. Alimentación

Para la sujeción de la batería Li-Po se ha diseñado e impreso en 3D una caja protectora que se fijará al cuerpo de metacrilato mediante cinta adhesiva, ya que permite una fácil retirada en caso de necesidad de carga o descarga de la batería. Además, se ha añadido un interruptor con conectores de tipo XT60 para facilitar la alimentación del conjunto.

Por último, el conversor DC-DC se atornillará al cuerpo del mismo modo que la placa Arduino, mediante el uso de tornillería M3.

4 Pruebas de servicio

Durante la realización del proyecto se irán realizando varias pruebas, desde el *testeo* inicial de componentes hasta la prueba final del conjunto completo. Dichas pruebas se expondrán a continuación, recalcando en cada caso los factores más relevantes a tener en cuenta a la hora de extraer conclusiones.

4.1. Prueba de componentes electrónicos

Antes de empezar con cualquier montaje, es necesario comprobar el correcto funcionamiento de cada componente de manera individual. Para ello, se realizarán en orden las siguientes pruebas:

1. **Arduino Due:** Para comprobar el funcionamiento de la placa Arduino, simplemente se conectará la placa al ordenador mediante cable y se tratará de ejecutar un programa sencillo, como por ejemplo el parpadeo de un LED integrado.

2. **Adafruit Motor Shield V2:** Para comprobar el funcionamiento y la correcta comunicación entre placas, se apilará el conjunto de las tres placas y se ejecutará sobre el Arduino un programa escáner de I2C (incluido en el Anexo, en la sección de Códigos). Si se detectan las direcciones I2C de las tres placas (Arduino y dos *shields*), el montaje será correcto y no habrá fallos de comunicación.

3. **Módulo Bluetooth HC-06:** Para comprobar el funcionamiento del módulo Bluetooth simplemente se realizarán las conexiones necesarias en la placa Arduino y se intentará vincular al dispositivo móvil.

4. **Motores DC y servomotores:** Finalmente se comprobarán los motores DC y los servomotores de manera tanto individual como en conjunto. Para ello, se intentarán mover los elementos uno a uno y posteriormente se tratará de mover el conjunto entero al mismo tiempo.

Si todas las pruebas han obtenido resultados satisfactorios, se podrá proseguir con el montaje de los componentes electrónicos sobre el cuerpo del robot. Una vez finalizado el montaje, es recomendable repetir las pruebas para ver si mantienen el mismo comportamiento una vez añadido el conjunto mecánico.

4.2. Prueba de movimiento

Con el conjunto mecánica-electrónica ensamblado, la siguiente prueba consiste en realizar movimientos sencillos con el robot, como por ejemplo avanzar unos pocos metros en línea recta o girar hacia izquierda o derecha. Si el robot es capaz de realizar movimientos de este tipo, ya se puede pasar a las últimas pruebas.

4.3. Prueba del control de trayectorias

Las pruebas del control de trayectorias consisten en probar a realizar distintas rutas sencillas dentro de un entorno controlado, por ejemplo en interiores amplios. Esta prueba solamente es útil para comprobar si el seguimiento de trayectorias funciona a grandes rasgos, más tarde ya se procederá a mejorar el programa y a probar el sistema en un entorno poco favorable siendo este el objetivo principal del proyecto.

4.4. Prueba del control remoto

Finalmente, la prueba del control remoto será similar a la del control de trayectorias, solo que empleando un dispositivo móvil para verificar la correcta comunicación entre emisor (móvil) y receptor (robot). Es conveniente realizar la prueba en un entorno controlado, con el objetivo de no dañar el sistema en el caso de algún error.

Parte IV

Presupuesto

1 Presupuesto del proyecto

En este capítulo se presentará el presupuesto del proyecto, realizado mediante el método de costes según naturaleza. El presupuesto se dividirá de la misma manera que el resto del trabajo, siguiendo la clasificación propuesta en el organigrama del proyecto.

Para la realización del proyecto se omitirá tanto el coste del cuerpo de metacrilato como el de las piezas impresas en 3D, ya que han sido proporcionados por los servicios de corte e impresión 3D de la universidad y no se dispone de ningún presupuesto. Sin embargo, seguirán incluyéndose los elementos en las tablas para reflejar su presencia dentro del proyecto.

1.1. Presupuesto de materiales

1.1.1. Mecánica

1.1.1.1. Cuerpo principal

CUERPO PRINCIPAL			
Artículo	Precio/Unidad (€)	Cantidad	Subtotal (€)
Plancha superior 240x280 *mm*	5,00	1	5,00
Plancha lateral 40x280 *mm*	2,00	2	4,00
Plancha transversal 40x240 *mm*	2,00	2	4,00
TOTAL CUERPO PRINCIPAL			**13,00 €**

Cuadro 6: Presupuesto del cuerpo principal

1.1.1.2. Mecanismo *Rocker-Bogie*

MECANISMO *ROCKER-BOGIE*			
Artículo	Precio/Unidad (€)	Cantidad	Subtotal (€)
Barra central	3,00	1	3,00
Unión en I	2,50	2	5,00
Unión en T	2,50	1	2,50
Unión en T (simétrica)	2,50	1	2,50
Unión en L	2,50	1	2,50
Unión en L (simétrica)	2,50	1	2,50
Soporte servo del.	2,50	1	2,50
Soporte servo del. (simétrica)	2,50	1	2,50
Soporte servo tras.	2,50	1	2,50
Soporte servo tras. (simétrica)	2,50	1	2,50
Soporte motor cent.	2,50	1	2,50
Soporte motor cent. (simétrica)	2,50	1	2,50
Soporte motor esquina	2,50	4	10,00
Unión servo-motor	2,50	4	10,00
Perfil de aluminio 1 m	5,49	1	5,49
TOTAL MECANISMO *ROCKER-BOGIE*			**58,49 €**

Cuadro 7: Presupuesto del mecanismo *Rocker-Bogie*

1.1.1.3. Ruedas

RUEDAS			
Artículo	Precio/Unidad (€)	Cantidad	Subtotal (€)
Rueda filamento PLA	2,00	6	12,00
TOTAL RUEDAS			**12,00 €**

Cuadro 8: Presupuesto de las ruedas

1.1.1.4. Resumen mecánica

RESUMEN MECÁNICA	
Conjunto	Subtotal (€)
Cuerpo principal	13,00
Mecanismo *Rocker-Bogie*	58,49
Ruedas	12,00
TOTAL MECÁNICA	**83,49 €**

Cuadro 9: Resumen del presupuesto de la mecánica

1.1.2. Electrónica

1.1.2.1. Arduino

ARDUINO			
Artículo	Precio/Unidad (€)	Cantidad	Subtotal (€)
Arduino Due	50,78	1	50,78
Adafruit Motor Shield V2	19,95	2	39,90
Módulo Bluetooth HC-06	3,75	1	3,75
TOTAL ARDUINO			**94,43 €**

Cuadro 10: Presupuesto del conjunto Arduino

1.1.2.2. Motores y servomotores

MOTORES Y SERVOMOTORES			
Artículo	Precio/Unidad (€)	Cantidad	Subtotal (€)
Motor DC	49,55	6	297,30
Servomotor	12,35	4	49,40
TOTAL MOTORES Y SERVOMOTORES			**346,70 €**

Cuadro 11: Presupuesto de motores y servomotores

1.1.2.3. Alimentación

ALIMENTACIÓN			
Artículo	Precio/Unidad (€)	Cantidad	Subtotal (€)
Batería Li-Po	19,95	1	19,95
Cargador batería Li-Po	25,90	1	25,90
Caja batería filamento PLA	2,00	1	2,00
Conversor DC-DC	4,50	1	4,50
TOTAL ALIMENTACIÓN			**52,35 €**

Cuadro 12: Presupuesto de la alimentación

1.1.2.4. Resumen electrónica

RESUMEN ELECTRÓNICA	
Conjunto	Subtotal (€)
Arduino	94,43
Motores y servomotores	346,70
Alimentación	52,35
TOTAL ELECTRÓNICA	**493,48 €**

Cuadro 13: Resumen del presupuesto de la electrónica

1.1.3. Otros elementos

1.1.3.1. Ferretería

FERRETERÍA			
Artículo	Precio/Unidad (€)	Cantidad	Subtotal (€)
Rodamiento 15x35x11 *mm*	3,76	4	15,04
Varilla roscada M5 1 *m*	1,26	1	1,26
Alcayata (Pack 4 uds.)	2,09	1	2,09
Tornillería métricas variadas	-	-	11,58
TOTAL FERRETERÍA			**29,97 €**

Cuadro 14: Presupuesto de ferretería

1.1.3.2. Electricidad y cableado

ELECTRICIDAD Y CABLEADO			
Artículo	Precio/Unidad (€)	Cantidad	Subtotal (€)
Cable Rojo/Negro 1.5 mm^2 5 *m*	3,89	1	3,89
Conectores XT60	1,90	1	1,90
Regleta de conexión	1,09	1	1,09
Conector alimentación Arduino	3,95	1	3,95
Interruptor XT60	9,14	1	9,14
Cable Dupont MM 40 *cm*	2,40	1	2,40
Cable Dupont MH 40 *cm*	2,40	1	2,40
Cable Dupont MM 20 *cm*	1,60	1	1,60
Cable Dupont MM 20 *cm*	1,60	1	1,60
*Nota: Todos los cables Dupont vienen en conjuntos de 40 uds.			
TOTAL ELECTRICIDAD Y CABLEADO			**27,97 €**

Cuadro 15: Presupuesto de electricidad y cableado

1.1.3.3. Sujeción

SUJECIÓN			
Artículo	Precio/Unidad (€)	Cantidad	Subtotal (€)
Paquete bridas 100 uds.	2,39	1	2,39
Sujeta-bridas adhesivo 10 uds.	3,49	1	3,49
Cinta adhesiva aislante	7,36	1	7,36
Pegamento universal	6,49	1	6,49
TOTAL SUJECIÓN			**19,73 €**

Cuadro 16: Presupuesto de sujeción

1.1.3.4. Resumen otros elementos

RESUMEN OTROS ELEMENTOS	
Conjunto	Subtotal (€)
Ferretería	29,97
Electricidad y cableado	27,97
Sujeción	19,73
TOTAL OTROS ELEMENTOS	**77,67 €**

Cuadro 17: Resumen del presupuesto de otros elementos

1.2. Presupuesto de mano de obra

MANO DE OBRA			
Descripción	Precio/Hora (€)	Horas	Subtotal (€)
MONTAJE			
Diseño 3D de las ruedas	20,00	2	40,00
Diseño 3D de los elementos de unión	20,00	2	40,00
Diseño 3D del cuerpo de metacrilato	20,00	2	40,00
Montaje del cuerpo de metacrilato	20,00	2	20,00
Montaje del mecanismo *Rocker-Bogie*	20,00	4	80,00
Ensamblaje del conjunto mecánico	20,00	2	40,00
Soldadura de las placas Adafruit	20,00	2	40,00
Montaje de ruedas, motores y servos	20,00	3	60,00
Cableado del conjunto electrónico	20,00	6	120,00
PROGRAMACIÓN			
Diseño CAD del *layout* del terreno	20,00	2	40,00
Diseño CAD del esquema eléctrico	20,00	4	80,00
Seguimiento de trayectorias	20,00	16	320,00
Control remoto	20,00	8	160,00
PRUEBAS			
Componentes y movimiento	20,00	8	160,00
Seguimiento de trayectorias	20,00	16	320,00
Control remoto	20,00	8	160,00
TOTAL MANO DE OBRA			**1.720,00 €**

Cuadro 18: Presupuesto de mano de obra

1.3. Resumen del presupuesto del proyecto

RESUMEN PRESUPUESTO PROYECTO	
Conjunto	Subtotal (€)
MATERIALES	
Mecánica	83,49
Electrónica	493,48
Otros elementos	77,67
MANO DE OBRA	
Mano de obra	1.720,00
TOTAL PRESUPUESTO PROYECTO	**2.374,64 €**

Cuadro 19: Resumen del presupuesto total del proyecto

Parte V

Anexos

A Cálculos desarrollados

En el presente apartado se desarrollarán los cálculos realizados a lo largo del proyecto, más concretamente, de cara al desarrollo del código para el seguimiento de trayectorias.

A.1. Cálculo de la velocidad lineal de las ruedas

Para poder llevar a cabo el seguimiento de trayectorias, primero es necesario conocer los parámetros de los motores con los que se trabaja. Para ello, se han extraído de los *datasheets* las siguientes especificaciones:

Relación de reducción	RPM_{MAX} **sin carga**	**Diámetro de las ruedas**
70:1	150	85 mm

Cuadro 20: Especificaciones técnicas de los motores DC

Ya conocidos los parámetros del motor, se puede calcular la velocidad lineal máxima de la siguiente manera:

$$VelocidadLineal_{MAX} = RPM_{MAX} * \frac{\pi * Diametro(m)}{60} = 150 * \frac{\pi * 0{,}085}{60} = 0{,}67 m/s$$

Finalmente, una vez obtenido el valor de la velocidad lineal máxima, se puede calcular la velocidad lineal en función de la señal PWM de la siguiente forma:

$$VelocidadLineal_{PWM} = \frac{PWM}{255} * VelocidadLineal_{MAX}$$

Por ejemplo, para un valor de PWM de 150, la velocidad lineal que tendría el robot es la siguiente:

$$VelocidadLineal_{150} = \frac{150}{255} * 0{,}67 = 0{,}39 m/s$$

Este cálculo es de gran importancia para el desarrollo del código ya que, conociendo los puntos a recorrer de una trayectoria, permite calcular el desplazamiento que el robot deberá realizar entre posición actual y objetivo.

B Códigos y programas

A continuación se encuentran adjuntos todos los códigos desarrollados a lo largo del proyecto. Se incluirán tanto los códigos de prueba como los códigos finales correspondientes al seguimiento de trayectorias y al control remoto, aunque solamente se desarrollarán estos últimos.

La distribución es la siguiente:

- **Códigos de prueba**
  - Prueba de la placa Arduino (parpadeo LED)
  - Escáner de dispositivos I2C
  - Prueba individual de motores DC
  - Prueba de servomotores
  - Prueba de movimiento del conjunto completo
- **Obtención de trayectorias**
- **Seguimiento de trayectorias**
- **Control remoto**

B.1. Códigos de prueba

B.1.1. Prueba de la placa Arduino (parpadeo LED)

El código de prueba de la placa Arduino consiste en el ejemplo *Blink* proporcionado en el propio IDE de Arduino, por lo que no requiere de una explicación más extensa. Solamente se ha utilizado para comprobar que la placa Arduino funciona correctamente a la hora de recibir y ejecutar un programa sencillo.

```
void setup()
{
  pinMode(LED_BUILTIN, OUTPUT);
}

void loop()
{
  digitalWrite(LED_BUILTIN, HIGH);
  delay(1000);
  digitalWrite(LED_BUILTIN, LOW);
  delay(1000);
}
```

B.1.2. Escáner de dispositivos I2C

El escáner de dispositivos I2C sirve para identificar los dispositivos conectados a la placa Arduino mediante dicho protocolo de comunicación. Si todo funciona correctamente, aparecerán las direcciones 0x70, 0x60 y 0x61.

```
#include <Wire.h>

void setup() {
  // Iniciar el bus I2C
  Wire1.begin();

  // Iniciar la comunicacion serie
  Serial.begin(9600);
  while (!Serial); // Esperar a que se abra el puerto serie
  Serial.println("\nDue I2C Scanner");
}

void loop() {
  byte error, address;
  int nDevices;

  Serial.println("Scanning...");

  nDevices = 0;
  for (address = 1; address < 127; address++) {
    // Iniciar una transmision I2C hacia la direccion especifica
    Wire1.beginTransmission(address);
    error = Wire1.endTransmission();

    if (error == 0) {
      Serial.print("I2C device found at address 0x");
      if (address < 16)
        Serial.print("0");
      Serial.print(address, HEX);
      Serial.println("  !");

      nDevices++;
    } else if (error == 4) {
      Serial.print("Unknown error at address 0x");
      if (address < 16)
        Serial.print("0");
      Serial.println(address, HEX);
    }
  }
  if (nDevices == 0)
    Serial.println("No I2C devices found\n");
  else
    Serial.println("done\n");

  delay(5000); // Esperar 5 segundos para el siguiente escaneo
}
```

B.1.3. Prueba individual de motores DC

Este código prueba uno de los motores DC conectado a una de las placas Adafruit, arrancándolo a baja velocidad durante un segundo. Con tal de probar todos los motores, solamente es necesario cambiar la placa y el puerto del motor.

```
// PRUEBA INDIVIDUAL DE MOTORES

#include <Wire.h>
#include <Adafruit_MotorShield.h>

Adafruit_MotorShield AFMS = Adafruit_MotorShield(0x61);

Adafruit_DCMotor *myMotor = AFMS.getMotor(4);

void setup()
{
  AFMS.begin(1600, &Wire1);

  myMotor->setSpeed(100);
  myMotor->run(FORWARD);
  delay(1000);
  myMotor->run(RELEASE);
}

void loop()
{
}
```

B.1.4. Prueba de servomotores

A diferencia del código anterior, este prueba los cuatro servos a la vez. Para ello, simplemente los posiciona en el ángulo deseado.

```
// PRUEBA INDIVIDUAL DE SERVOS

#include <Servo.h>

Servo servo1;
Servo servo2;

void setup()
{
  servo1.attach(9);
  servo2.attach(10);
  servo1.write(90);
  servo2.write(90);
}

void loop()
{
}
```

B.1.5. Prueba de movimiento del conjunto completo

Finalmente, la última prueba consiste en mover todo el conjunto de motores DC y servomotores a la vez. Para ello se ha desarrollado un código consistente en un movimiento en línea recta, un giro de los servomotores y un movimiento circular. Además, se ha implementado un botón para que el movimiento no inicie hasta que éste haya sido pulsado.

```
// PRUEBA GENERAL
#include <Wire.h>
#include <Adafruit_MotorShield.h>
#include <Servo.h>

// DECLARACION DE LOS SHIELDS
Adafruit_MotorShield shield60 = Adafruit_MotorShield(0x60);
Adafruit_MotorShield shield61 = Adafruit_MotorShield(0x61);

// DECLARACION DE LOS MOTORES
Adafruit_DCMotor *MFL = shield61.getMotor(3);
Adafruit_DCMotor *MML = shield60.getMotor(3);
Adafruit_DCMotor *MBL = shield61.getMotor(4);
Adafruit_DCMotor *MFR = shield60.getMotor(2);
Adafruit_DCMotor *MMR = shield61.getMotor(1);
Adafruit_DCMotor *MBR = shield60.getMotor(1);

// DECLARACION DE LOS SERVOS
Servo SF;
Servo SB;

const int pinBoton = 53;

void setup()
{
  // INICIALIZAR PLACAS
  shield60.begin(1600, &Wire1);
  shield61.begin(1600, &Wire1);

  // INICIALIZAR SERVOS
  SF.attach(9);
  SB.attach(10);
  SF.write(90);
  SB.write(90);

  // INICIALIZAR MOTORES
  MFL->setSpeed(50);
  MML->setSpeed(50);
  MBL->setSpeed(50);
  MFR->setSpeed(50);
  MMR->setSpeed(50);
  MBR->setSpeed(50);

  pinMode(pinBoton, INPUT_PULLUP);
}
```

```
46
47 void loop()
48 {
49   while (digitalRead(pinBoton) == HIGH) {
50   }
51
52   // PRUEBA
53   MFL->run(FORWARD);
54   MML->run(FORWARD);
55   MBL->run(FORWARD);
56   MFR->run(FORWARD);
57   MMR->run(FORWARD);
58   MBR->run(FORWARD);
59
60   delay(3000);
61
62   MFL->run(RELEASE);
63   MML->run(RELEASE);
64   MBL->run(RELEASE);
65   MFR->run(RELEASE);
66   MMR->run(RELEASE);
67   MBR->run(RELEASE);
68
69   delay(1000);
70
71   SF.write(50);
72   SB.write(130);
73
74   delay(1000);
75
76   MFL->run(FORWARD);
77   MML->run(FORWARD);
78   MBL->run(FORWARD);
79   MFR->run(FORWARD);
80   MMR->run(FORWARD);
81   MBR->run(FORWARD);
82
83   delay(3000);
84
85   MFL->run(RELEASE);
86   MML->run(RELEASE);
87   MBL->run(RELEASE);
88   MFR->run(RELEASE);
89   MMR->run(RELEASE);
90   MBR->run(RELEASE);
91 }
```

B.2. Obtención de trayectorias

Para la obtención de las trayectorias a seguir por el robot, se ha desarrollado un código para AutoCAD que permite obtener los n puntos que forman una *spline*. Para ello, simplemente basta con crear una *spline* cualquiera sobre el *layout* del terreno y seleccionar el número de puntos en que se desea dividirla. El programa guardará el resultado en un fichero de texto con dos filas, una para las coordenadas de los puntos en X y otra para las coordenadas en Y.

```
(defun C:DividirSpline ()
  (setq spline (car (entsel "\nSelecciona la spline a dividir: ")))
  (setq n (getint "\nIngresa el numero de puntos para dividir la
   spline: "))

  (setq splineLength (vlax-curve-getdistatparam spline (vlax-curve-
   getendparam spline)))
  (setq step (/ splineLength (1- n)))

  (setq outputFile (getfiled "Guardar como" "" "txt" 1))
  (setq file (open outputFile "w"))

  ;; Inicializar listas para las coordenadas X y Y
  (setq x-coords '())
  (setq y-coords '())

  (setq param 0.0)
  (repeat n
    (setq pt (vlax-curve-getpointatparam spline param))
    (setq x (rtos (car pt) 2 2))
    (setq y (rtos (cadr pt) 2 2))

    ;; Acumular coordenadas en las listas correspondientes
    (setq x-coords (append x-coords (list x)))
    (setq y-coords (append y-coords (list y)))

    (setq param (+ param step))
  )

  ;; Escribir coordenadas X en la primera fila
  (write-line (apply 'strcat (mapcar '(lambda (v) (strcat v ",")) x
   -coords)) file)
  ;; Escribir coordenadas Y en la segunda fila
  (write-line (apply 'strcat (mapcar '(lambda (v) (strcat v ",")) y
   -coords)) file)

  (close file)

  (princ "\nSpline dividida y coordenadas guardadas en el archivo
   .")
  (princ)
)
```

B.3. Seguimiento de trayectorias

A continuación se va a desarrollar el funcionamiento del código del seguimiento de trayectorias. Tal y como se ha expuesto en el diagrama de flujo presentado anteriormente, el programa consiste en un bucle que recorre uno a uno los puntos que forman una trayectoria, calculando la dirección a tomar y el desplazamiento que debe hacer el robot para llegar a cada *waypoint*.

Primero, en la sección del código conocida como preámbulo, se añaden las librerías necesarias, así como se declaran todos los objetos y variables de utilidad como las *shields*, motores y servomotores, etc. También se definirán aquí aspectos como la velocidad de los motores o los puntos que forman la trayectoria a seguir.

```
// INCLUDES-------------------------------------------------
#include <Adafruit_MotorShield.h>
#include <Servo.h>
#include <math.h>

// DECLARACION DE LOS SHIELDS-------------------------------
Adafruit_MotorShield shield60 = Adafruit_MotorShield(0x60);
Adafruit_MotorShield shield61 = Adafruit_MotorShield(0x61);

// DECLARACION DE LOS MOTORES-------------------------------
Adafruit_DCMotor *MFL = shield61.getMotor(3);
Adafruit_DCMotor *MML = shield60.getMotor(3);
Adafruit_DCMotor *MBL = shield61.getMotor(4);
Adafruit_DCMotor *MFR = shield60.getMotor(2);
Adafruit_DCMotor *MMR = shield61.getMotor(1);
Adafruit_DCMotor *MBR = shield60.getMotor(1);

// DECLARACION DE LOS SERVOS--------------------------------
Servo SF;
Servo SB;

// VECTORES DE PUNTOS---------------------------------------
const float X[] = {};
const float Y[] = {};
const int numPuntos = ;

// CONSTANTES-----------------------------------------------
const int velocidadMotores = ; // 0 a 255
const float velocidadLineal = ; // velocidadMotores*0.67/255
const int pinBoton = 53;
```

A continuación, aún dentro del preámbulo, se definirán las funciones que se van a utilizar a lo largo del programa. Estas son algunas de las más importantes:

- **calcularAnguloSF:** Calcula el ángulo al que deeberán situarse los dos servomotores delanteros. Para ello, recibe como argumento las coordenadas X e Y del punto actual y el punto de destino, con lo que calcula el ángulo mediante la arco tangente de las diferencias y finalmente devuelve el resultado en grados, restringiéndolo de 0 a 180º. Para determinar el ángulo de los servomotores traseros, solamente habrá que restar el resultado obtenido a 180, obteniendo el ángulo suplementario.
- **calcularDistancia:** Calcula la distancia en línea recta entre el punto actual y el punto destino mediante Pitágoras, devolviendo el resultado en metros.
- **calcularTiempo:** Calcula el tiempo que deberá tardar el vehículo en desplazarse desde el punto actual al punto objetivo en función de la velocidad lineal del robot y la distancia entre puntos, devolviendo el resultado en milisegundos.

```
// FUNCIONES ------------------------------------------------
void motoresON()
{
  MFL->run(FORWARD);
  MML->run(FORWARD);
  MBL->run(FORWARD);
  MFR->run(FORWARD);
  MMR->run(FORWARD);
  MBR->run(FORWARD);
}

void motoresOFF()
{
  MFL->run(RELEASE);
  MML->run(RELEASE);
  MBL->run(RELEASE);
  MFR->run(RELEASE);
  MMR->run(RELEASE);
  MBR->run(RELEASE);
}

void moverServos(int index)
{
  float xObjetivo = X[index+1];
  float yObjetivo = Y[index+1];
  float xActual = X[index];
  float yActual = Y[index];

  int anguloSF = calcularAnguloSF(xObjetivo, yObjetivo, xActual,
   yActual);
  int anguloSB = calcularAnguloSB(anguloSF);

  SF.write(anguloSF);
```

```
33    SB.write(anguloSB);
34  }
35
36  int calcularAnguloSF(float xObjetivo, float yObjetivo, float
        xActual, float yActual)
37  {
38    float deltaX = xObjetivo - xActual;
39    float deltaY = yObjetivo - yActual;
40
41    float anguloRadianes = atan2(deltaY, deltaX);
42    float anguloGrados = radiansToDegrees(anguloRadianes);
43
44    return constrain(90 + anguloGrados, 0, 180);
45  }
46
47  int calcularAnguloSB(int anguloSF)
48  {
49    return constrain(180 - anguloSF, 0, 180);
50  }
51
52  float radiansToDegrees(float radianes)
53  {
54    return radianes * (180.0 / PI);
55  }
56
57  float calcularDistancia(int index)
58  {
59    float xObjetivo = X[index+1];
60    float yObjetivo = Y[index+1];
61    float xActual = X[index];
62    float yActual = Y[index];
63
64    float deltaX = xObjetivo - xActual;
65    float deltaY = yObjetivo - yActual;
66
67    return sqrt(pow(deltaX, 2) + pow(deltaY, 2));
68  }
69
70  float calcularTiempo(float distancia)
71  {
72    return distancia / velocidadLineal * 1000;
73  }
```

En un programa para Arduino, el *setup* es el bloque donde se situarán las instrucciones que solamente se quiera ejecutar una vez al inicio del programa. En este caso, se inicializarán los diversos componentes con los valores deseados.

```
// SETUP-----------------------------------------------------
void setup()
{
  // INICIALIZAR PLACAS
  shield60.begin(1600, &Wire1);
  shield61.begin(1600, &Wire1);

  // INICIALIZAR SERVOS
  SF.attach(9);
  SB.attach(10);
  SF.write(90);
  SB.write(90);

  // INICIALIZAR MOTORES
  MFL->setSpeed(velocidadMotores);
  MML->setSpeed(velocidadMotores);
  MBL->setSpeed(velocidadMotores);
  MFR->setSpeed(velocidadMotores);
  MMR->setSpeed(velocidadMotores);
  MBR->setSpeed(velocidadMotores);

  // INICIALIZAR BOTON
  pinMode(pinBoton, INPUT_PULLUP);
}
```

Finalmente, el *loop* es aquella sección que se repetirá de manera infinita. Aquí es donde toman lugar las acciones descritas en el diagrama de flujo del programa.

```
// LOOP------------------------------------------------------
void loop()
{
  // ESPERAR A QUE SE PULSE EL BOTON DE INICIO
  while (digitalRead(pinBoton) == HIGH)
  {
  }

  // RECORRER LOS PUNTOS DE LA TRAYECTORIA
  for (int i = 0; i < numPuntos - 1; i++)
  {
    float distancia = calcularDistancia(i);
    int tiempo = calcularTiempo(distancia);
    moverServos(i);
    motoresON();
    delay(tiempo);
    motoresOFF();
  }

  // DETENER LOS MOTORES AL FINAL DE LA TRAYECTORIA
  motoresOFF();
}
```

B.4. Control remoto

Por último, con el objetivo de ofrecer al usuario una manera más fácil de controlar el vehículo, se ha desarrollado un control remoto desde un dispositivo móvil mediante Bluetooth que permite manejar el robot de dos maneras distintas.

```
// INCLUDES-----------------------------------------------
#include <Adafruit_MotorShield.h>
#include <Servo.h>
#include <math.h>

// DECLARACION DE LOS SHIELDS-----------------------------
Adafruit_MotorShield shield60 = Adafruit_MotorShield(0x60);
Adafruit_MotorShield shield61 = Adafruit_MotorShield(0x61);

// DECLARACION DE LOS MOTORES-----------------------------
Adafruit_DCMotor *MFL = shield61.getMotor(3);
Adafruit_DCMotor *MML = shield60.getMotor(3);
Adafruit_DCMotor *MBL = shield61.getMotor(4);
Adafruit_DCMotor *MFR = shield60.getMotor(2);
Adafruit_DCMotor *MMR = shield61.getMotor(1);
Adafruit_DCMotor *MBR = shield60.getMotor(1);

// DECLARACION DE LOS SERVOS------------------------------
Servo SF;
Servo SB;

// CONSTANTES---------------------------------------------
int x, y, xPad, yPad, xDir=90, yVel=0, direccion, velocidad;
char charAcelerometro;
float rollDir, pitchVel;
const int pinBoton = 53;
byte inByte[7];

// FUNCIONES----------------------------------------------
void motoresForward(int velocidadMotores)
{
  MFL->setSpeed(velocidadMotores);
  MML->setSpeed(velocidadMotores);
  MBL->setSpeed(velocidadMotores);
  MFR->setSpeed(velocidadMotores);
  MMR->setSpeed(velocidadMotores);
  MBR->setSpeed(velocidadMotores);

  MFL->run(FORWARD);
  MML->run(FORWARD);
  MBL->run(FORWARD);
  MFR->run(FORWARD);
  MMR->run(FORWARD);
  MBR->run(FORWARD);
}

void motoresBackward(int velocidadMotores)
```

```
48 {
49   MFL->setSpeed(velocidadMotores);
50   MML->setSpeed(velocidadMotores);
51   MBL->setSpeed(velocidadMotores);
52   MFR->setSpeed(velocidadMotores);
53   MMR->setSpeed(velocidadMotores);
54   MBR->setSpeed(velocidadMotores);
55 
56   MFL->run(BACKWARD);
57   MML->run(BACKWARD);
58   MBL->run(BACKWARD);
59   MFR->run(BACKWARD);
60   MMR->run(BACKWARD);
61   MBR->run(BACKWARD);
62 }
63 
64 void moverServos(int angulo)
65 {
66   SF.write(angulo);
67   SB.write(180-angulo);
68 }
69 
70 // SETUP----------------------------------------------------
71 void setup()
72 {
73   // INICIALIZAR PUERTOS SERIE
74   Serial.begin(115200);
75   Serial1.begin(115200);
76 
77   // INICIALIZAR PLACAS
78   shield60.begin(1600, &Wire1);
79   shield61.begin(1600, &Wire1);
80 
81   // INICIALIZAR SERVOS
82   SF.attach(9);
83   SB.attach(10);
84   SF.write(90);
85   SB.write(90);
86 
87   // INICIALIZAR BOTON
88   pinMode(pinBoton, INPUT_PULLUP);
89 }
90 
91 void loop()
92 {
93   do
94   {
95     if(Serial1.available()>=7)
96     {
97       for(int i=1 ; i<=7 ; i++)
98       {
99         inByte[i]=Serial1.read();
100       }
```

```
101
102         if(inByte[2]=='X' && inByte[5]=='Y')
103         {
104           x=(inByte[3]-48)*10+(inByte[4]-48);
105           xPad=map(x,10,99,120,60);
106           y=(inByte[6]-48)*10+(inByte[7]-48);
107           yPad=map(y,10,99,255,-255);
108         }
109
110         if(inByte[1]=='V')
111         {
112           yVel=yPad;
113           if(yVel >= 0) motoresForward(yVel);
114           else
115           {
116             yVel = yVel*-1;
117             motoresBackward(yVel);
118           }
119         }
120
121         if(inByte[1]=='D')
122         {
123           xDir=xPad;
124           moverServos(xDir);
125         }
126       }
127     } while(digitalRead(pinBoton) == HIGH);
128
129     do
130     {
131       if(Serial1.available()>0)
132       {
133         charAcelerometro=Serial1.read();
134         if(charAcelerometro=='A')
135         {
136           rollDir=Serial1.parseFloat();
137           if(rollDir > 50) rollDir = 50;
138           if(rollDir < -50) rollDir = -50;
139           direccion = map(rollDir, -50, 50, 120, 60);
140           moverServos(direccion);
141
142           pitchVel=Serial1.parseFloat();
143           if(pitchVel > 50) pitchVel = 50;
144           if(pitchVel < -50) pitchVel = -50;
145           velocidad = map(pitchVel, -50, 50, 255, -255);
146           if(velocidad >= 0) motoresForward(velocidad);
147           else motoresBackward(abs(velocidad));
148         }
149       }
150     } while(digitalRead(pinBoton) == HIGH);
151 }
```

C Objetivos de Desarrollo Sostenible

En este apartado se desarrollará el grado de relación del trabajo con los Objetivos de Desarrollo Sostenible, o ODS.

OBJETIVOS DE DESARROLLO SOSTENIBLE			
	Alto	Medio	Bajo
1. Fin de la pobreza			X
2. Hambre cero		X	
3. Salud y bienestar			X
4. Educación de calidad			X
5. Igualdad de género			X
6. Agua limpia y saneamiento			X
7. Energía asequible y no contaminante		X	
8. Trabajo decente y crecimiento económico		X	
9. Industria, innovación e infraestructuras	X		
10. Reducción de las desigualdades			X
11. Ciudades y comunidades sostenibles		X	
12. Producción y consumo responsables		X	
13. Acción por el clima	X		
14. Vida submarina			X
15. Vida de ecosistemas terrestres	X		
16. Paz, justicia e instituciones sólidas			X
17. Alianzas para lograr objetivos			X

Cuadro 21: Grado de relación del trabajo con los ODS

Los ODS con un grado de relación más alto con el proyecto, por tanto, son:

- **9. Industria, innovación e infraestructuras:** El sector agrario es uno de los sectores más estancados a nivel tecnológico. Por ello, es necesario crear una infraestructura de tecnologías de la información y la comunicación que permita este tipo de proyectos en el campo.
- **13. Acción por el clima:** Uno de los mayores problemas en el entorno rural es el uso de combustibles fósiles y maquinaria antigua. Este proyecto busca ofrecer una alternativa más sostenible, pudiendo por ejemplo alimentarse exclusiva o parcialmente mediante placas fotovoltaicas.
- **15. Vida de ecosistemas terrestres:** Directamente relacionado con el punto anterior, eliminar o reducir el uso de maquinaria pesada alimentada mediante combustibles fósiles puede mejorar considerablemente las condiciones de vida de la fauna y flora que habita en entornos rurales.

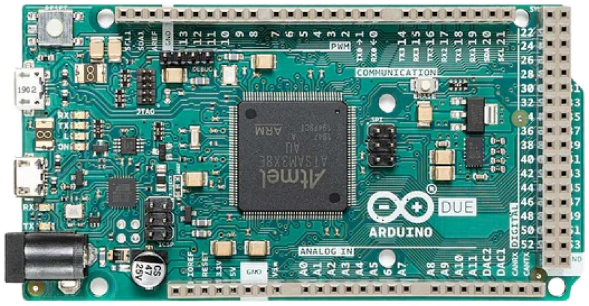
D *Datasheets* de los componentes

En este apartado se incluirán los *datasheets* de los diferentes componentes electrónicos que forman parte del trabajo. La distribución que se seguirá es la siguiente:

1. **Conjunto Arduino**
    a) Arduino Due
    b) Adafruit Motor Shield V2
2. **Conjunto motores y servomotores**
    a) Motor DC
    b) Servomotor

D.1. Conjunto Arduino

Product Reference Manual
SKU: A000062



Description

The Arduino Due is a groundbreaking microcontroller board featuring the Atmel SAM3X8E ARM Cortex-M3 CPU, making it the first Arduino board built around a 32-bit ARM core microcontroller. With its 54x digital input/output pins, 12x analog inputs, 4x UARTs, USB OTG capability, and 84 MHz clock, the Due offers enhanced performance and versatility for a wide range of projects. Compatible with all Arduino shields designed for operation at 3.3 V and compliant with the 1.0 Arduino pinout standard, the Due is a powerful tool for both beginners and experienced makers alike.

Target Areas

Embedded Systems Development, Robotics, 3D Printing, CNC Machines, Prototyping

Contents

1 Application Examples

The Arduino Due combines the performance of the Atmel SAM3X8E microcontroller with the flexibility of the Arduino platform, offering a versatile solution for developers, hobbyists, and professionals alike. With its 32-bit architecture and clock speed of 84 MHz, the Due delivers robust performance for demanding applications.

- **Embedded Systems Development:** The Arduino Due can be utilized to create a real-time data acquisition system for monitoring and analyzing environmental parameters in industrial settings. By interfacing sensors such as temperature, humidity, and pressure sensors with the Due's abundant I/O pins, developers can capture real-time data and process it using the Due's powerful microcontroller. The system can then transmit this data wirelessly or via USB to a host computer for analysis, allowing for continuous monitoring and remote management of critical processes.
- **Robotics:** The Arduino Due can serve as the brain of an autonomous mobile robot capable of navigating and interacting with its environment. By integrating sensors such as ultrasonic range finders, gyroscopes, and encoders, developers can equip the robot with perception capabilities to sense its surroundings and detect obstacles. Using the Due's abundant I/O pins and powerful processing capabilities, algorithms for localization, mapping, and path planning can be implemented to enable autonomous navigation. Additionally, actuators such as motors or servos can be controlled by the Due to execute motion commands, allowing the robot to move and manipulate objects in its environment autonomously.
- **3D Printing & CNC Machines:** The Arduino Due can function as a versatile controller for DIY projects. By interfacing stepper motor drivers and end-stop switches with the Due's numerous I/O pins, enthusiasts can create their own 3D printers or CNC machines. The Due's high-speed processing capabilities enable precise control of stepper motors for accurate positioning and movement.
- **Prototyping:** The Arduino Due serves as an invaluable tool for quickly iterating and testing new ideas for IoT devices. By leveraging the Due's extensive I/O capabilities and compatibility with various sensors, communication modules, and actuators, developers can rapidly assemble and test prototypes of IoT devices. Whether it is a smart home sensor node, a weather station, or a remote monitoring system, the Arduino Due provides a flexible platform for integrating components, writing firmware, and validating functionality. With the Due's support for Arduino libraries and easy-to-use development environment, prototypers can focus on innovation and experimentation, accelerating the process of bringing ideas to fruition.

2 Features

2.1 General Specifications Overview

The Arduino Due is a versatile microcontroller board designed for a wide range of applications. Powered by the Atmel SAM3X8E ARM Cortex-M3 CPU, it offers high performance and a robust set of features, making it suitable for complex projects. The Due's 32-bit architecture provides enhanced processing capabilities compared to traditional Arduino boards. Designed with a similar form factor to the Arduino® Mega, it maintains compatibility with most Arduino shields through its extensive set of I/O pins and headers. The following table summarizes the board's main features.

Feature	Description
Microcontroller	**Atmel SAM3X8E ARM Cortex-M3** 32-bit ARM Cortex-M3 / 84 MHz Clock speed
Memory	**SAM3X** 512 KB Flash / 96 KB SRAM (divided into two banks: 64 KB and 32 KB)
USB-to-serial	**ATmega16U2** connected to the SAM3X hardware UART
Digital Inputs	Digital Inputs not 5 V compatible (x54)
Analog Inputs	The Due's analog inputs pins measure from ground to a maximum value of 3.3 V (x12)
PWM Pins	PWM Pins with 8 bits resolution (x12)
Communication	UART (x4), I2C (x2), SPI (x1 SPI header), Native USB port (x1), Programming USB port (x1)
Power	Input voltage (VIN): 7-12 VDC / DC Current per I/O Pin: 8 mA
Dimensions	101.6 mm x 53.34 mm
Weight	36 g
Operating Temperature	-40 °C to +85 °C
Certifications	CE/RED, UKCA, FCC, IC, RCM, RoHS, REACH, WEEE

2.2 Microcontroller

Component	Details
Atmel SAM3X8E	32-bit ARM Cortex-M3 at 84 MHz
Flash Memory	512 KB
Programming Memory	96 KB SRAM (divided into two banks: 64 KB and 32 KB)

2.3 Inputs

Characteristics	Details
Number of inputs	54x digital inputs, 12x analog inputs
Inputs overvoltage protection	Yes
Antipolarity protection	Yes

2.4 Outputs

Characteristics	Details
DAC1 and DAC2	True analog output 12-bits resolution (4096 levels)
PWM outputs	12x PWM outputs

3 Accessories

- USB Cable Type-A Male to Micro Type-B Male (Not included)

4 Related Products

- Arduino Mega Proto Shield Rev3 (A000080)
- Arduino 4 Relays Shield (A000110)
- Arduino Motor Shield Rev3 (A000079)

5 Rating

5.1 Recommended Operating Conditions

Symbol	Description	Min	Typ	Max	Unit
V_{IN}	Input voltage from VIN pad	6.0	7.0	16	V
V_{USB}	Input voltage from USB connector	4.8	5.0	5.5	V
V_{DD}	Input high-level voltage	0.7*V_{DD}		V_{DD}	V
V_{IL}	Input low-level voltage	0		0.3*V_{DD}	V
T_{OP}	Operating Temperature	-40	25	85	°C

Note: V_{DD} controls the logic level and is connected to the 3.3 V power rail. V_{AREF} is for the analog logic.

5.2 Power Specification

Property	Min	Typ	Max	Unit
Supply voltage	7.0	-	12	V
Permissible range	6.0	-	16	V

Safety Note: Unlike most traditional Arduino boards, the Arduino Due board runs at 3.3 V. Keep in mind the maximum voltage that the I/O pins can tolerate is 3.3 V. Applying voltages higher than 3.3 V to any I/O pin could damage the board.

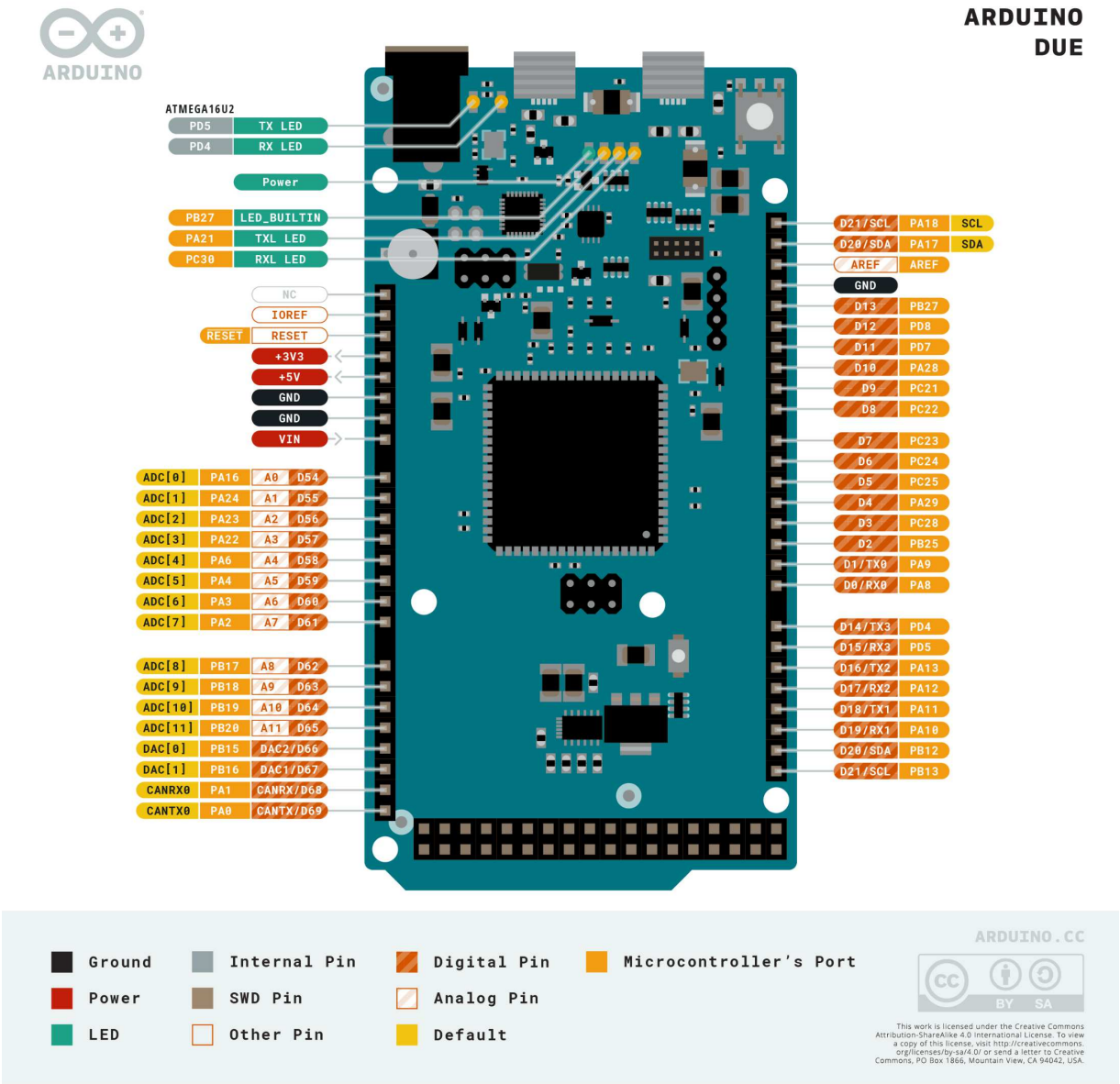
5.3 Current Consumption

Parameter	Symbol	Min	Typ	Max	Unit
Normal Mode Current Consumption	I_{NM}	130	---	800	mA

6 Functional Overview

6.1 Pinout

The Arduino Due pinout is shown in the following figure.



Arduino Due pinout

Safety Note: Disconnect power before board modifications to avoid short-circuiting.

6.2 Full Pinout Table

The full pinout of the Arduino Due is available in the following tables.

6.2.1 Board's 24-Pin Header

Pin	Function	Type	Description
1	NC	NC	Not Connected
2	IOREF	IOREF	Reference for digital logic voltage - connected to 3.3 V
3	Reset	Reset	Reset
4	+3V3	Power	+3V3 Power Rail
5	+5V	Power	+5V Power Rail
6	GND	Power	Ground
7	GND	Power	Ground
8	VIN	Power	Voltage Input
9	A0	Analog	Analog input 0 / GPIO
10	A1	Analog	Analog input 1 / GPIO
11	A2	Analog	Analog input 2 / GPIO
12	A3	Analog	Analog input 3 / GPIO
13	A4	Analog	Analog input 4 / GPIO
14	A5	Analog	Analog input 5 / GPIO
15	A6	Analog	Analog input 6 / GPIO
16	A7	Analog	Analog input 7 / GPIO
17	A8	Analog	Analog input 8 / GPIO
18	A9	Analog	Analog input 9 / GPIO
19	A10	Analog	Analog input 10 / GPIO
20	A11	Analog	Analog input 11 / GPIO
21	DAC0	Analog	Digital to Analog Converter 0
22	DAC1	Analog	Digital to Analog Converter 1
23	CANRX	Digital	CAN Bus Receiver
24	CANTX	Digital	CAN Bus Transmitter

Board's 24-Pin Header pinout

6.2.2 Board's 26-Pin Header

Pin	Function	Type	Description
1	D21/SCL1	Digital	GPIO 21 / I2C 1 Clock
2	D20/SDA1	Digital	GPIO 20 / I2C 1 Dataline
3	AREF	Digital	Analog Reference Voltage
4	GND	Power	Ground
5	D13/SCK	Digital	GPIO 13 / SPI Clock (PWM~)
6	D12/CIPO	Digital	GPIO 12 / SPI Controller In Peripheral Out (PWM~)
7	D11/COPI	Digital	GPIO 11 / SPI Controller Out Peripheral In (PWM~)
8	D10/CS	Digital	GPIO 10 / SPI Chip Select (PWM~)
9	D9/SDA2	Digital	GPIO 9 / I2C 2 Dataline (PWM~)
10	D8/SCL2	Digital	GPIO 8 / I2C 2 Clockline (PWM~)
11	D7	Digital	GPIO 7 (PWM~)
12	D6	Digital	GPIO 6 (PWM~)
13	D5	Digital	GPIO 5 (PWM~)
14	D4	Digital	GPIO 4 (PWM~)
15	D3	Digital	GPIO 3 (PWM~)
16	D2	Digital	GPIO 2 (PWM~)
17	D1/TX0	Digital	GPIO 1 / Serial 0 Transmitter
18	D0/TX0	Digital	GPIO 0 / Serial 0 Receiver
19	D14/TX3	Digital	GPIO 14 / Serial 3 Transmitter
20	D15/RX3	Digital	GPIO 15 / Serial 3 Receiver
21	D16/TX2	Digital	GPIO 16 / Serial 2 Transmitter
22	D17/RX2	Digital	GPIO 17 / Serial 2 Receiver
23	D18/TX1	Digital	GPIO 18 / Serial 1 Transmitter
24	D19/RX1	Digital	GPIO 19 / Serial 1 Receiver
25	D20/SDA	Digital	GPIO 20 / I2C 0 Dataline
26	D21/SCL	Digital	GPIO 21 / I2C 0 Clock

Board's 26-Pin Header pinout

6.2.3 SPI

The board provides an SPI interface and full access to its pinout as it can be seen in the following table.

Pin	Function	Type	Description
1	CIPO	Internal	Controller In Peripheral Out
2	+5V	Internal	Power Supply of 5V
3	SCK	Internal	Serial Clock
4	COPI	Internal	Controller Out Peripheral In
5	RESET	Internal	Reset
6	GND	Internal	Ground

SPI pinout

6.2.4 Digital Pins D22 - D53 LHS

Pin	Function	Type	Description
1	+5V	Power	+5V Power Rail
2	D22	Digital	GPIO 22
3	D24	Digital	GPIO 24
4	D26	Digital	GPIO 26
5	D28	Digital	GPIO 28
6	D30	Digital	GPIO 30
7	D32	Digital	GPIO 32
8	D34	Digital	GPIO 34
9	D36	Digital	GPIO 36
10	D38	Digital	GPIO 38
11	D40	Digital	GPIO 40
12	D42	Digital	GPIO 42
13	D44	Digital	GPIO 44
14	D46	Digital	GPIO 46
15	D48	Digital	GPIO 48
16	D50	Digital	GPIO 50
17	D52	Digital	GPIO 52
18	GND	Power	Ground

D22 - D53 LHS pinout

6.2.5 Digital Pins D22 - D53 RHS

Pin	Function	Type	Description
1	+5V	Power	+5V Power Rail
2	D23	Digital	GPIO 23
3	D25	Digital	GPIO 25
4	D27	Digital	GPIO 27
5	D29	Digital	GPIO 29
6	D31	Digital	GPIO 31
7	D33	Digital	GPIO 33
8	D35	Digital	GPIO 35
9	D37	Digital	GPIO 37
10	D39	Digital	GPIO 39
11	D41	Digital	GPIO 41
12	D43	Digital	GPIO 43
13	D45	Digital	GPIO 45
14	D47	Digital	GPIO 47
15	D49	Digital	GPIO 49
16	D51	Digital	GPIO 51
17	D53	Digital	GPIO 53
18	GND	Power	Ground

D22 - D53 RHS pinout

6.2.6 JTAG Pins

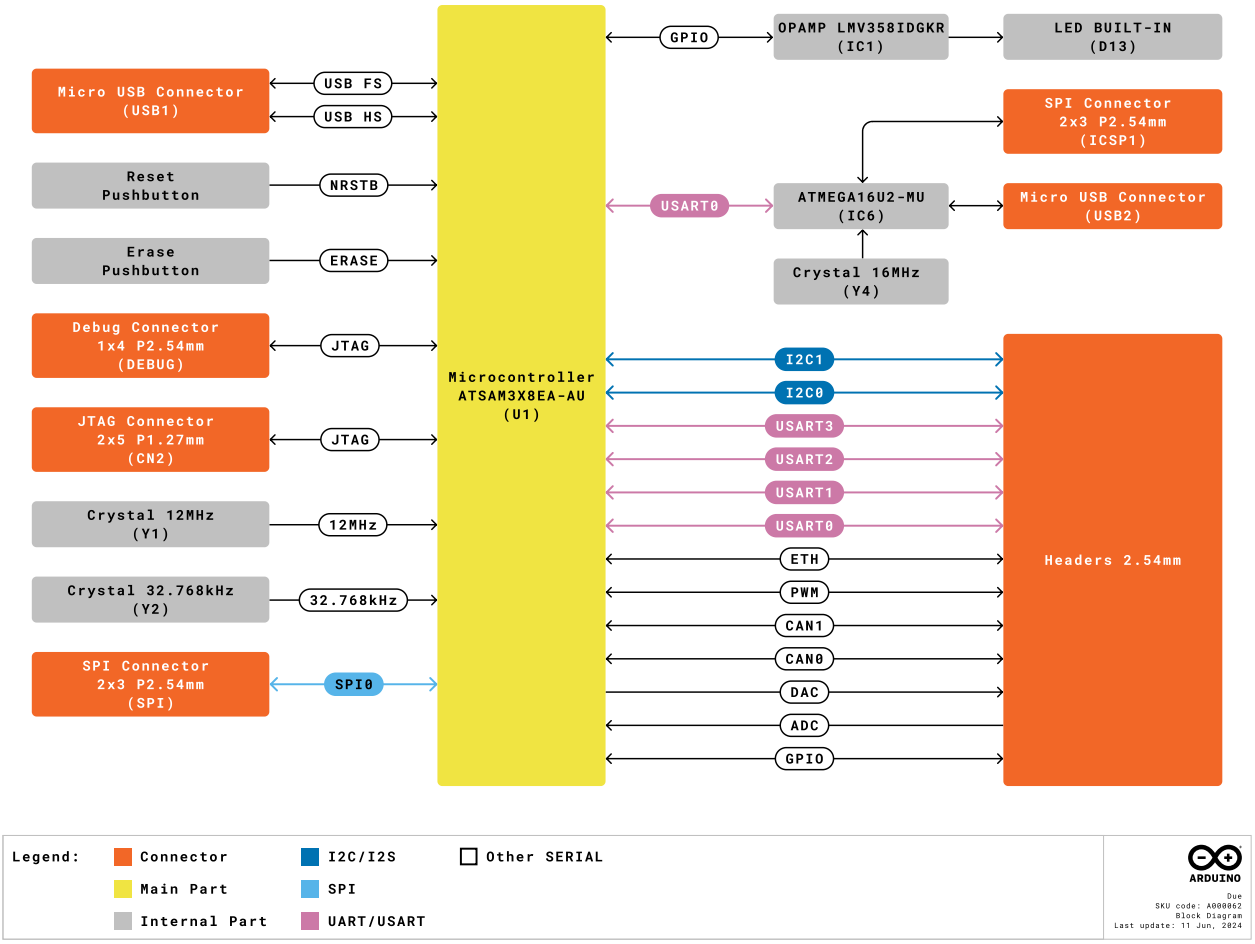
The board provides access to the debugging interface using the JTAG pins as it can be seen in the following table.

Pin	Function	Type	Description
1	Reset	Reset	Reset
2	GND	Power	GROUND
3	TDI	Digital	Test Data In
4	N/C	-	Not Connected
5	TDO	Digital	Test Data Out
6	GND	Power	GROUND
7	TCK	Digital	Test Clock
8	GND	Power	GROUND
9	TMS	Digital	Test Mode Select
10	+3V3	Power	+3V3 Power Rail

Debugging's JTAG pinout

6.3 Block Diagram

The block diagram with the main parts of the product can be checked in the following image:



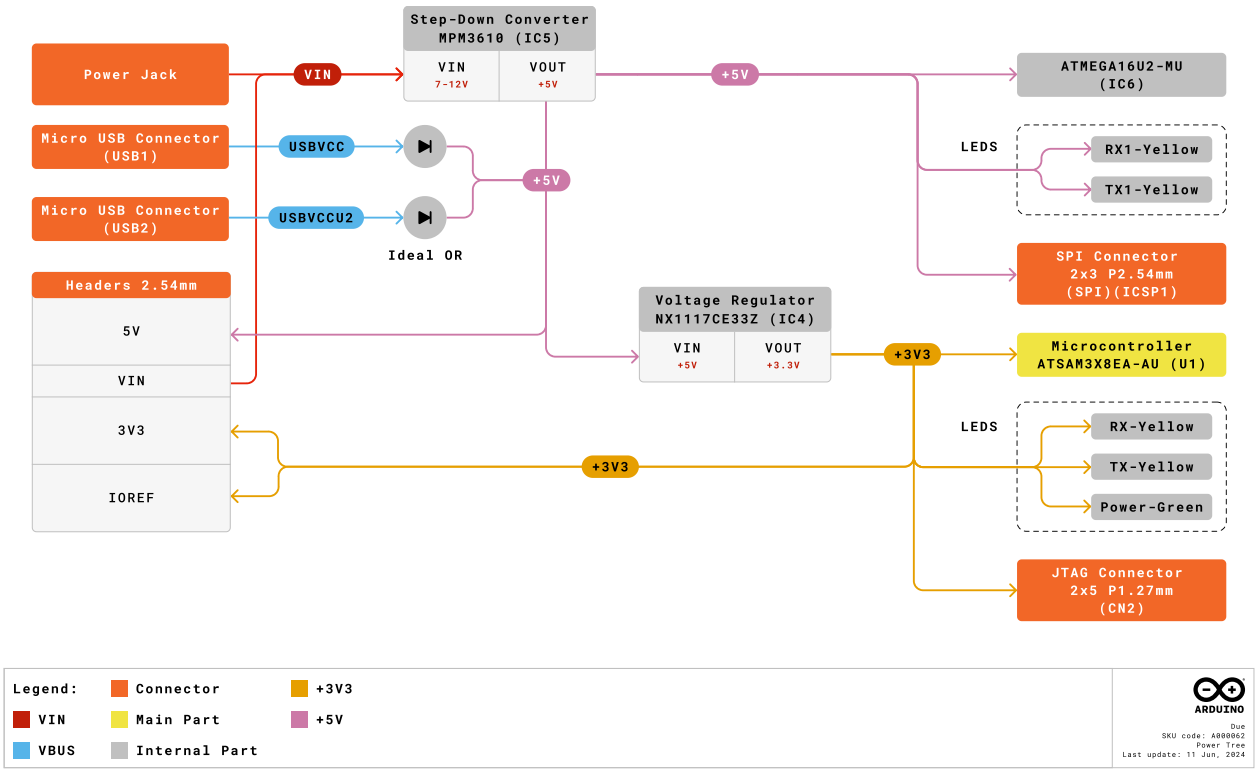
Arduino Due Block Diagram

6.4 Power Supply

The Arduino Due can be powered in multiple ways:

- USB Type-B port (Native port and Programming port).
- Using an external voltage source connected to VIN pin, which has a recommended voltage range of 7-12 VCC.
- The Power Jack: The Due can be powered using a DC power supply connected to the power jack, which accepts a voltage range of 7 to 12 V.

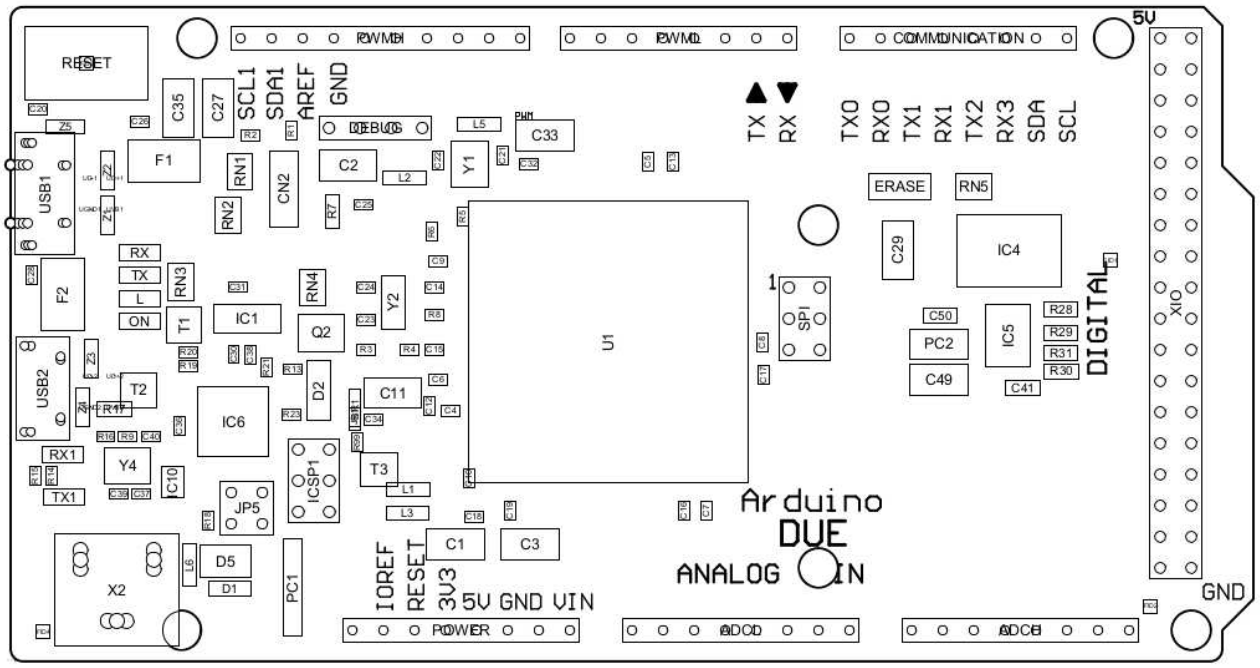
It is essential to note that the Arduino Due operates at 3.3 V, so any external power source must be regulated to this voltage level. Additionally, the power supply should be able to provide sufficient current for the board's operation and any connected peripherals.



Arduino Due Power Tree

6.5 Product Topology

In the following drawing you can see the main integrated circuits and passive components of the Arduino Due board.



Arduino Due Topology

Ref.	Description
U1	Atmel SAM3X8E ARM Cortex-M3
USB1	Native USB port
USB2	Programming USB port
X2	Power Jack VIN 7-12 VCC
ERASE	ERASE Button
RESET	Reset Button
DEBUG	Debug JTAG pinnout
SPI	SPI pinout
ICSP1	ICSPI1 Pinout

6.5.1 JTAG Connector

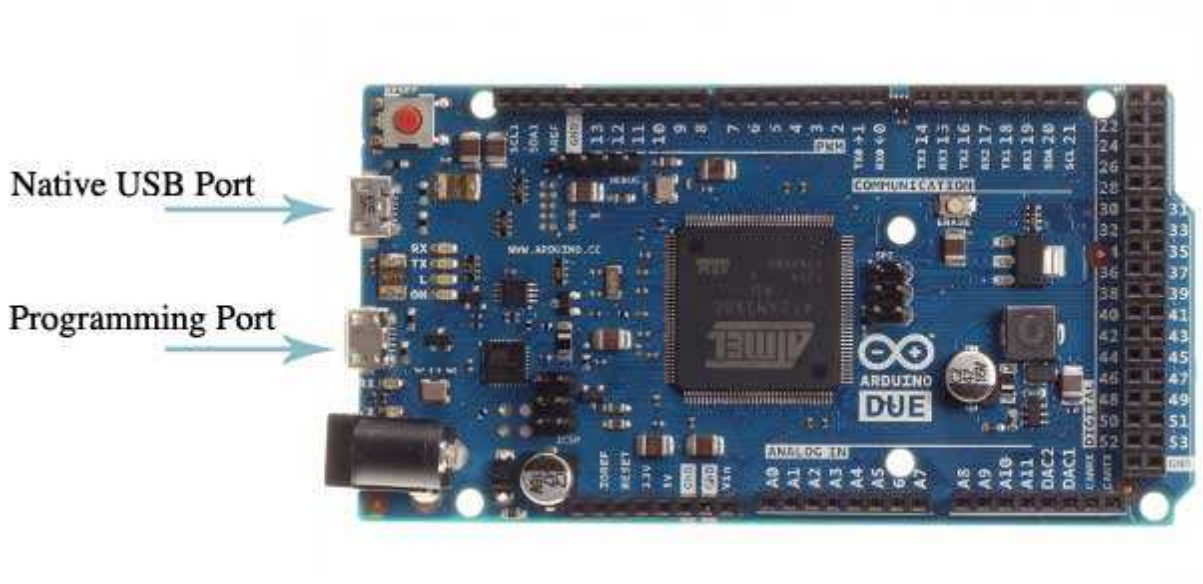
Debugging capabilities are integrated directly into the Arduino Due and are accessible via the 6-pin JTAG connector.

Pin	Function	Type	Description
1	Reset	Reset	Reset
2	GND	Power	GROUND
3	TDI	Digital	Test Data In
4	N/C	-	Not Connected
5	TDO	Digital	Test Data Out
6	GND	Power	GROUND
7	TCK	Digital	Test Clock
8	GND	Power	GROUND
9	TMS	Digital	Test Mode Select
10	+3V3	Power	+3V3 Power Rail

JTAG pinout

6.5.2 Native USB Port

The Arduino Due's Native USB port features a USB Type-B connector. This port allows the board to communicate directly with a computer as a USB device, enabling functionalities such as USB host/device capabilities and USB OTG (On-The-Go) functionality.



Arduino Due USB port

6.5.3 Programming USB Port

This port allows the board to be programmed and powered via a USB connection to a computer. It facilitates serial communication between the Arduino Due and the computer, enabling the uploading of sketches and interaction with the Arduino IDE. The port is connected to the ATmega16U2 microcontroller, which acts as a USB-to-serial converter, simplifying the programming process. When connected to a computer, the Arduino IDE recognizes the board as a COM port, enabling seamless communication for programming and debugging purposes.

6.5.4 Board's 24-Pin Header Connector

The 24-pin header connector provides a range of interfaces and general-purpose pins essential for various applications

These pins offer a range of functionalities, including analog and digital input/output, power supply connections, analog-to-digital, digital-to-analog conversion and CAN bus communication.

Pin	Function	Type	Description
1	NC	NC	Not Connected
2	IOREF	IOREF	Reference for digital logic voltage - connected to 3.3V
3	Reset	Reset	Reset
4	+3V3	Power	+3V3 Power Rail
5	+5V	Power	+5V Power Rail
6	GND	Power	Ground
7	GND	Power	Ground
8	VIN	Power	Voltage Input
9	A0	Analog	Analog input 0 / GPIO
10	A1	Analog	Analog input 1 / GPIO
11	A2	Analog	Analog input 2 / GPIO
12	A3	Analog	Analog input 3 / GPIO
13	A4	Analog	Analog input 4 / GPIO
14	A5	Analog	Analog input 5 / GPIO
15	A6	Analog	Analog input 6 / GPIO
16	A7	Analog	Analog input 7 / GPIO
17	A8	Analog	Analog input 8 / GPIO
18	A9	Analog	Analog input 9 / GPIO
19	A10	Analog	Analog input 10 / GPIO
20	A11	Analog	Analog input 11 / GPIO
21	DAC0	Analog	Digital to Analog Converter 0
22	DAC1	Analog	Digital to Analog Converter 1
23	CANRX	Digital	CAN Bus Receiver
24	CANTX	Digital	CAN Bus Transmitter

Board's 24-Pin Header pinout

6.5.5 Board's 26-Pin Header Connector

The 26-pin header connector on the Arduino Due offers a comprehensive set of interfaces and versatile pins crucial for diverse applications

These pins offer a range of functionalities, including digital input/output, serial communication, PWM (Pulse Width Modulation) outputs, and I2C (Inter-Integrated Circuit) communication.

Pin	Function	Type	Description
1	D21/SCL1	Digital	GPIO 21 / I2C 1 Clock
2	D20/SDA1	Digital	GPIO 20 / I2C 1 Dataline
3	AREF	Digital	Analog Reference Voltage
4	GND	Power	Ground
5	D13/SCK	Digital	GPIO 13 / SPI Clock (PWM~)
6	D12/CIPO	Digital	GPIO 12 / SPI Controller In Peripheral Out (PWM~)
7	D11/COPI	Digital	GPIO 11 / SPI Controller Out Peripheral In (PWM~)
8	D10/CS	Digital	GPIO 10 / SPI Chip Select (PWM~)
9	D9/SDA2	Digital	GPIO 9 / I2C 2 Dataline (PWM~)
10	D8/SCL2	Digital	GPIO 8 / I2C 2 Clockline (PWM~)
11	D7	Digital	GPIO 7 (PWM~)
12	D6	Digital	GPIO 6 (PWM~)
13	D5	Digital	GPIO 5 (PWM~)
14	D4	Digital	GPIO 4 (PWM~)
15	D3	Digital	GPIO 3 (PWM~)
16	D2	Digital	GPIO 2 (PWM~)
17	D1/TX0	Digital	GPIO 1 / Serial 0 Transmitter
18	D0/TX0	Digital	GPIO 0 / Serial 0 Receiver
19	D14/TX3	Digital	GPIO 14 / Serial 3 Transmitter
20	D15/RX3	Digital	GPIO 15 / Serial 3 Receiver
21	D16/TX2	Digital	GPIO 16 / Serial 2 Transmitter
22	D17/RX2	Digital	GPIO 17 / Serial 2 Receiver
23	D18/TX1	Digital	GPIO 18 / Serial 1 Transmitter
24	D19/RX1	Digital	GPIO 19 / Serial 1 Receiver
25	D20/SDA	Digital	GPIO 20 / I2C 0 Dataline
26	D21/SCL	Digital	GPIO 21 / I2C 0 Clock

Board's 26-Pin Header pinout

6.5.6 SPI

These pins facilitate communication between the Arduino Due and external SPI devices

Pin	Function	Type	Description
1	CIPO	Internal	Controller In Peripheral Out
2	+5V	Internal	Power Supply of 5 V
3	SCK	Internal	Serial Clock
4	COPI	Internal	Controller Out Peripheral In
5	RESET	Internal	Reset
6	GND	Internal	Ground

SPI pinout

6.5.7 D22 to D53 on Left and Right Side

These digital pins provide a wide range of GPIO (General Purpose Input/Output) capabilities for interfacing with external sensors, actuators, and other digital devices in Arduino Due projects.

Pin	Function	Type	Description
1	+5V	Power	+5V Power Rail
2	D22	Digital	GPIO 22
3	D24	Digital	GPIO 24
4	D26	Digital	GPIO 26
5	D28	Digital	GPIO 28
6	D30	Digital	GPIO 30
7	D32	Digital	GPIO 32
8	D34	Digital	GPIO 34
9	D36	Digital	GPIO 36
10	D38	Digital	GPIO 38
11	D40	Digital	GPIO 40
12	D42	Digital	GPIO 42
13	D44	Digital	GPIO 44
14	D46	Digital	GPIO 46
15	D48	Digital	GPIO 48
16	D50	Digital	GPIO 50
17	D52	Digital	GPIO 52
18	GND	Power	Ground

D22 - D53 LHS pinout

Pin	Function	Type	Description
1	+5V	Power	+5V Power Rail
2	D23	Digital	GPIO 23
3	D25	Digital	GPIO 25
4	D27	Digital	GPIO 27
5	D29	Digital	GPIO 29
6	D31	Digital	GPIO 31
7	D33	Digital	GPIO 33
8	D35	Digital	GPIO 35
9	D37	Digital	GPIO 37
10	D39	Digital	GPIO 39
11	D41	Digital	GPIO 41
12	D43	Digital	GPIO 43
13	D45	Digital	GPIO 45
14	D47	Digital	GPIO 47
15	D49	Digital	GPIO 49
16	D51	Digital	GPIO 51
17	D53	Digital	GPIO 53
18	GND	Power	Ground

D22 - D53 RHS pinout

7 Device Operation

7.1 Getting Started - IDE

If you want to program your Arduino Due while offline you need to install the Arduino® Desktop IDE **[1]**. To connect the Arduino Due to your computer, you will need a USB Type-B cable, which can also provide power to the board, as indicated by the LED (DL1).

7.2 Getting Started - Arduino Cloud Editor

All Arduino boards, including this one, work out-of-the-box on the Arduino® Cloud Editor **[2]**, by just installing a simple plugin.

The Arduino Cloud Editor is hosted online, therefore it will always be up-to-date with the latest features and support for all boards. Follow **[3]** to start coding on the browser and upload your sketches onto your board.

7.3 Getting Started - Arduino Cloud

All Arduino IoT enabled products are supported on Arduino Cloud which allows you to log, graph and analyze sensor data, trigger events, and automate your home or business.

7.4 Online Resources

Now that you have gone through the basics of what you can do with the board you can explore the endless possibilities it provides by checking exciting projects on ProjectHub **[4]**, the Arduino Library Reference **[5]**, and the online store **[6]**; where you will be able to complement your board with sensors, actuators and more.

7.5 Board Recovery

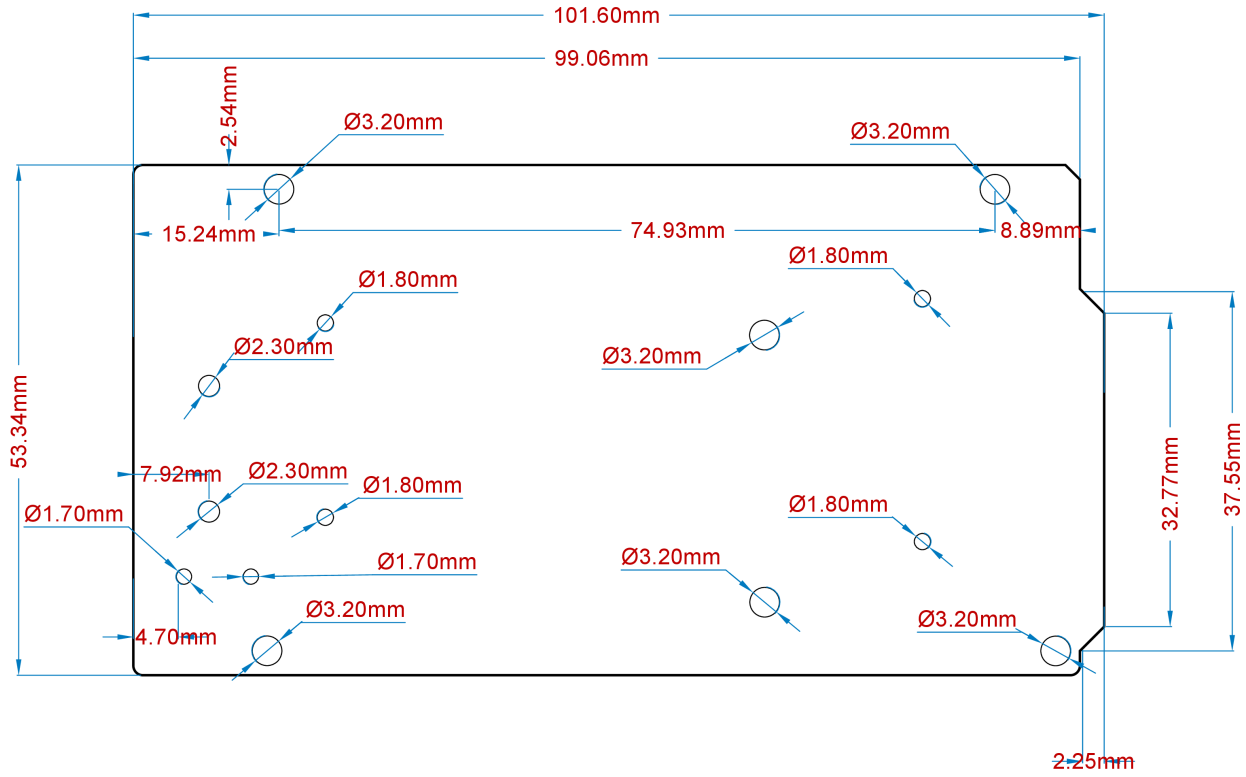
All Arduino boards have a built-in bootloader which allows flashing the board via USB. In case a sketch locks up the processor and the board is not reachable anymore via USB, it is possible to enter bootloader mode by double-tapping the reset button right after the power-up.

8 Mechanical Information

The Arduino Due is a microcontroller board measuring 101.52 mm x 53.3 mm, featuring two USB-B connectors and a big quantity of GPIO pins headers.

8.1 Board Dimensions

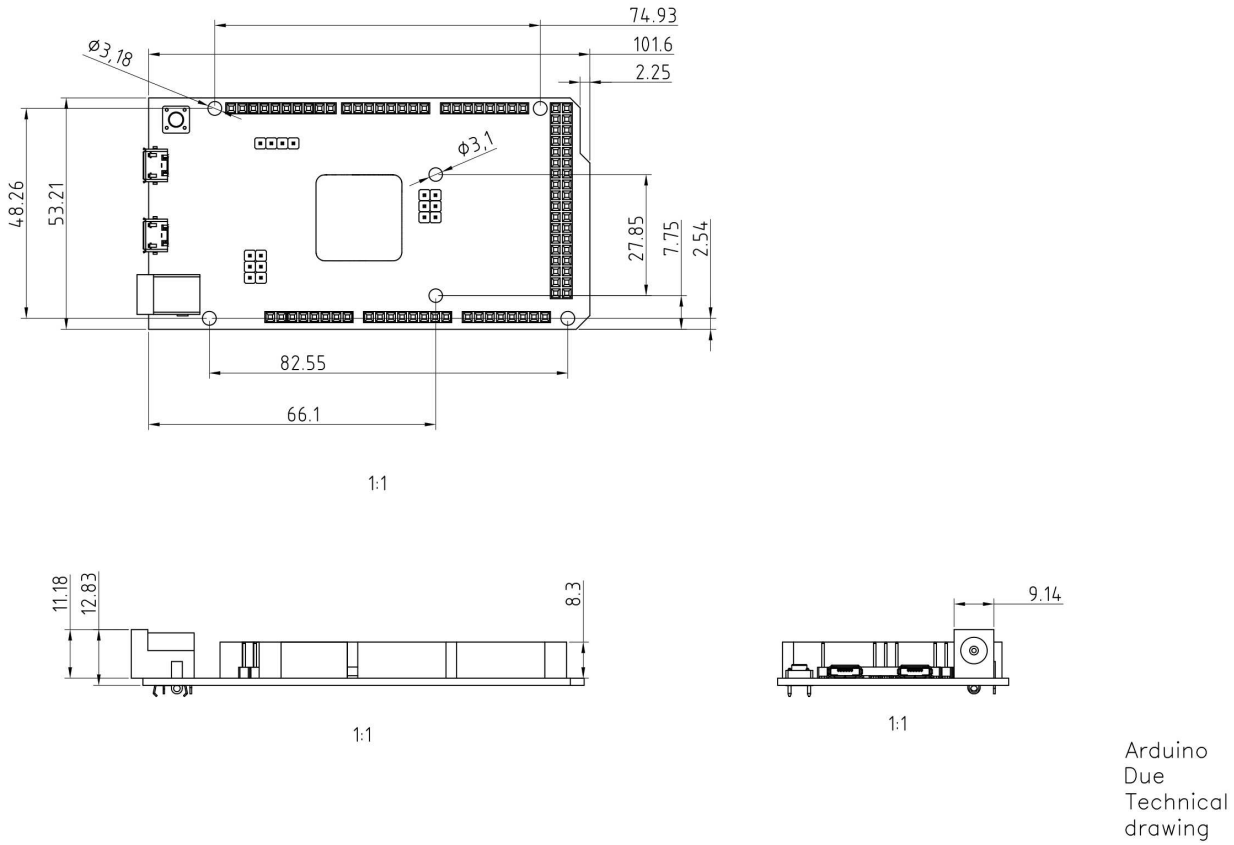
The Arduino Due board outline and mounting holes dimensions are shown in the figure below; all the dimensions are in mm.



Arduino Due Mounting Holes And Board Outline

8.2 Board Connectors

Connectors of the Arduino Due are placed on the left side of the board; their placement is shown in the figure below. All the dimensions are in mm.



Arduino Due Technical drawing

9 Certifications

9.1 Certifications Summary

Certification	Status
CE/RED (Europe)	Yes
UKCA (UK)	Yes
FCC (USA)	Yes
IC (Canada)	Yes
RCM (Australia)	Yes
RoHS	Yes
REACH	Yes
WEEE	Yes

9.2 Declaration of Conformity CE DoC (EU)

We declare under our sole responsibility that the products above are in conformity with the essential requirements of the following EU Directives and therefore qualify for free movement within markets comprising the European Union (EU) and European Economic Area (EEA).

9.3 Declaration of Conformity to EU RoHS & REACH 211 01/19/2021

Arduino boards are in compliance with RoHS 2 Directive 2011/65/EU of the European Parliament and RoHS 3 Directive 2015/863/EU of the Council of 4 June 2015 on the restriction of the use of certain hazardous substances in electrical and electronic equipment.

Substance	Maximum limit (ppm)
Lead (Pb)	1000
Cadmium (Cd)	100
Mercury (Hg)	1000
Hexavalent Chromium (Cr6+)	1000
Poly Brominated Biphenyls (PBB)	1000
Poly Brominated Diphenyl ethers (PBDE)	1000
Bis(2-Ethylhexyl) phthalate (DEHP)	1000
Benzyl butyl phthalate (BBP)	1000
Dibutyl phthalate (DBP)	1000
Diisobutyl phthalate (DIBP)	1000

Exemptions: No exemptions are claimed.

Arduino Boards are fully compliant with the related requirements of European Union Regulation (EC) 1907 /2006 concerning the Registration, Evaluation, Authorization and Restriction of Chemicals (REACH). We declare none of the SVHCs (https://echa.europa.eu/web/guest/candidate-list-table), the Candidate List of Substances of Very High Concern for authorization currently released by ECHA, is present in all products (and also package) in quantities totaling in a concentration equal or above 0.1%. To the best of our knowledge, we also declare that our products do not contain any of the substances listed on the "Authorization List" (Annex XIV of the REACH regulations) and

Substances of Very High Concern (SVHC) in any significant amounts as specified by the Annex XVII of Candidate list published by ECHA (European Chemical Agency) 1907 /2006/EC.

9.4 Conflict Minerals Declaration

As a global supplier of electronic and electrical components, Arduino is aware of our obligations with regard to laws and regulations regarding Conflict Minerals, specifically the Dodd-Frank Wall Street Reform and Consumer Protection Act, Section 1502. Arduino does not directly source or process conflict minerals such as Tin, Tantalum, Tungsten, or Gold. Conflict minerals are contained in our products in the form of solder or as a component in metal alloys. As part of our reasonable due diligence, Arduino has contacted component suppliers within our supply chain to verify their continued compliance with the regulations. Based on the information received thus far we declare that our products contain Conflict Minerals sourced from conflict-free areas.

9.5 FCC Caution

Any Changes or modifications not expressly approved by the party responsible for compliance could void the user's authority to operate the equipment.

This device complies with part 15 of the FCC Rules. Operation is subject to the following two conditions:

(1) This device may not cause harmful interference

(2) this device must accept any interference received, including interference that may cause undesired operation.

FCC RF Radiation Exposure Statement:

1. This Transmitter must not be co-located or operating in conjunction with any other antenna or transmitter.
2. This equipment complies with RF radiation exposure limits set forth for an uncontrolled environment.
3. This equipment should be installed and operated with a minimum distance of 20 cm between the radiator & your body.

Note: This equipment has been tested and found to comply with the limits for a Class B digital device, pursuant to part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference in a residential installation. This equipment generates, uses and can radiate radio frequency energy and, if not installed and used in accordance with the instructions, may cause harmful interference to radio communications. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does cause harmful interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures:

- Reorient or relocate the receiving antenna.
- Increase the separation between the equipment and receiver.
- Connect the equipment into an outlet on a circuit different from that to which the receiver is connected.
- Consult the dealer or an experienced radio/TV technician for help.

English: User manuals for license-exempt radio apparatus shall contain the following or equivalent notice in a conspicuous location in the user manual or alternatively on the device or both. This device complies with Industry Canada license-exempt RSS standard(s). Operation is subject to the following two conditions:

(1) this device may not cause interference

(2) this device must accept any interference, including interference that may cause undesired operation of the device.

French: Le présent appareil est conforme aux CNR d'Industrie Canada applicables aux appareils radio exempts de licence. L'exploitation est autorisée aux deux conditions suivantes:

(1) l' appareil nedoit pas produire de brouillage

(2) l'utilisateur de l'appareil doit accepter tout brouillage radioélectrique subi, même si le brouillage est susceptible d'en compromettre le fonctionnement.

IC SAR Warning:

English: This equipment should be installed and operated with a minimum distance of 20 cm between the radiator and your body.

French: Lors de l' installation et de l' exploitation de ce dispositif, la distance entre le radiateur et le corps est d 'au moins 20 cm.

Important: The operating temperature of the EUT can't exceed 85°C and shouldn't be lower than -40°C.

Hereby, Arduino S.r.l. declares that this product is in compliance with essential requirements and other relevant provisions of Directive 2014/53/EU. This product is allowed to be used in all EU member states.

Company Information

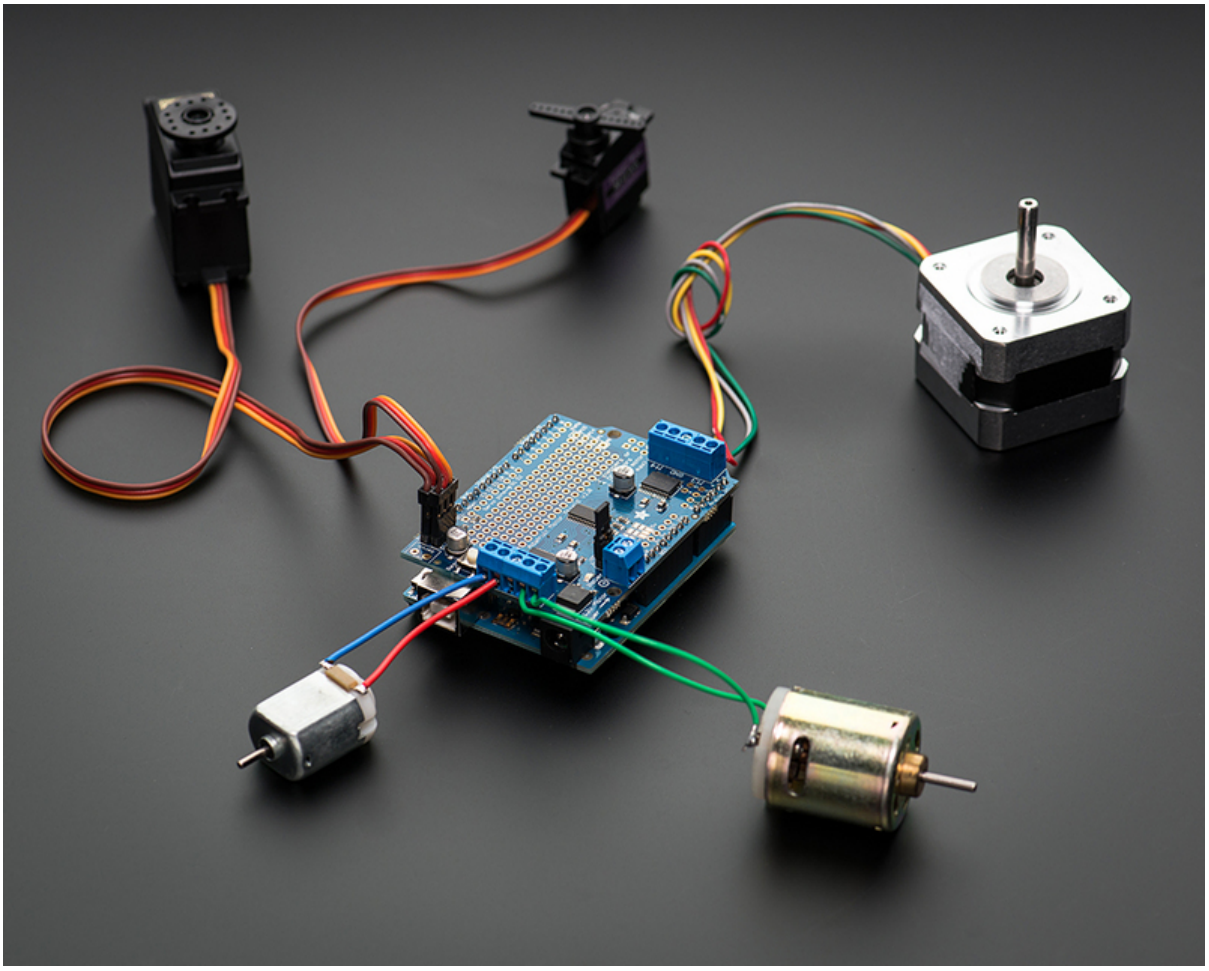
Company name	Arduino SRL
Company Address	Via Andrea Appiani, 25 - 20900 MONZA (Italy)

Reference Documentation

Ref	Link
Arduino IDE (Desktop)	https://www.arduino.cc/en/Main/Software
Arduino IDE (Cloud)	https://create.arduino.cc/editor
Cloud IDE Getting Started	https://docs.arduino.cc/cloud/web-editor/tutorials/getting-started/getting-started-web-editor
Project Hub	https://create.arduino.cc/projecthub?by=part&part_id=11332&sort=trending
Library Reference	https://github.com/arduino-libraries/
Online Store	https://store.arduino.cc/

Adafruit Motor Shield V2

Created by lady ada

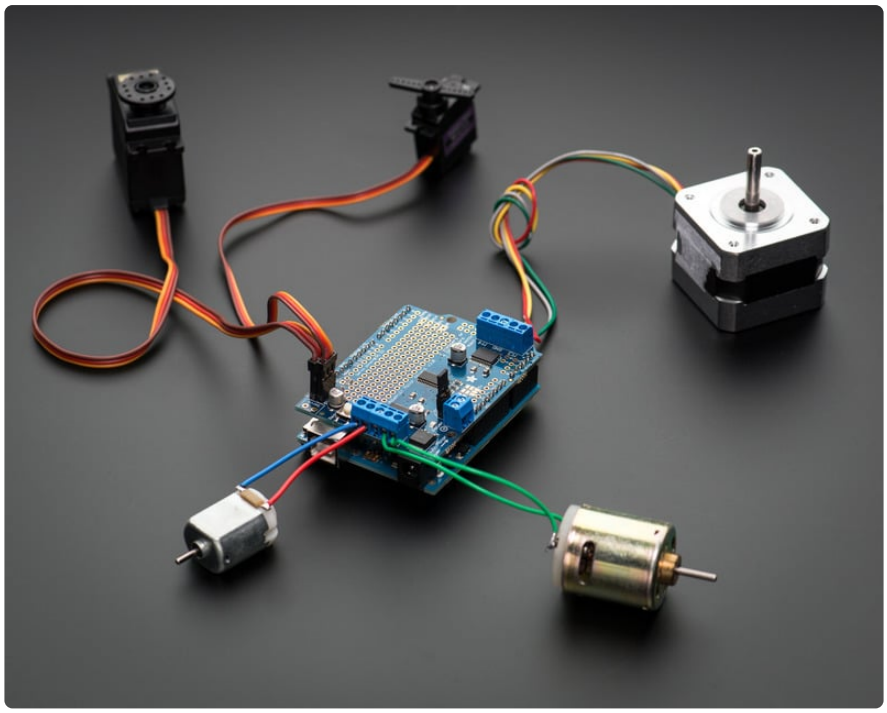


https://learn.adafruit.com/adafruit-motor-shield-v2-for-arduino

Last updated on 2024-08-05 01:26:26 PM EDT

Table of Contents

Overview



The original Adafruit Motorshield kit is one of our most beloved kits, which is why we decided to make something even better. We have upgraded the shield kit to make the bestest, easiest way to drive DC and Stepper motors. This shield will make quick work of your next robotics project! We kept the ability to drive up to 4 DC motors or 2 stepper motors, but added many improvements:

Instead of a L293D darlington driver, we now have the TB6612 MOSFET drivers with 1.2A per channel current capability (you can draw up to 3A peak for approx 20ms at a time). It also has much lower voltage drops across the motor so you get more torque out of your batteries, and there are built-in flyback diodes as well.

Instead of using a latch and the Arduino's PWM pins, we have a **fully-dedicated PWM driver chip** onboard. This chip handles all the motor and speed controls over I2C. Only two GPIO pins (SDA & SCL) plus 5v and GND are required to drive the multiple motors, and since it's I2C you can also connect any other I2C devices or shields to the same pins. This also makes it drop-in compatible with any Arduino, such as the Uno, Leonardo, Due and Mega R3.

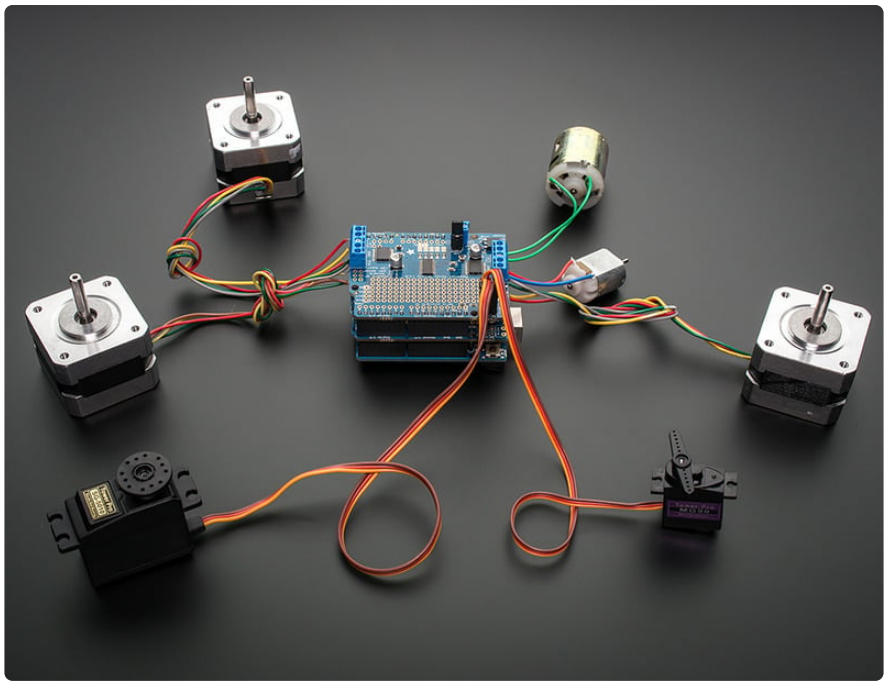
Motor control is through I2C bus communication with this board and not through PWM, serial, or other methods.

Completely stackable design: 5 address-select pins means up to 32 stackable shields: that's 64 steppers or 128 DC motors! What on earth could you do with that many steppers? I have no idea but if you come up with something send us a photo because that would be a pretty glorious project.

Lots of other little improvements such as a polarity protection FET on the power pins and a big prototyping area. And the shield is assembled and tested here at Adafruit so all you have to do is solder on straight or stacking headers and the terminal blocks.

Lets check out these specs again:

- **2 connections for 5V 'hobby' servos** connected to the Arduino's high-resolution dedicated timer - no jitter!
- 4 H-Bridges: TB6612 chipset provides **1.2A per bridge** (3A for brief 20ms peaks) with thermal shutdown protection, internal kickback protection diodes. Can run motors on 4.5VDC to 13.5VDC.
- **Up to 4 bi-directional DC** motors with individual 8-bit speed selection (so, about 0.5% resolution)
- **Up to 2 stepper motors** (unipolar or bipolar) with single coil, double coil, interleaved or micro-stepping.
- Motors automatically disabled on power-up
- Big terminal block connectors to easily hook up wires (18-26AWG) and power
- Arduino reset button brought up top
- Polarity protected 2-pin terminal block and jumper to connect external power, for separate logic/motor supplies
- Tested compatible with Arduino UNO, Leonardo, ADK/Mega R3, Diecimila & Duemilanove. Works with Due with 3.3v logic jumper. Works with Mega/ADK R2 and earlier with 2 wire jumpers.
- Download the easy-to-use Arduino software library, check out the examples and you're ready to go!
- **5v** or **3.3v** compatible logic levels - jumper configurable.



As of Arduino 1.5.6-r2 BETA, there is a bug in the Due Wire library that prevents multiple Motor Shields from working properly with the Due!

FAQ

How many motors can I use with this shield?

You can use 2 DC hobby servos that run on 5V and up to 4 DC motors or 2 stepper motors (or 1 stepper and up to 2 DC motors) that run on 5-12VDC

Can I connect more motors?

Yes, by stacking shields! Every shield you stack on will add 4 DC motors or 2 stepper motors (or 1 more stepper and 2 more DC motors).
You will not gain more servo connections as the servo contacts go to pin #9 and #10 on the Arduino.

What if I also need some more servos?

Check out our lovely servo shield, also stackable with this motor shield and adds 16 free-running servos per shield http://learn.adafruit.com/adafruit-16-channel-pwm-slash-servo-shield (https://adafru.it/ciQ)

What Arduinos is this shield compatible with?

It is tested to work with Duemilanove, Diecimila, Uno (all revisions), Leonardo and Mega/ADK R3 and higher.

It can work with Mega R2 and lower if you solder a jumper wire from the shield's SDA pin to Digital 20 and the SCL pin to Digital 21

To use a V3 shield with older processor boards that do not have an IOREF pin, you must configure the logic voltage level. Find the set of 3 pads labeled "Logic" and solder a bridge from the center pad to either the 5v or 3v pad on either side depending on the voltage of your board (e.g. 5v for UNO & Mega. 3v for Due).

As of Arduino 1.5.6-r2 BETA, there is a bug in the Due Wire library that prevents multiple Motor Shields from working properly!

I get the following error trying to run the example code: "error: Adafruit_MotorShield.h: No such file or directory...."

Make sure you have installed the Adafruit_MotorShield library

---

How do I install the library?

Check the tutorial page on the subject here http://learn.adafruit.com/adafruit-motor-shield-v2-for-arduino/install-software (https://adafru.it/ciO)

What stepper motors can I used with the shield?

The TB6612B driver chip are simple H-bridge drivers with a 1.2A continuous current limit. There is no active current limiting, so you need to choose a stepper motor that will not try to pull more than that. For a detailed explanation, see this guide (https://adafru.it/r2d)

A simple rule of thumb is to use a motor with a phase resistance of 10 ohms or more. This will be safe to use with supply voltages up to 12v.

The NEMA 17 motor we have in the shop has a phase resistance of about 35 ohms, so it is a good match for the shield.

Can I use this NEMA-17 motor?

NEMA-17 is just a motor frame-size designation. It tells us that the motor body is 1.7" square. It tells us nothing about the electrical characteristics. You will need to know at least the motor's phase resistance in order to determine compatibility.

See this guide for details: Matching the Driver to the Stepper (https://adafru.it/r2d)

HELP! My motor doesnt work! - HELP! My motor doesnt work!...But the servos work FINE!

Is the power LED lit? The Stepper and DC motor connections will not work if the onboard green Power LED is not lit brightly!
You must connect 5-12VDC power to the shield through the POWER terminal blocks or through the DC barrel jack on the Arduino and VIN jumper.

What is the green Power LED for?

The LED indicates the **DC/Stepper motor power supply is working**. If it is not lit brightly, then the DC/Stepper motors will not run. The servo ports are 5V powered and does not use the DC motor supply

---

What pins are/are not used on the motor shield?

GND and either 5v (default) or 3.3v are required to power the logic on-board. (5v or 3v operation is selectable via jumper)

The shield uses the SDA and SCL i2c pins to control DC and stepper motors. On the Arduino UNO these are also known as A4 and A5. On the Mega these are also known as Digital 20 and 21. On the Leonardo these are also known as digital 2 and 3. Do not use those pins on those Arduinos with this shield with anything other than an i2c sensor/driver.

Since the shield uses I2C to communicate, you can connect any other i2c sensor or driver to the SDA/SCL pins as long as they do not use address **0x60** (the default address of the shield) or **0x70** (the 'all call' address that this chip uses for group-control)

If you want to use the servo connections, they are on pins #9 and #10. If you do not use the connector then those pins are simply not used.

You can use any other pins for any other use

Note that pins A4 and A5 are connected to SDA and SCL for compatibility with classic Arduinos. These pins are not available for use on other processors.

How can I connect to the unused pins?

All pins are broken out into 0.1" spaced header along the edges of the shield

My Arduino freaks out when the motors are running! Is the shield broken?

Motors take a lot of power, and can cause 'brownouts' that reset the Arduino. For that reason the shield is designed for seperate (split) supplies - one for the electronics and one for the motor. Doing this will prevent brownouts. Please read the user manual for information about appropriate power supplies.

I'm trying to build this robot and it doesn't seem to run on a 9V battery....

You **cannot** power motors from a 9V battery. You must use AA batteries or a lead acid battery for motors.

Can this shield control small 3V motors?

Not really, its meant for larger, 5V+ motors. It does not work for 3V motors unless you overdrive them at 5V and then they will burn out faster

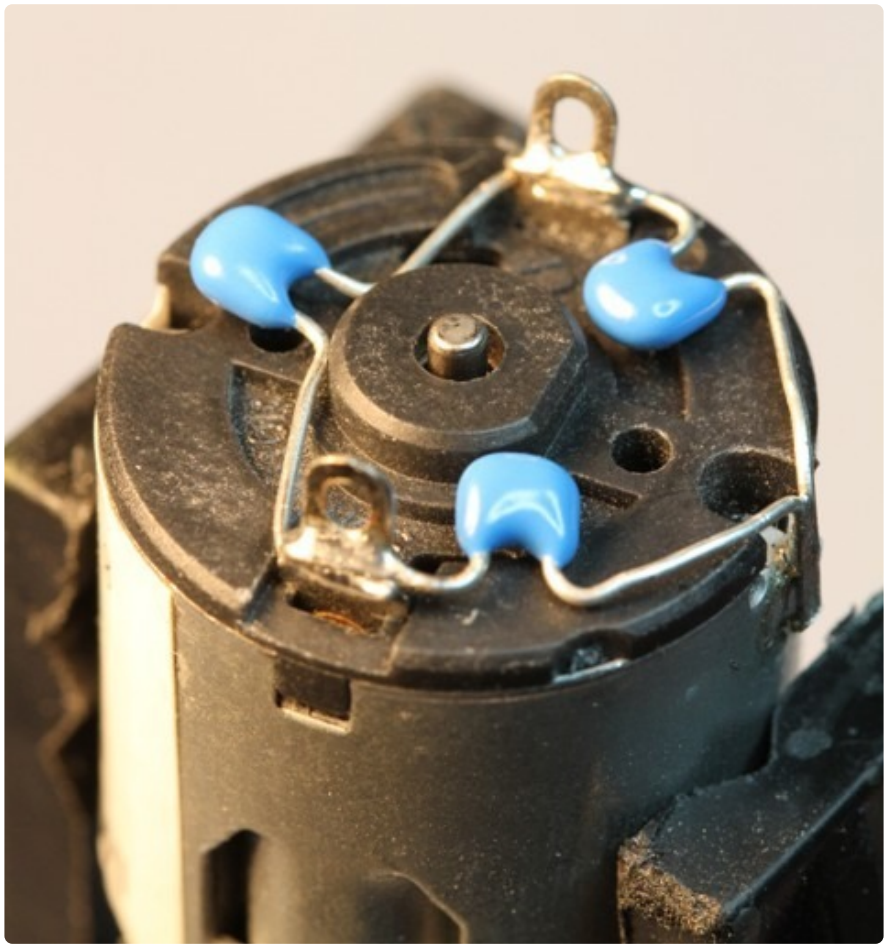
I have good solid power supplies, but the DC motors seem to 'cut out' or 'skip'.

Try soldering a ceramic or disc 0.1uF capacitor between the motor tabs (on the motor itself!) this will reduce noise that could be feeding back into the circuit (thanks macegr (https://adafru.it/clc)!)

When the motors start running nothing else works.

Many small DC motor have a lot of "brush noise". This feeds back into the Arduino circuitry and causes unstable operation. This problem can be solved by soldering some 0.1uF ceramic noise suppression capacitors to the motor.

You will need 3 total. 1 between the motor terminals, and one from each terminal to the motor casing.



But my motor already has a capacitor on it and it still doesn't work.

These motors generate a lot of brush noise and usually need the full 3-capacitor treatment for adequate suppression.

Why don't you just design capacitors into the shield?

They would not be effective there. The noise must be suppressed at the source or the motor leads will act like antennae and broadcast it to the rest of the system!

Why won't my stepper motor go any faster?

Since the shield is controlled by i2c, the maximum step rate is limited by the i2c bus speed. The default bus speed is 100KHz and can be increased to 400KHz by editing the library file in your Arduino installation folder. The file can be found in **hardware/libraries/wire/utility/twi.h**.

Find the line with: "#define TWI_FREQ 100000L"
and change it to "#define TWI_FREQ 400000L"

Or, you can add the following code to your setup() function: (Note: this line must be inserted after the call to begin())

TWBR = ((F_CPU /400000l) - 16) / 2; // Change the i2c clock to 400KHz

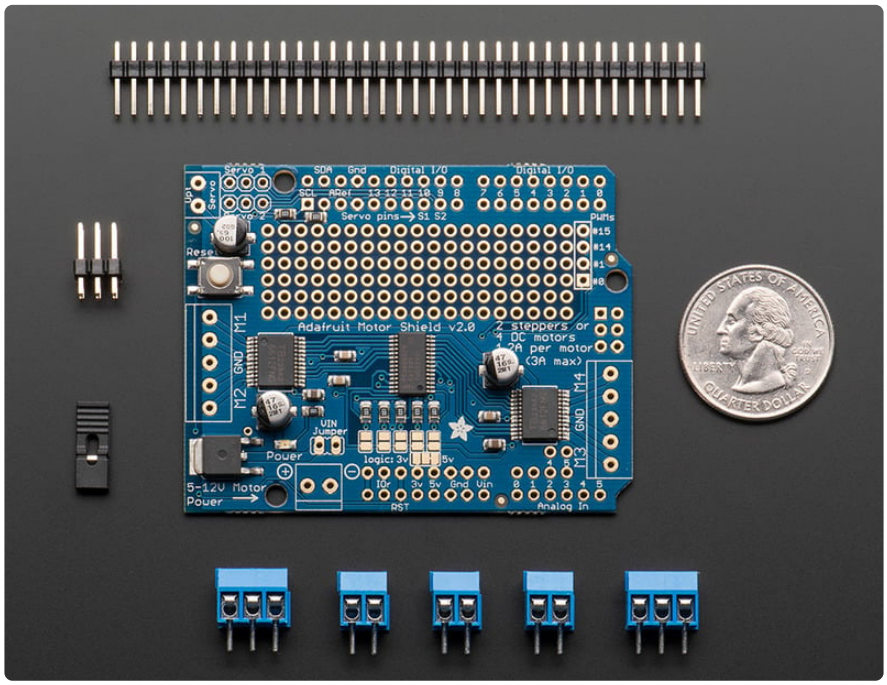
What I2C addresses are used by this shield?

The shield is addressable from 0x60-0x7F. 0x70 is an "all call" address that all boards will answer to.

My shield doesn't work with my LED backpack.

Some backpacks have a default address of 0x70. This is the "all call" address of the controller chip on the motor shield. If you re-address your backpack, it will work with the shield.

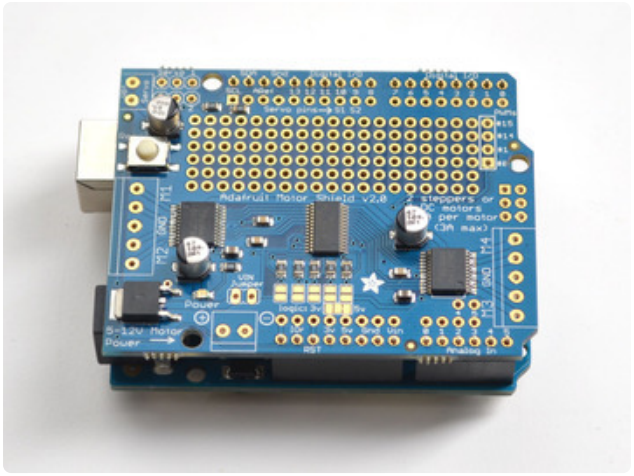
Install Headers & Terminals



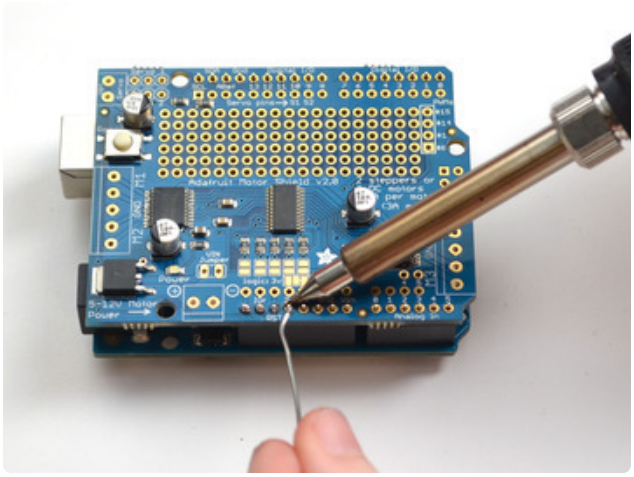
Installing Standard Headers

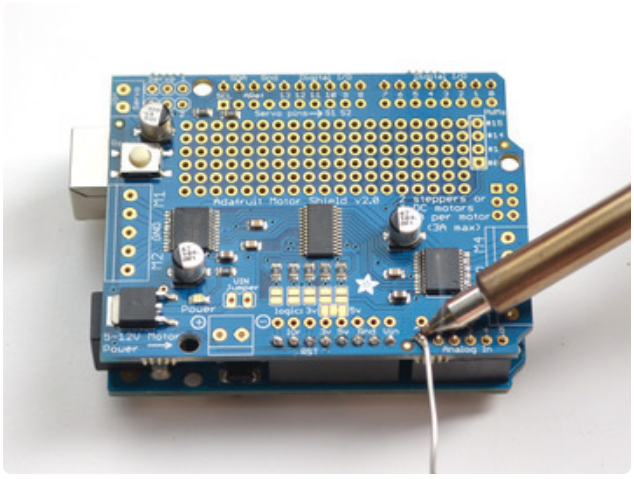
The shield comes with 0.1" standard header. Standard header does not permit stacking but it is mechanically stronger and they're much less expensive too! If you want to stack a shield on top, do not perform this step as it is not possible to uninstall the headers once soldered in! Skip down to the bottom for the stacking tutorial

Break apart the 0.1" header into 6, 8 and/or 10-pin long pieces and slip the long ends into the headers of your Arduino



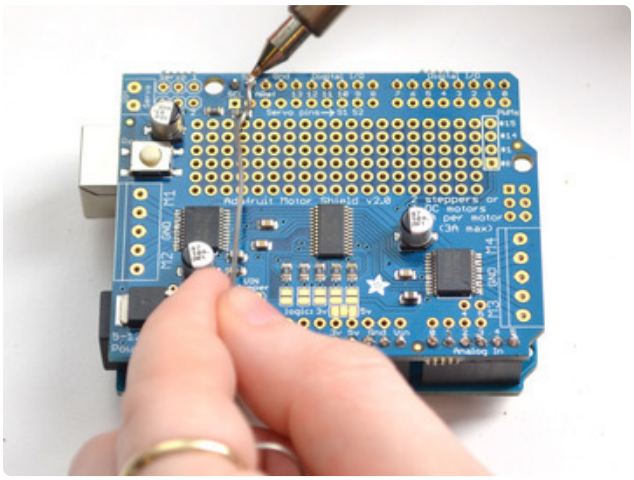
Place the assembled shield on top of the header-ed Arduino so that all of the short parts of the header are sticking through the outer set of pads

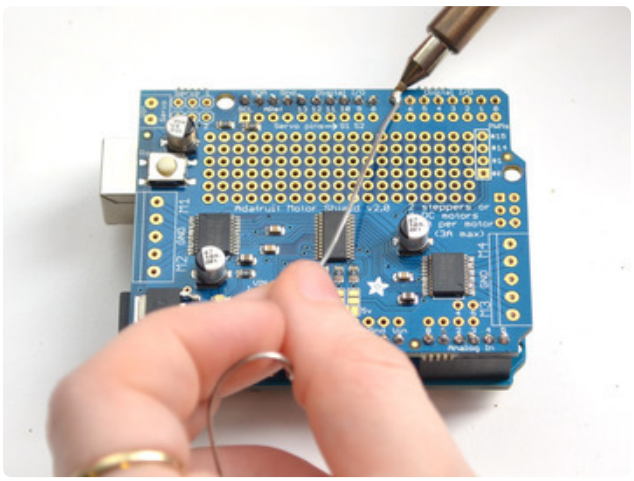


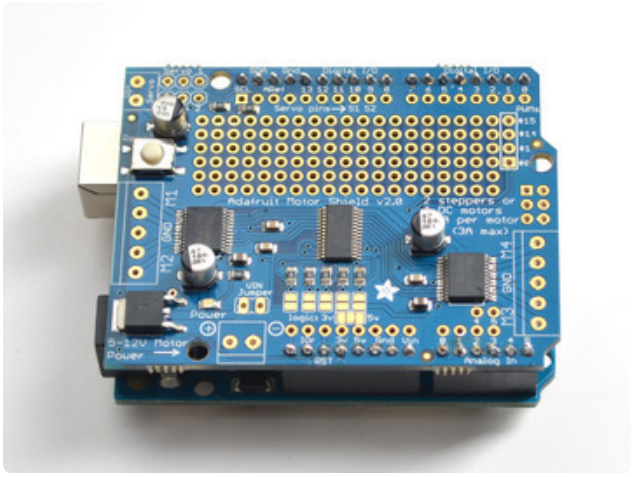


Solder each one of the pins into the shield to make a secure connection

Next, you will attach the terminal blocks, power jumper and servo connections



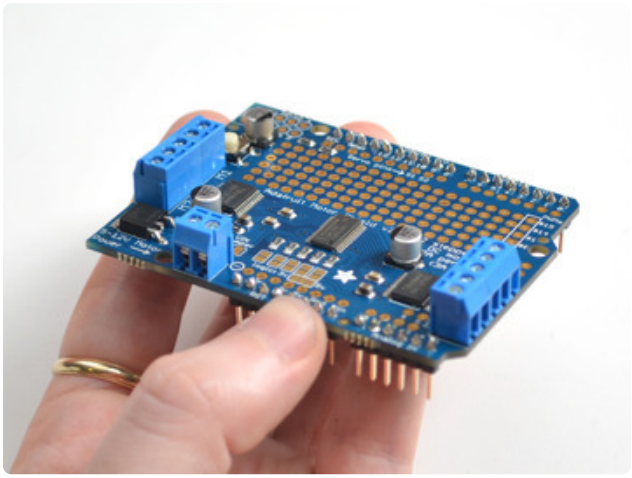




That's it! Now you can install the terminal blocks and jumper...

Installing Terminal Blocks and more

After you have installed either normal or stacking headers, you must install the terminal blocks.

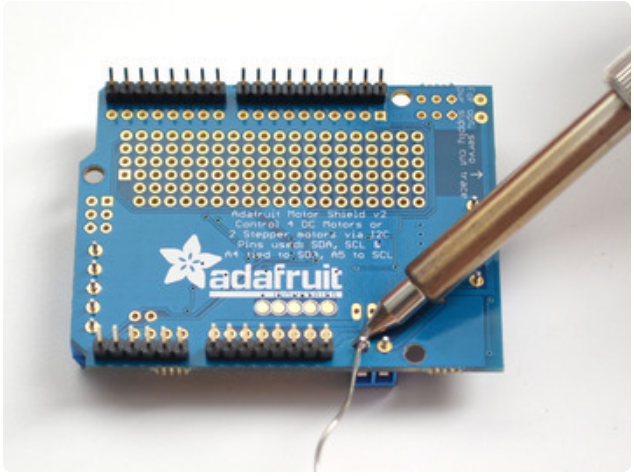


Next we will install the terminal blocks. These are how we will connect power and motors to the shield. They're much easier to use than soldering direct, just use a small screwdriver to release/attach wires!

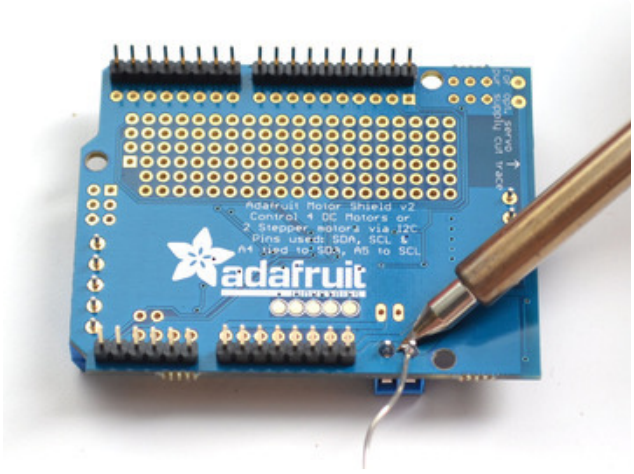
First, though, we must solder them in.

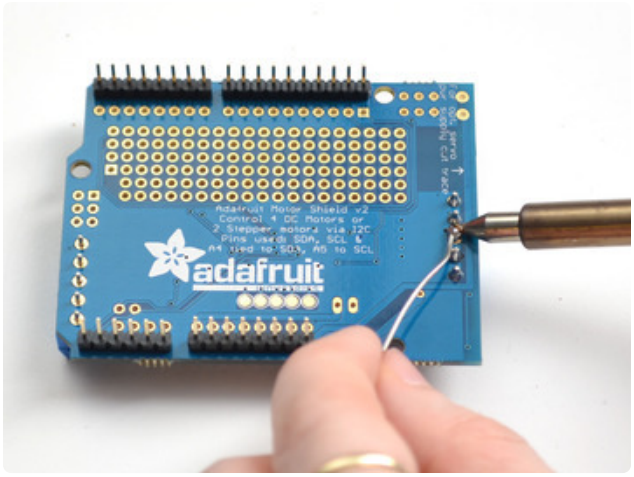
Slide the 3-pin terminal blocks into 2-pin terminal blocks so that you have 2 x 5-pin and 1 x 2-pin blocks. The two 5-pin sets go on either side. The 2-pin piece goes near the bottom of the shield. Make sure that the open holes of the terminal blocks face **out**!

Flip the board over so that you can see & solder the pins of the terminal blocks

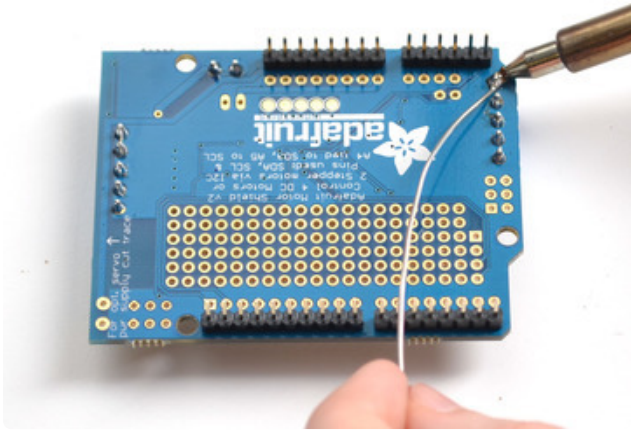


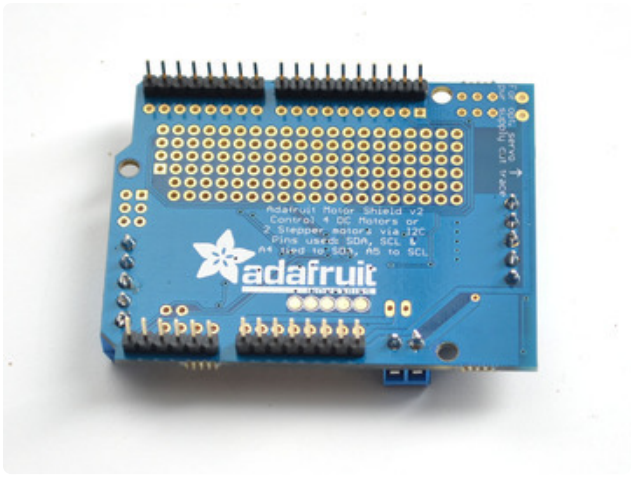
Solder in the two pins of the external power terminal-block



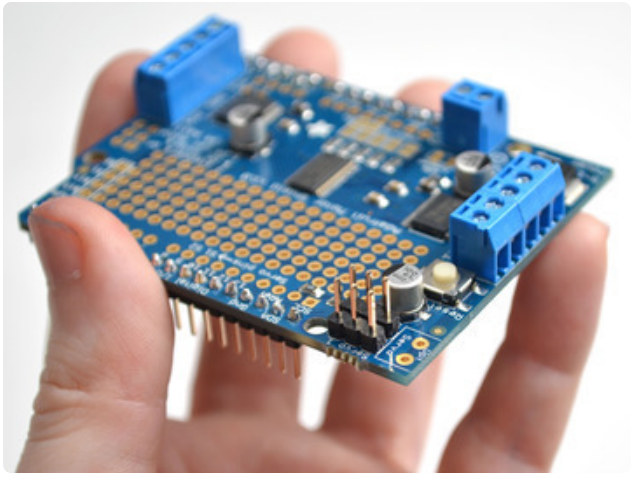


Solder in both motor blocks, 5 pads each



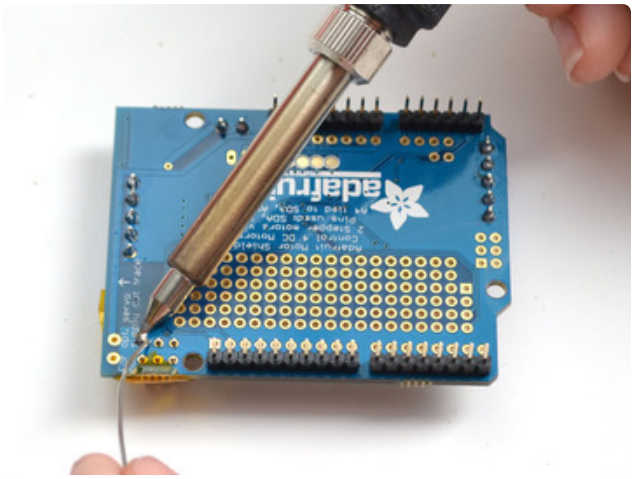


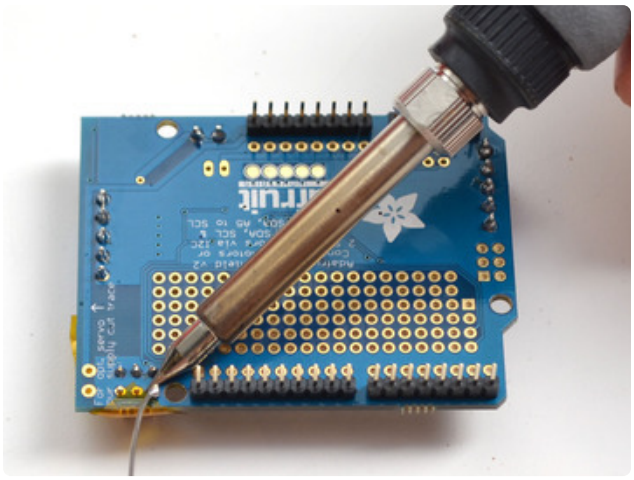
That's it for the terminal blocks. Next up, servo connections.



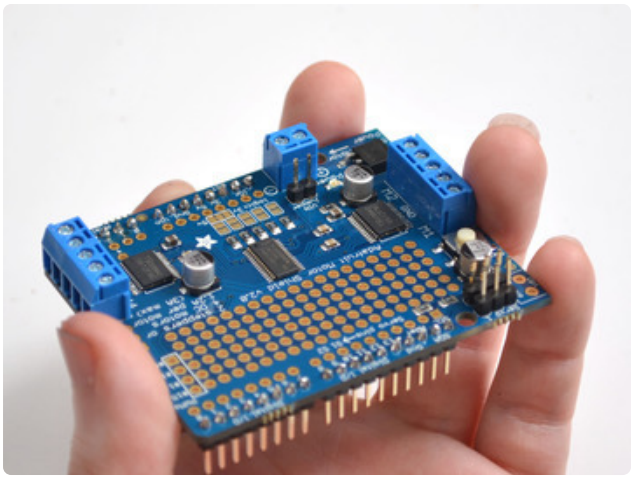
OK next up take the **2x3 pin header** and place it with the short legs down into the top corner where it says SERVO 1 and SERVO 2

You might have to sort of angle the part a little to get it to fit into both sets of 3-pin holes. we did this so it wont fall out easily when you turn it over!



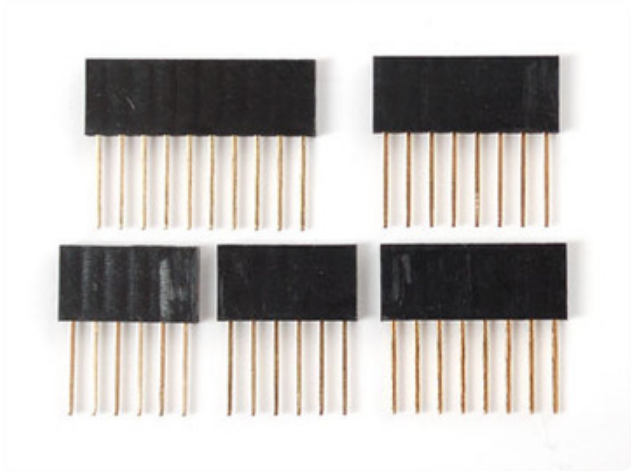


Then flip the board over and solder the 6 pins



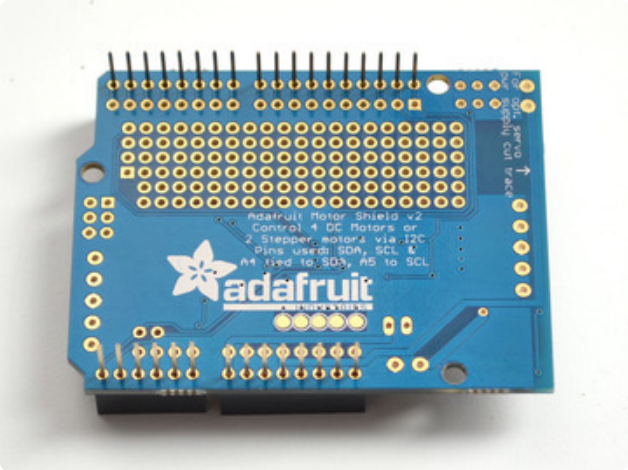
Finally, break off a 2-pin piece of header and place it next to the POWER terminal block, short legs down, tape it in place if necessary and solder it in.

Installing with Stacking Headers

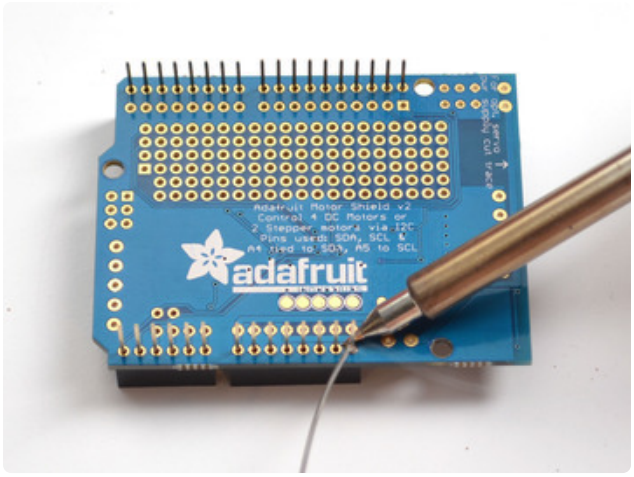


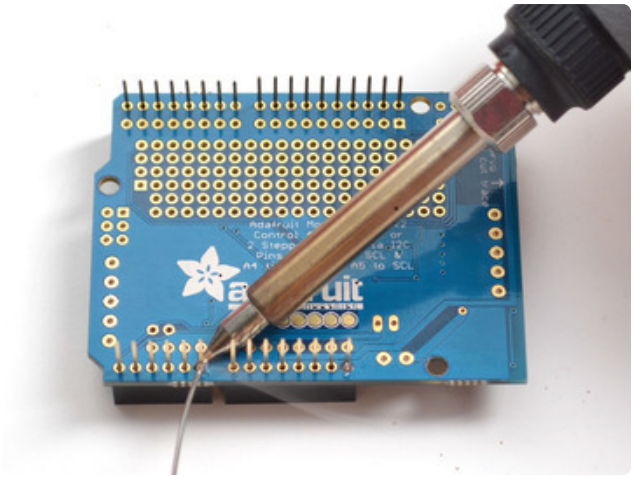
You will need to purchase Arduino stacking headers for this step, the shield does not come with them. (http://adafru.it/85)

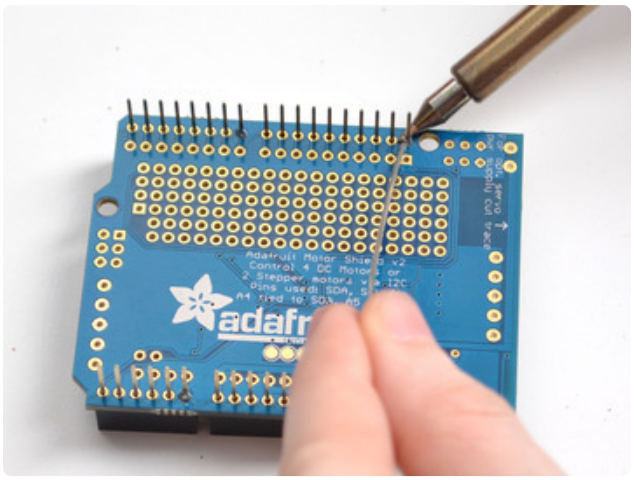
We don't show soldering in the 2x3 stacking header but you should solder that in as well - even though this shield does not use it, the one above may need those pins!



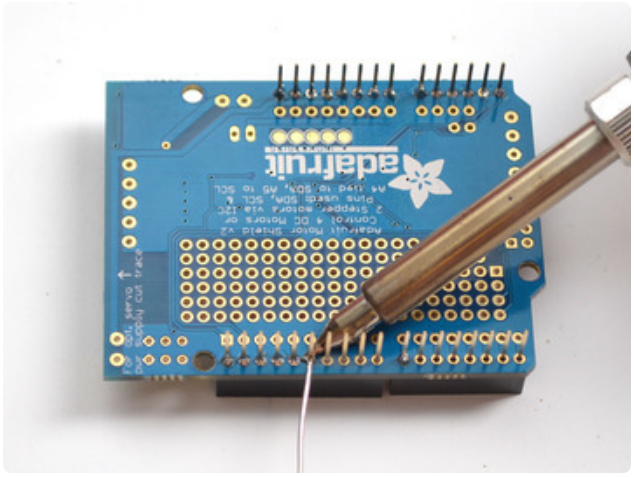
Start by sliding the 10 pin, 2 x 8 pin and 6-pin stacking headers into the outer rows of the shield from the top. Then flip the board over so its resting on the four headers. Pull on the legs if necessary to straighten them out.

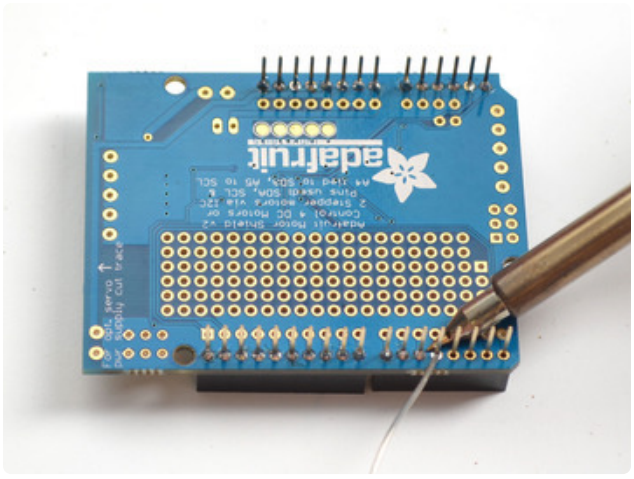






Tack one pin of each header, to get them set in place before more soldering. If the headers go crooked you can re-heat the one pin while re-positioning to straighten them up





Once you've tacked and straightened all the headers, go back and solder the remaining pins for each header.

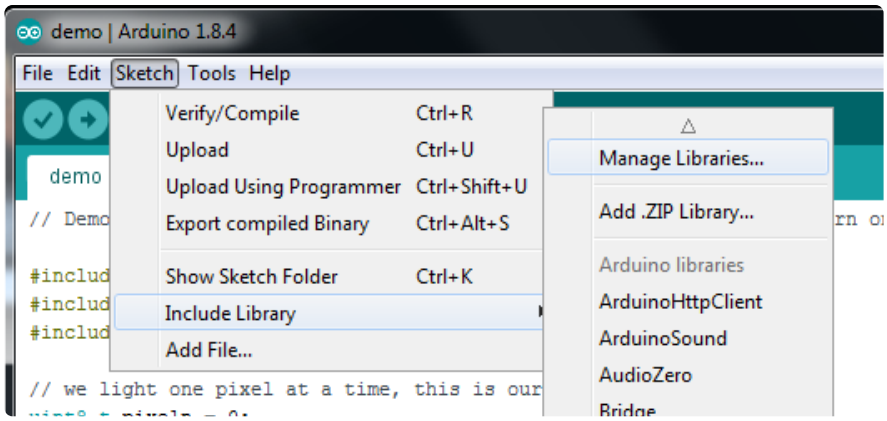
Install Software

Install Adafruit Motor Shield V2 library

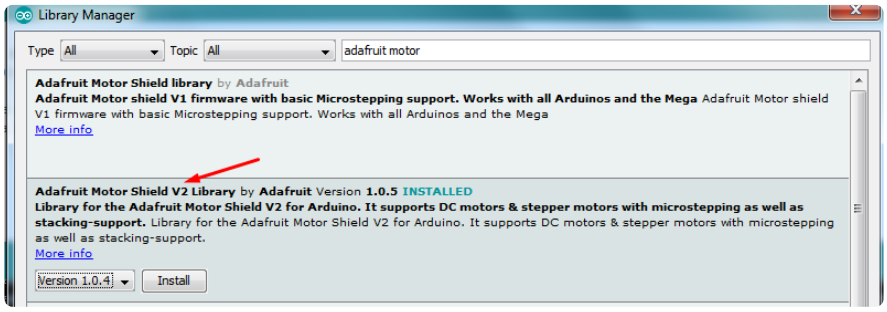
To use the shield on an Arduino, you'll need to install the Adafruit Motorshield v2 library. **This library is not compatible with the older AF_Motor library** used for v1 shields. However, if you have code for the older shield, adapting the code to use the new shield isn't difficult. We had to change the interface a little to support shield stacking, & we think its worth it!

To begin controlling motors, you will need to install the Adafruit_Motor_Shield_V2_Library library (code on our github repository) (https://adafru.it/ciN). It is available from the Arduino library manager so we recommend using that.

From the IDE open up the library manager...



And type in **adafruit motor** to locate the library. Click **Install**



If you plan to use AccelStepper for acceleration control or for simultaneous control of multiple stepper motors, you will also need to download and install the AccelStepper library:

AccelStepper Library

https://adafru.it/lpB

For more details on how to install Arduino libraries, check out our detailed tutorial! (https://adafru.it/aYM)

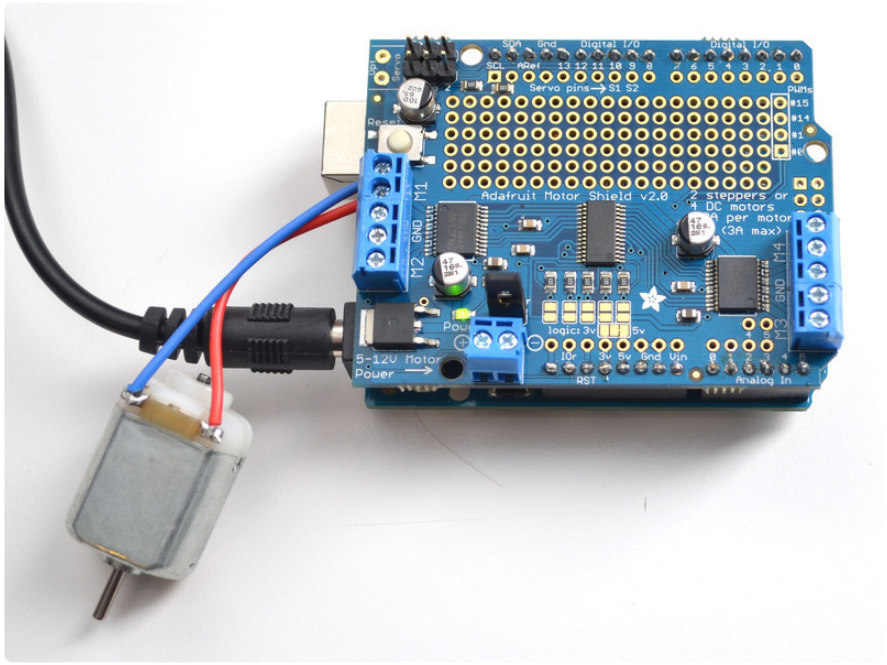
Running the Example Code

DC Motor

The library comes with a few examples to get you started up fast. We suggest getting started with the DC motor example. You can use any DC motor that can be powered by 6V-12VDC

First, restart the IDE to make sure the new library is loaded.

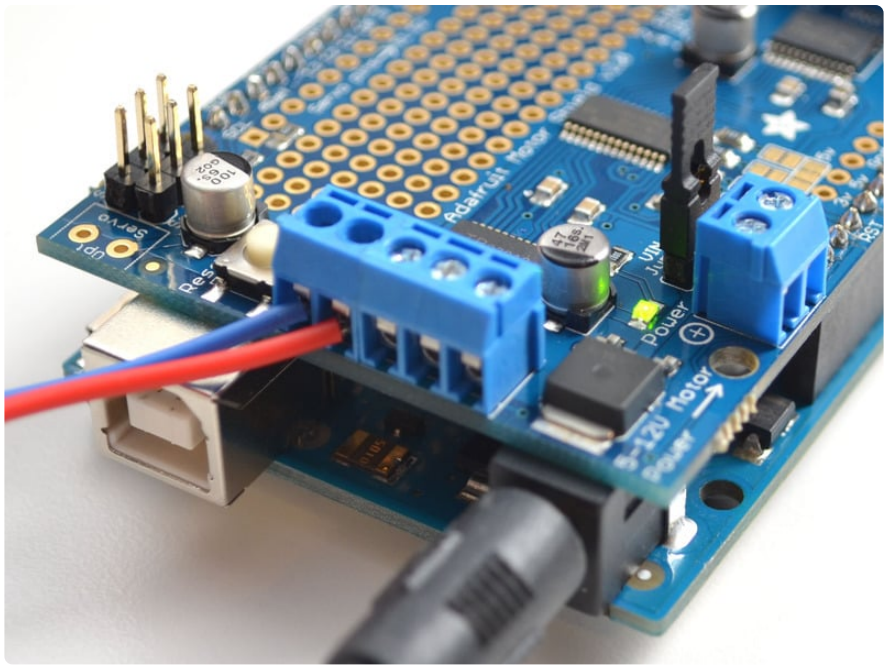
Plug the shield into the Arduino and connect a DC motor to **motor port 1** - it does not matter which wire goes into which terminal block as motors are bi-directional. Connect to the top two terminal ports, do not connect to the middle pin (GND) See the photo below for the red and blue wire example. Be sure to screw down the terminal blocks to make a good connection!



You must also supply 5-12VDC to power the motor. There are two ways to do this

1. You can power the Arduino via the **DC Barrel Jack** and **insert the VIN Jumper** shown as the tall black handle right next to the green Power LED below
2. You can power the Arduino via the DC Barrel jack **or** USB port. Then Power the shield via the 5-12VDC motor power terminal port, the double terminal block next to the green Power LED **and remove the VIN jumper**

If the Green LED next to the power terminal block isn't lit up brightly do not continue!



Once you have verified the motor is connected properly **and** you have the power LED lit up brightly, we can upload our code.

In the IDE, load **File->Examples->Adafruit_MotorShield->DCMotorTest**

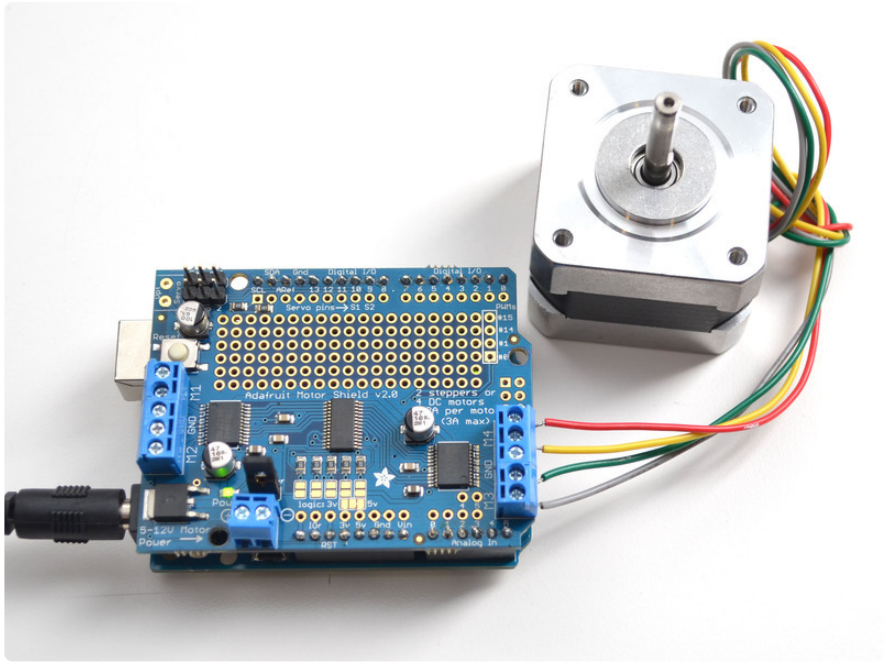
You should see and hear the DC motor turn on and move back and forth, attaching a slip of paper or tape as a 'flag' can help you visualize the movement if you have trouble seeing the movement

Stepper Motor Test

You can also test a stepper motor connection with the shield. The shield can run **unipolar** (5-wire and 6-wire) and **bipolar** (4-wire) steppers. It cannot run steppers with any other # of wires! The code is the same for unipolar or bipolar motors, the wiring is just slightly different.

Plug the shield into the Arduino and connect a stepper motor to **motor port 2** - unlike DC motors, the wire order does 'matter'. Connect to the top two terminal ports (coil #1) and the bottom two terminal ports (coil #2).

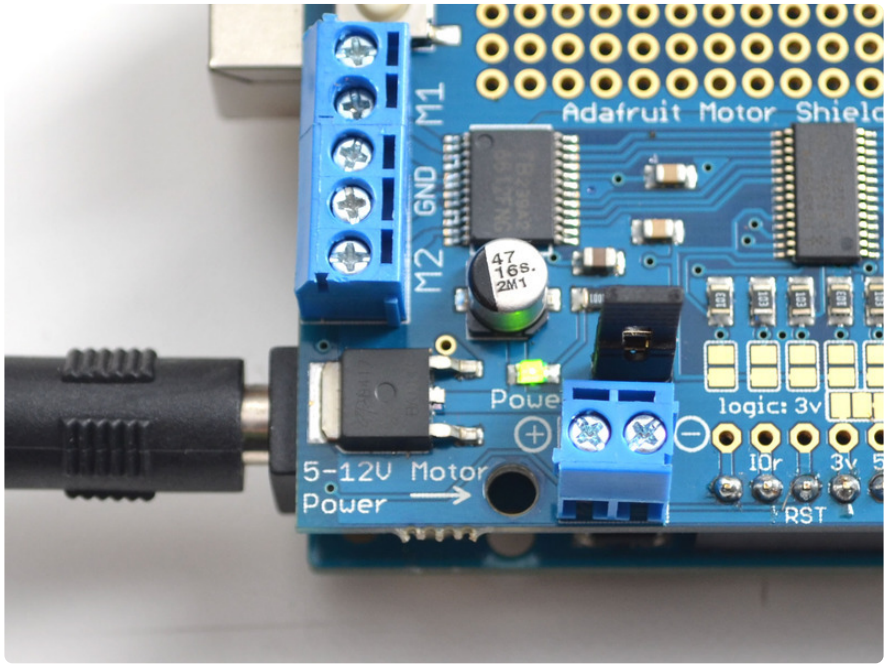
- If you have a bipolar motor, do not connect to the middle pin (GND).
- If you are using a unipolar motor with 5 wires, connect the common wire to GND.
- If you are using a unipolar motor with 6 wires, you can connect the two 'center coil wires' together to GND



You must also supply 5-12VDC to power the motor. There are two ways to do this

1. You can power the Arduino via the **DC Barrel Jack** and **insert the VIN Jumper** shown as the tall black handle right next to the green Power LED below
2. You can power the Arduino via the DC Barrel jack **or** USB port. Then Power the shield via the 5-12VDC motor power terminal port, the double terminal block next to the green Power LED **and remove the VIN jumper**

If the Green LED isn't lit up brightly **do not continue** - you must power it via the VIN jumper **or** the terminal block

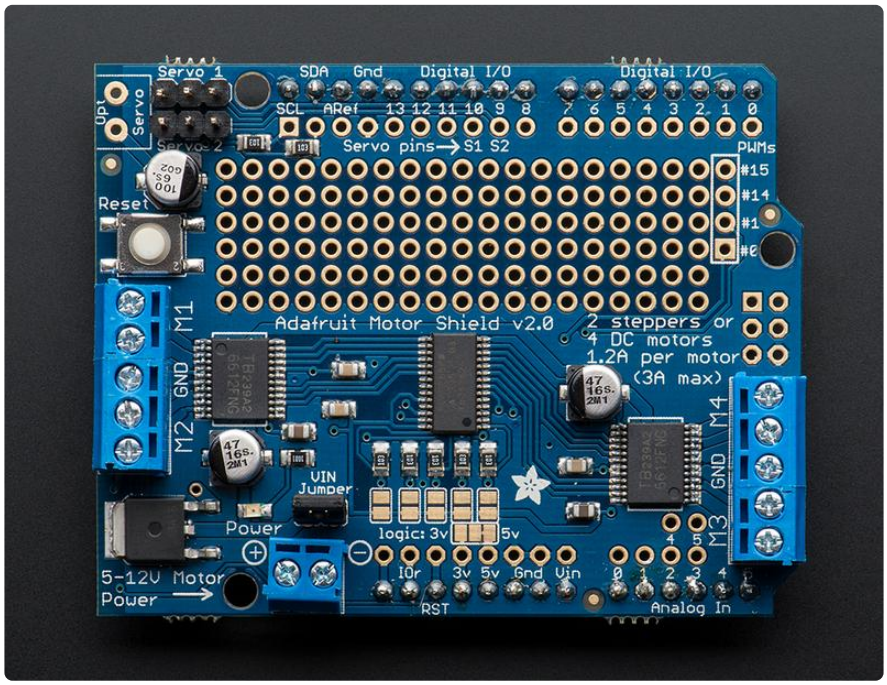


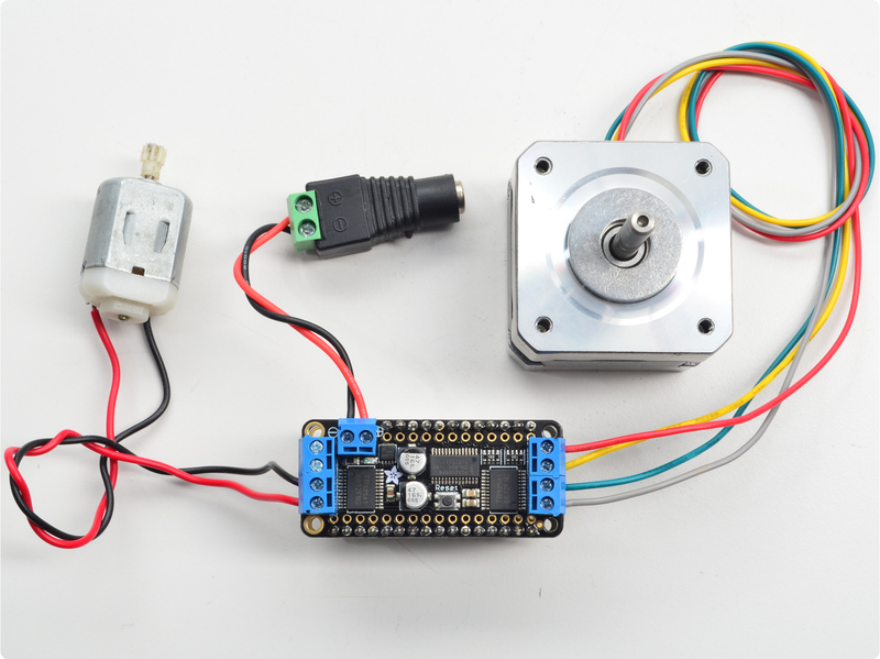
Once you have verified the motor is connected properly **and** you have the power LED lit up brightly, we can upload our code.

In the IDE, load **File->Examples->Adafruit_MotorShield->StepperTest**

You should see and hear the stepper motor turn on and move back and forth, attaching a slip of paper or tape as a 'flag' can help you visualize the movement if you have trouble seeing the movement. There are four ways to move a stepper, with varying speed, torque and smoothness tradeoffs. This example code will demonstrate all four.

Library Reference





class Adafruit_MotorShield;

The Adafruit_MotorShield class represents a motor shield and must be instantiated before any DCMotors or StepperMotors can be used. You will need to declare one Adafruit_MotorShield for each shield in your system.

Adafruit_MotorShield(uint8_t addr = 0x60);

The constructor takes one optional parameter to specify the i2c address of the shield. The default address of the constructor (0x60) matches the default address of the boards as shipped. If you have more than one shield in your system, each shield must have a unique address.

void begin(uint16_t freq = 1600);

begin() must be called in setup() to initialize the shield. An optional frequency parameter can be used to specify something other than the default maximum: 1.6KHz PWM frequency.

Adafruit_DCMotor *getMotor(uint8_t n);

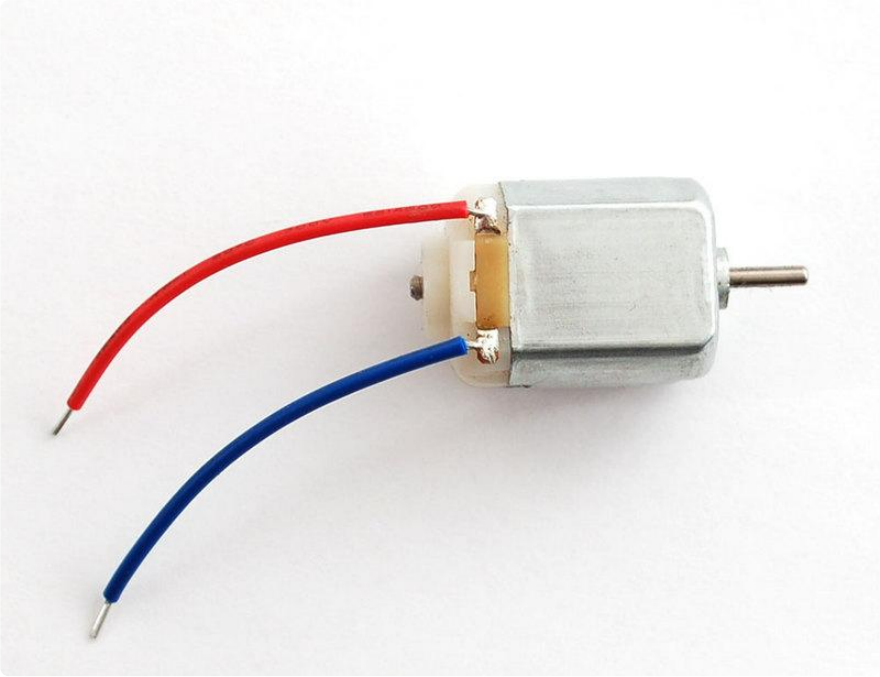
This function returns one of 4 pre-defined DC motor objects controlled by the shield. The parameter specifies the associated motor channel: 1-4.

Adafruit_StepperMotor *getStepper(uint16_t steps, uint8_t n);

This function returns one of 2 pre-defined stepper motor objects controlled by the shield.
The first parameter specifies the number of steps per revolution.
The second parameter specifies the associated stepper channel: 1-2.

void setPWM(uint8_t pin, uint16_t val);
void setPin(uint8_t pin, boolean val);

These are low-level functions to control pins on the on-board PWM driver chip. These functions are intended for internal use only.



class Adafruit_DCMotor

The Adafruit_DCMotor class represents a DC motor attached to the shield. You must declare an Adafruit_DCMotor for each motor in your system.

Adafruit_DCMotor(void);

The constructor takes no arguments. The motor object is typically initialized by assigning a motor object retrieved from the shield class as below:

```
// Create the motor shield object with the default I2C address
Adafruit_MotorShield AFMS = Adafruit_MotorShield();

// Select which 'port' M1, M2, M3 or M4. In this case, M1
```

```
Adafruit_DCMotor *myMotor = AFMS.getMotor(1);
// You can also make another motor on port M2
Adafruit_DCMotor *myOtherMotor = AFMS.getMotor(2);
```

void run(uint8_t);

The run() function controls the motor state. The parameter can have one of 3 values:

- **FORWARD** - Rotate in a forward direction
- **BACKWARD** - Rotate in the reverse direction
- **RELEASE** - Stop rotation

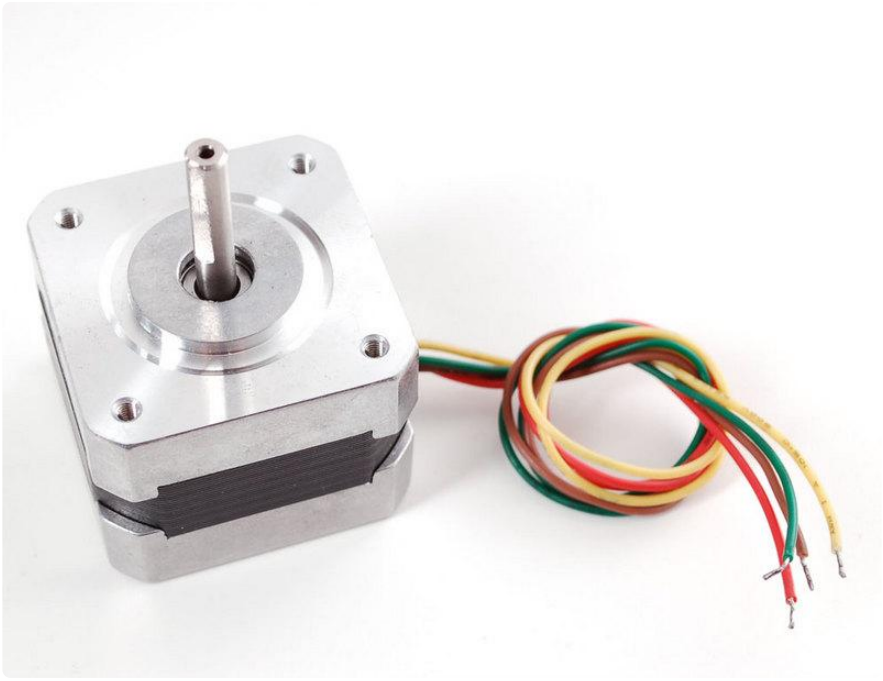
Note that the "FORWARD" and "BACKWARD" directions are arbitrary. If they do not match the actual direction of your vehicle or robot, simple swap the motor leads.

Also note that "RELEASE" simply cuts power to the motor. It does not apply any braking.

void setSpeed(uint8_t);

The setSpeed() function controls the power level delivered to the motor. The speed parameter is a value between 0 and 255.

Note that setSpeed just controls the power delivered to the motor. The actual speed of the motor will depend on several factors, including: The motor, the power supply and the load.



class Adafruit_StepperMotor

The Adafruit_StepperMotor class represents a stepper motor attached to the shield. You must declare an Adafruit_StepperMotor for each stepper motor in your system.

Adafruit_StepperMotor(void);

The constructor takes no arguments. The stepper motor is typically initialized by assigning a stepper object retrieved from the shield as below:

```
// Create the motor shield object with the default I2C address
Adafruit_MotorShield AFMS = Adafruit_MotorShield();

// Connect a stepper motor with 200 steps per revolution (1.8 degree)
// to motor port #2 (M3 and M4)
Adafruit_StepperMotor *myMotor = AFMS.getStepper(200, 2);
```

void step(uint16_t steps, uint8_t dir, uint8_t style = SINGLE);

The step() function controls stepper motion.

- The first parameter specifies how many steps to move.
- The second parameter specifies the direction: FORWARD or BACKWARD
- The last parameter specifies the stepping style: SINGLE, DOUBLE, INTERLEAVED or MICROSTEP

The ste() function is synchronous and does not return until all steps are complete. When complete the motor remains powered to apply "holding torque" to maintain

position.

void setSpeed(uint16_t);

The setSpeed() function controls the speed of the stepper motor rotation. Speed is specified in RPM.

uint8_t onestep(uint8_t dir, uint8_t style);

The oneStep() function is a low-level internal function called by step(). But it can be useful to call on its own to implement more advanced functions such as acceleration or coordinating simultaneous movement of multiple stepper motors. The direction and style parameters are the same as for step(), but onestep() steps exactly once.

Note: Calling step() with a step count of 1 is not the same as calling onestep(). The step function has a delay based on the speed set in setSpeed(). onestep() has no delay.

void release(void);

The release() function removes all power from the motor. Call this function to reduce power requirements if holding torque is not required to maintain position.

Arduino Library Docs

Arduino Library Docs (https://adafru.it/AvQ)

Powering Motors

Motors need a lot of energy, especially cheap motors since they're less efficient.

Voltage requirements:

The first important thing to figure out what voltage the motor is going to use. If you're lucky your motor came with some sort of specifications. Some small hobby motors are only intended to run at 1.5V, but its just as common to have 6-12V motors. The motor controllers on this shield are designed to run from **5V to 12V**.

MOST 1.5-3V MOTORS WILL NOT WORK

Current requirements:

The second thing to figure out is how much current your motor will need. The motor driver chips that come with the kit are designed to provide up to 1.2 A per motor, with 3A peak current. Note that the 'peak' rating is for very brief peaks such as during startup. Peak current levels can only be tolerated for a few milliseconds. If you will be pushing the 1.2A continuous limit you'll probably want to put a heat-sink on the motor driver, otherwise you will get thermal failure, possibly burning out the chip.

You can't run motors off of a 9V battery so don't waste your time/batteries!

Use a big Lead Acid or NiMH battery pack. Its also very much suggested that you set up two power supplies (split supply) one for the Arduino and one for the motors. **99% of 'weird motor problems**' are due to noise on the power line from sharing power supplies and/or not having a powerful enough supply! Even small DC motors can draw up to 3 Amps when they stall.

Setting up your shield for powering Hobby Servos

Servos are powered off of the same regulated 5V that the Arduino uses. This is OK for the small hobby servos suggested. Basically, power up your Arduino with the USB port or DC barrel jack and you're good to go. If you want something beefier, cut the trace going to the optional servo power terminal and wire up your own 5-6V supply!

Setting up your shield for powering DC and Stepper Motors

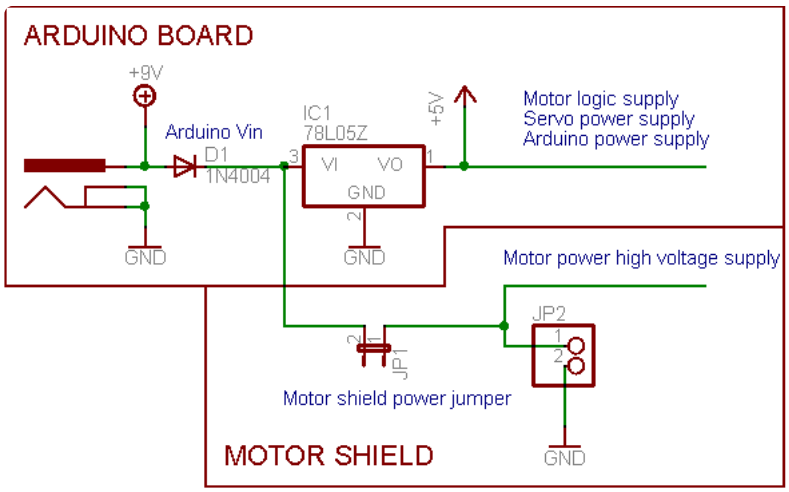
The motors are powered off of a 'high voltage supply' and NOT the regulated 5V. **Don't connect the motor power supply to the Arduino's 5V power pin**. This is a very very very bad idea unless you are sure you know what you're doing! You could damage your Arduino and/or USB port!

There are two places you can get your motor 'high voltage supply' from.

1. One is the DC barrel jack on the Arduino board
2. The other is the 2-terminal block on the shield that is labeled **DC Motor Power 5-12VDC**.

The DC Jack on the Arduino has a protection diode so you won't be able to mess things up too bad if you plug in the wrong kind of power. The terminal block has a protection FET so you will not damage the arduino/shield if you wire up your battery supply backwards, but it wont work either!

Here's how it works:



If you would like to have a **single DC power supply for the Arduino and motors**

Say a wall adapter or a single battery pack with 6-12VDC output, simply plug it into the DC jack on the Arduino or the 2-pin power terminal block on the shield. Place the power jumper on the motor shield.

Note that you may have problems with Arduino resets if the battery supply is not able to provide constant power, so it is not a suggested way of powering your motor project. You cannot use a 9V battery for this, it must be 4 to 8 AA batteries or a single/double lead acid battery pack.

If you would like to have the **Arduino powered off of USB** and the **motors powered off of a DC power supply**

Plug in the USB cable. Then connect the motor supply to the power terminal block on the shield. Do not place the jumper on the shield.

This is a suggested method of powering your motor project as it has a split supply, one power supply for logic, and one supply for motors

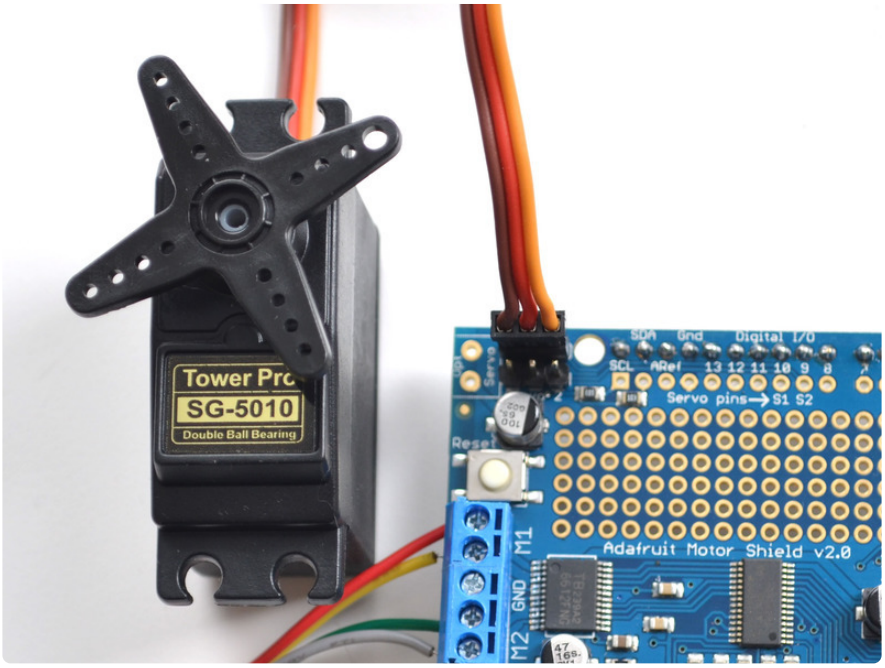
If you would like to have **2 separate DC power supplies for the Arduino and motors**.

Plug in the supply for the Arduino into the DC jack, and connect the motor supply to the power terminal block. Make sure the jumper is removed from the motor shield.

No matter what, if you want to use the DC motor/Stepper system the motor shield LED should be lit indicating good motor power

---

Using RC Servos



Hobby servos are the easiest way to get going with motor control. They have a 3-pin 0.1" female header connection with +5V, ground and signal inputs. The motor shield simply brings out the PWM output lines from Arduino pins 9 and 10 to two 3-pin headers so that its easy to plug in and go. They can take a lot of power so a 9V battery wont last more than a few minutes!

The nice thing about using the onboard PWM is that its very precise and goes about its business in the background. You can use the built in **Servo** library

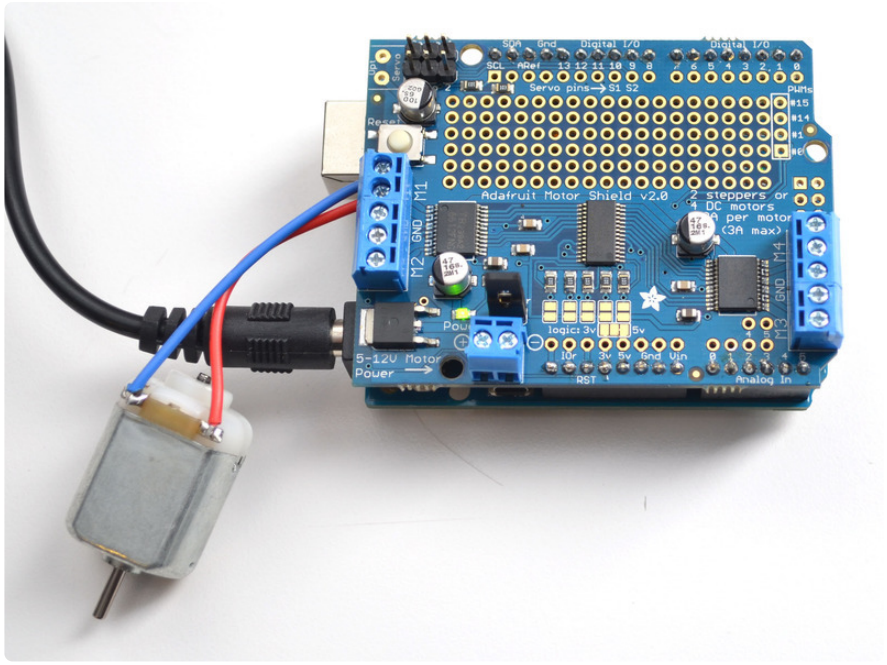
Using the servos is easy, please read the official Arduino documentation for how to use them and see the example Servo sketches in the IDE (https://adafru.it/aOD).

Powering Servos

Power for the Servos comes from the Arduino's on-board 5V regulator, powered directly from the USB or DC power jack on the Arduino. If you need an external supply, cut the 5v trace on the bottom of the board and connect a 5V or 6V DC supply directly to the **Opt Servo** power input. Using an external supply is for advanced users as you can accidentally destroy the servos by connecting a power supply incorrectly!

When using external servo power, be careful not to let it short out against the USB socket shell on the processor board. Insulate the top of the USB socket with some electrical tape.

Using DC Motors



DC motors are used for all sort of robotic projects.

The motor shield can drive up to 4 DC motors bi-directionally. That means they can be driven forwards and backwards. The speed can also be varied at 0.5% increments using the high-quality built in PWM. This means the speed is very smooth and won't vary!

Note that the H-bridge chip is not meant for driving continuous loads of 1.2A, so this is for small motors. Check the datasheet for information about the motor to verify its OK!

Connecting DC Motors

To connect a motor, simply solder two wires to the terminals and then connect them to either the M1, M2, M3, or M4. Then follow these steps in your sketch

Include the required libraries

Make sure you **#include** the required libraries

```
#include &lt;Wire.h&gt;
#include &lt;Adafruit_MotorShield.h&gt;
#include "utility/Adafruit_MS_PWMServoDriver.h"
```

Create the Adafruit_MotorShield object

```
Adafruit_MotorShield AFMS = Adafruit_MotorShield();
```

Create the DC motor object

Request the DC motor from the Adafruit_MotorShield:

```
Adafruit_DCMotor *myMotor = AFMS.getMotor(1);
```

with **getMotor(port#)**. **Port#** is which port it is connected to. If you're using M1 its **1**, M2 use **2**, M3 use **3** and M4 use **4**

Connect to the Controller

In your `setup()` function, call `begin()` on the Adafruit_MotorShield object:

```
AFMS.begin();
```

Set default speed

Set the speed of the motor using **setSpeed(speed)** where the **speed** ranges from 0 (stopped) to 255 (full speed). You can set the speed whenever you want.

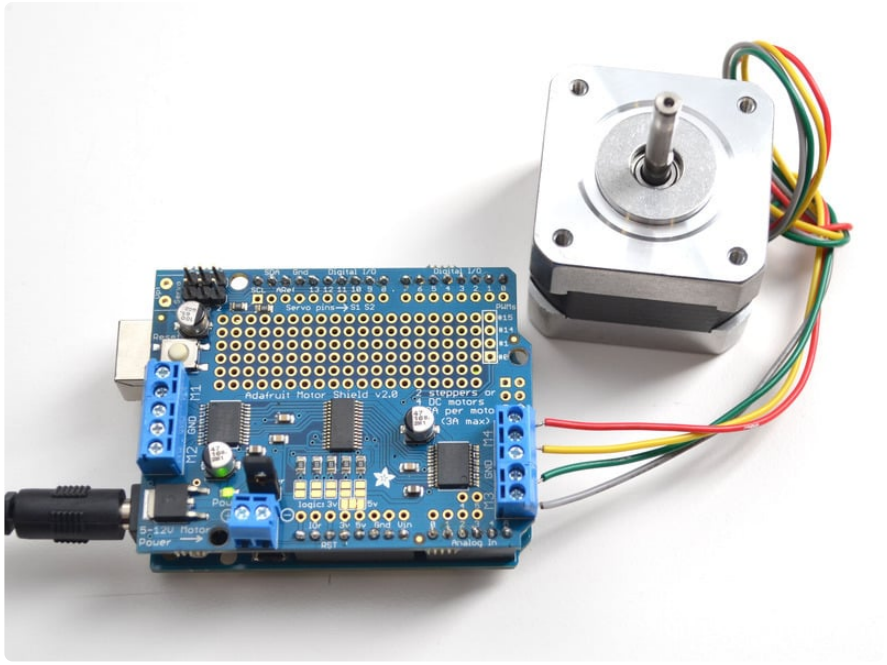
```
myMotor-&gt;setSpeed(150);
```

Run the motor

To run the motor, call **run(direction)** where **direction** is **FORWARD**, **BACKWARD** or **RELEASE**. Of course, the Arduino doesn't actually know if the motor is 'forward' or 'backward', so if you want to change which way it thinks is forward, simply swap the two wires from the motor to the shield.

```
myMotor-&gt;run(FORWARD);
```

Using Stepper Motors



Stepper motors are great for (semi-)precise control, perfect for many robot and CNC projects. This motor shield supports up to 2 stepper motors. The library works identically for bi-polar and uni-polar motors

Before connecting a motor, be sure to check the motor specifications for compatibility with the shield (https://adafru.it/r2d).

For unipolar motors: to connect up the stepper, first figure out which pins connected to which coil, and which pins are the center taps. If its a 5-wire motor then there will be 1 that is the center tap for both coils. Theres plenty of tutorials online on how to reverse engineer the coils pinout. (https://adafru.it/aOO) The center taps should both be connected together to the GND terminal on the motor shield output block. then coil 1 should connect to one motor port (say M1 or M3) and coil 2 should connect to the other motor port (M2 or M4).

For bipolar motors: its just like unipolar motors except theres no 5th wire to connect to ground. The code is exactly the same.

Running a stepper is a little more intricate than running a DC motor but its still very easy

Include the required libraries

Make sure you **#include** the required libraries

```
#include &lt;Wire.h&gt;
#include &lt;Adafruit_MotorShield.h&gt;
#include "utility/Adafruit_PWMServoDriver.h"
```

Create the Adafruit_MotorShield object

```
Adafruit_MotorShield AFMS = Adafruit_MotorShield();
```

Create the stepper motor object

Request the **Stepper** motor from the **Adafruit_MotorShield**:

```
Adafruit_StepperMotor *myMotor = AFMS.getStepper(200, 2);
```

...with `getStepper(steps, stepper#)`.

`Steps` indicates how many steps per revolution the motor has. A 7.5 degree/step motor has 360/7.5 = 48 steps.

`Stepper#` is which port it is connected to. If you're using M1 and M2, its port **1**. If you're using M3 and M4 indicate port **2**

Set default speed

Set the speed of the motor using `setSpeed(rpm)` where rpm is how many revolutions per minute you want the stepper to turn.

Run the motor

Then every time you want the motor to move, call the `step(#steps, direction, steptype)` procedure. **#steps** is how many steps you'd like it to take. direction is either **FORWARD** or **BACKWARD** and the step type is **SINGLE**, **DOUBLE**, **INTERLEAVE** or **MICROSTEP**.

- "Single" means single-coil activation
- "Double" means 2 coils are activated at once (for higher torque)

- "Interleave" means that it alternates between single and double to get twice the resolution (but of course its half the speed).
- "Microstepping" is a method where the coils are PWM'd to create smooth motion between steps.

Theres tons of information about the pros and cons of these different stepping methods in the resources page.

You can use whichever stepping method you want, changing it "on the fly" to as you may want minimum power, more torque, or more precision.

By default, the motor will 'hold' the position after its done stepping. If you want to release all the coils, so that it can spin freely, call `release()`

The stepping commands are 'blocking' and will return once the steps have finished.

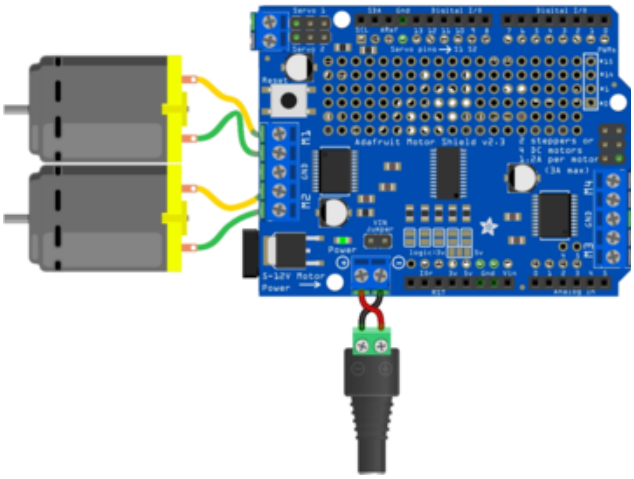
Because the stepping commands 'block' - you have to instruct the Stepper motors each time you want them to move. If you want to have more of a 'background task' stepper control, check out the AccelStepper library (https://adafru.it/X1e) . There are several AccelStepper examples included with the motor shield library.

Python & CircuitPython

We've written a handy CircuitPython library for the various DC Motor and Stepper kits called Adafruit CircuitPython MotorKit (https://adafru.it/DdU) that handles all the complicated setup for you. All you need to do is import the appropriate class from the library, and then all the features of that class are available for use. We're going to show you how to import the `MotorKit` class and use it to control DC and stepper motors with the Adafruit Stepper + DC Motor Shield.

CircuitPython Microcontroller Wiring

First assemble the Shield exactly as shown in the previous pages. There's no wiring needed to connect the Shield to the Metro. The example below shows wiring two DC motors to the Shield once it has been attached to a Metro. You'll want to connect a barrel jack to the power terminal to attach an appropriate external power source to the Shield. **The Shield will not function without an external power source!**



Connect the **two motor wires from the first motor** to **the M1 terminal on the Shield**.
Connect the **two motor wires from the second motor** to the **M2 terminal on the Shield**.
Connect the **positive side of the power terminal** to the **positive side of the barrel jack**.
Connect the **negative side of the power terminal** to the **negative side of the barrel jack**.

CircuitPython Installation of MotorKit and Necessary Libraries

You'll need to install a few libraries on your Metro board.

First make sure you are running the latest version of Adafruit CircuitPython (https://adafru.it/Amd) for your board.

Next you'll need to install the necessary libraries to use the hardware--carefully follow the steps to find and install these libraries from Adafruit's CircuitPython library bundle (https://adafru.it/uap). Our CircuitPython starter guide has a great page on how to install the library bundle (https://adafru.it/ABU).

If you choose, you can manually install the libraries individually on your board:

- **adafruit_pca9685.mpy**
- **adafruit_bus_device/**
- **adafruit_register/**
- **adafruit_motor/**
- **adafruit_motorkit.mpy**

Before continuing make sure your board's lib folder or root filesystem has the **adafruit_pca9685.mpy, adafruit_register, adafruit_motor, adafruit_bus_device** and **adafruit_motorkit.mpy** files and folders copied over.

Next connect to the board's serial REPL (https://adafru.it/Awz) so you are at the CircuitPython >>> prompt.

CircuitPython Usage

To demonstrate the usage, we'll initialise the library and use Python code to control DC and stepper motors from the board's Python REPL.

First you'll need to import and initialize the MotorKit class.

```
from adafruit_motorkit import MotorKit
kit = MotorKit()
```

DC Motors

The four motor spots on the Shield are available as `motor1`, `motor2`, `motor3`, and `motor4`.

In this example we'll use `motor1`.

Note: For small DC motors like sold in the shop you might run into problems with electrical noise they generate and erratic behavior on your board. If you see erratic behavior like the motor not spinning or the board resetting at high motor speeds this is likely the problem. See this motor guide FAQ page for information on capacitors you can solder to the motor to reduce noise (https://adafru.it/scl).

Now to move a motor you can set the `throttle` attribute. We don't call it speed because it doesn't correlate to a particular number of revolutions per minute (RPM). RPM depends on the motor and the voltage which is unknown.

For example to drive motor M1 forward at a full speed you set it to `1.0`:

```
kit.motor1.throttle = 1.0
```

To run the motor at half throttle forward use a decimal:

```
kit.motor1.throttle = 0.5
```

Or to reverse the direction use a negative throttle:

```
kit.motor1.throttle = -0.5
```

You can stop the motor with a throttle of `0`:

```
kit.motor1.throttle = 0
```

To let the motor coast and then spin freely set throttle to `None`.

```
kit.motor1.throttle = None
```

That's all there is to controlling DC motors with CircuitPython! With DC motors you can build fun moving projects like robots or remote controlled cars that glide around with ease.

Stepper Motors

Similar DC motors, stepper motors are available as `stepper1` and `stepper2`. `stepper1` is made up of the M1 and M2 terminals, and `stepper2` is made up of the M3 and M4 terminals.

We'll use `stepper1` in our example.

The most basic function (and the default) is to do one single coil step.

```
kit.stepper1.onestep()
```

You can also call the onestep function with two optional keyword arguments. To use these, you'll need to import stepper as well.

```
from adafruit_motor import stepper
```

Then you have access to the following options:

- `direction`, which should be one of the following constant values:
  - `stepper.FORWARD` (default)
  - `stepper.BACKWARD`.
- `style`, which should be one of the values:
  - `stepper.SINGLE` (default) for a full step rotation to a position where one single coil is powered
  - `stepper.DOUBLE` for a full step rotation to position where two coils are powered providing more torque
  - `stepper.INTERLEAVED` for a half step rotation interleaving single and double coil positions and torque

    - `stepper.MICROSTEP` for a microstep rotation to a position where two coils are partially active.
- `release()` which releases all the coils so the motor can free spin, and also won't use any power

The function returns the current step 'position' in microsteps which can be handy to understand how far the stepper has moved, or you can ignore the result.

To take a double-coil step backward call:

```
kit.stepper1.onestep(direction=stepper.BACKWARD, style=stepper.DOUBLE)
```

You can even use a loop to continuously call `onestep` and move the stepper, for example a loop of `200` microsteps forward for smooth movement:

```
for i in range(200):
    kit.stepper1.onestep(style=stepper.MICROSTEP)
```

That's all there is to controlling a stepper motor from CircuitPython! Steppers are handy motors for when you need smooth or precise control of something--for example 3D printers and CNC machines use steppers to precisely move tools around surfaces.

Full Example Code

For DC motors:

```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

"""Simple test for using adafruit_motorkit with a DC motor"""
import time
import board
from adafruit_motorkit import MotorKit

kit = MotorKit(i2c=board.I2C())

kit.motor1.throttle = 1.0
time.sleep(0.5)
kit.motor1.throttle = 0
```

For stepper motors:

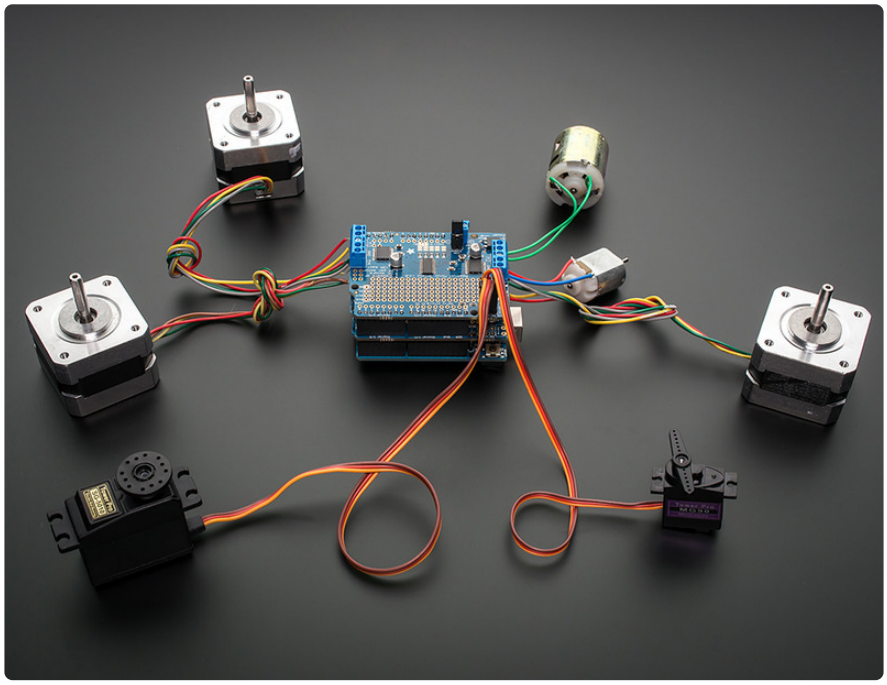
```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

"""Simple test for using adafruit_motorkit with a stepper motor"""
```

```
import time
import board
from adafruit_motorkit import MotorKit

kit = MotorKit(i2c=board.I2C())

for i in range(100):
    kit.stepper1.onestep()
    time.sleep(0.01)
```

Python Docs

Python Docs (https://adafru.it/DeE)

Stacking Shields



One of the cool things about this shield design is that it is possible to stack shields. Every shield you stack can control another 2 steppers or 4 DC motors (or a mix of the two)

You can stack up to 32 shields for a total of 64 steppers or 128 DC motors! Most people will probably just stack two or maybe three but hey, you never know. (PS if you drive 64 steppers from one of these shields send us a photo, OK?)

Note that stacking shields does not increase the servo connections - those are hard-wired to the Arduino digital 9 & 10 pins. If you need to control a lot of servos, you can use our 16-channel servo shield and stack it with this shield to add a crazy large # of servos. (http://adafru.it/1411)

Stacking shields is very easy. Each shield you want to stack on top of must have stacking headers installed. Check our instructions for how to do so. (https://adafru.it/ciP) The top shield does not have to have stacking headers unless you eventually want to put something on top of it.

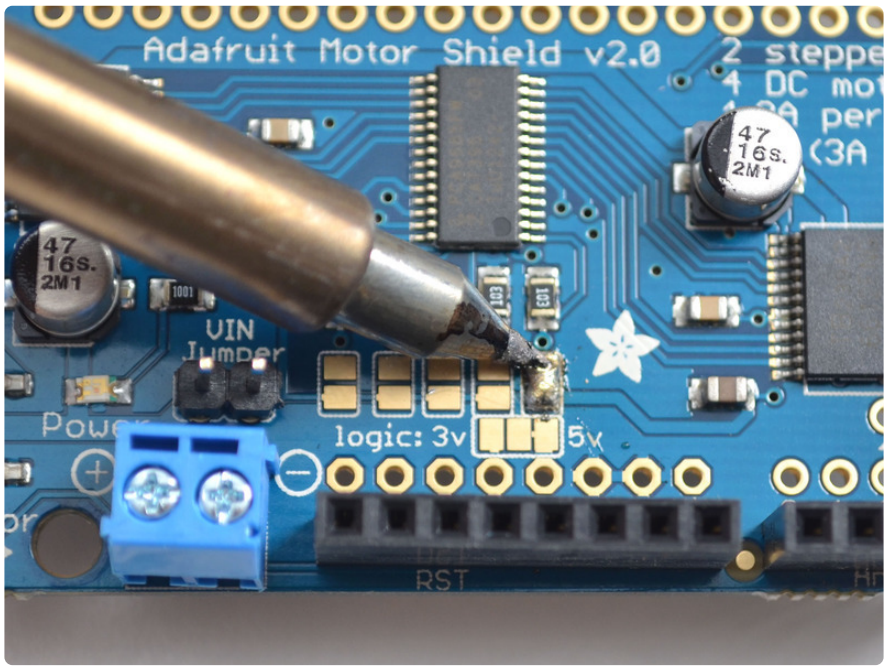
The only thing to watch for when stacking shields is every shield must have a unique I2C address. The default address is **0x60**. You can adjust the address of the shields to range from 0x60 to 0x7F for a total of 32 unique addresses.

Addressing the Shields

Each board in the chain must be assigned a unique address. This is done with the address jumpers on the lower edge of the board. The I2C base address for each board is 0x60. The binary address that you program with the address jumpers is added to the base I2C address.

To program the address offset, use a drop of solder to bridge the corresponding address jumper for each binary '1' in the address.

The right-most jumper is address bit #0, then to the left of that is address bit #1, etc up to address bit #4



Board 0: Address = 0x60 Offset = binary 0000 (no jumpers required)
Board 1: Address = 0x61 Offset = binary 0001 (bridge A0 as in the photo above)

Board 2: Address = 0x62 Offset = binary 0010 (bridge A1, to the left of A0)
Board 3: Address = 0x63 Offset = binary 0011 (bridge A0 & A1, two rightmost jumpers)
Board 4: Address = 0x64 Offset = binary 0100 (bridge A2, middle jumper)

etc.

Note that address 0x70 is the "all call" address for the controller chip on the shield. All boards will respond to address 0x70 - regardless of the address jumper settings.

Writing Code for Multiple Shields

The Adafruit_MotorShield library has the ability to control multiple shields, unlike the older AF_Motor library. First we must create a Motor Shield Controller for each shield, with the assigned address.

```
Adafruit_MotorShield AFMSbot(0x61); // Rightmost jumper closed
Adafruit_MotorShield AFMStop(0x60); // Default address, no jumpers
```

One motor shield is going to be called AFMSbot (bottom shield, so we remember) and one is AFMStop (top shield) so we can keep them apart. When you create the shield object, specify the address you set for it above.

Then we can request the motors connected to each one

```
// On the top shield, connect two steppers, each with 200 steps
Adafruit_StepperMotor *myStepper2 = AFMStop.getStepper(200, 1);
Adafruit_StepperMotor *myStepper3 = AFMStop.getStepper(200, 2);

// On the bottom shield connect a stepper to port M3/M4 with 200 steps
Adafruit_StepperMotor *myStepper1 = AFMSbot.getStepper(200, 2);
// And a DC Motor to port M1
Adafruit_DCMotor *myMotor1 = AFMSbot.getMotor(1);
```

You can request a stepper or DC motor from any port, just be sure to use the right AFMS controller object when you call **getMotor** or **getStepper**!

Then, both shields must have **begin** called, before you use the motors connected

```
AFMSbot.begin(); // Start the bottom shield
AFMStop.begin(); // Start the top shield
```

You can try out this code for yourself by setting up two shields and running the **File->Examples->Adafruit_MotorShield->StackingTest** example

---

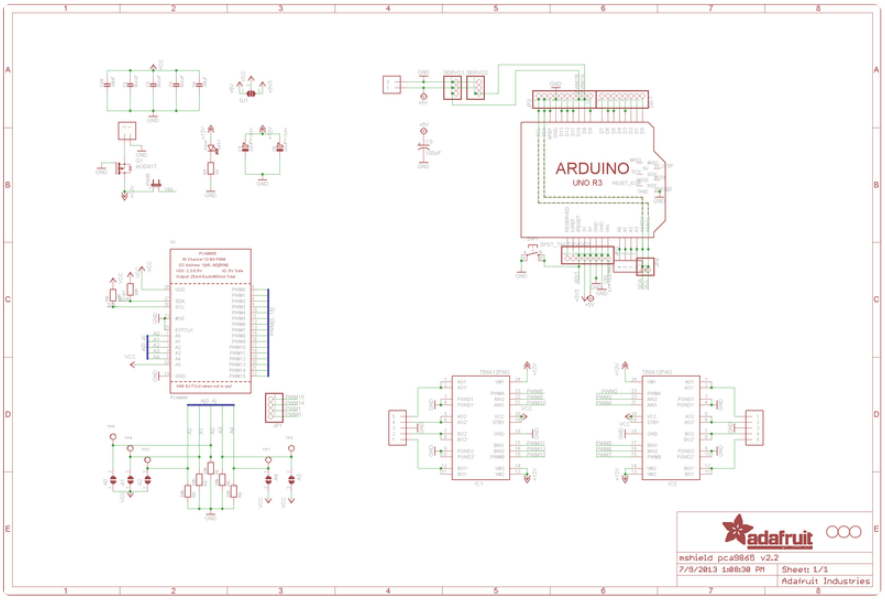
Resources

Motor ideas and tutorials

- Wikipedia has tons of information (https://adafru.it/aOF) on steppers
- Jones on stepper motor types (https://adafru.it/19oe)
- Jason on reverse engineering the stepper wire pinouts (https://adafru.it/aOI)

PCB files are on GitHub (https://adafru.it/DeF)

Schematic, click to embiggen



D.2. Conjunto motores y servomotores

37D Metal Gearmotors

Pololu 37D Metal Gearmotors are powerful brushed DC motors paired with 37mm-diameter gearboxes. There are nine different gearbox options available, ranging from 6.3:1 to 150:1, and two different motor options: 12 V and 24 V. The 24 V versions offer approximately the same speed and torque at 24 V as their 12 V counterparts do at 12 V, with approximately half the current draw. This datasheet includes two sets of performance graphs for each version, one at its nominal voltage and one at half of its nominal voltage. Each version is available with an integrated 64 CPR quadrature encoder on the motor shaft.

Note: The original versions of these gearmotors had gearboxes with all spur gears. In August 2019, these were replaced by functionally identical “Helical Pinion” versions that feature helical gears for the first stage of the gearbox, which reduces noise and vibration and improves efficiency. The picture on the right shows the helical pinion gear and first mating gear.

Performance summary and table of contents

Rated Voltage	Pololu Item #	Gear Ratio	No Load		At Maximum Efficiency				Max Power	Stall Extrapolation[2]		Graph Pages
			Speed	Current	Speed	Torque	Current	Output		Torque	Current	
		:1	RPM	A	RPM	kg·mm	A	W	W	kg·mm	A	
12 V	4750 [1]	1	10,000	0.2						5	5.5	
	4747, 4757	6.25	1600		1300	4.9	1.2	6.4	12	30		5, 6
	4748, 4758	10	1000		850	6.6	0.91	5.7	12	49		7, 8
	4741,4751	18.75	530		470	10	0.76	5.0	12	85		9, 10
	4742, 4752	30	330		280	18	0.78	5.1	12	140		11, 12
	4743, 4753	50	200		180	22	0.66	4.0	10	210		13, 14
	4744, 4754	70	150		130	32	0.68	4.2	10 [3]	270		15, 16
	4745, 4755	102.08	100		87	42	0.72	3.8	8 [3]	340		17, 18
	4746, 4756	131.25	76		66	60	0.74	4.1	6 [3]	450		19, 20
	2828, 2829	150	67		58	65	0.72	3.8	6 [3]	490		21, 22
24 V	4690 [1]	1	10,000	0.1						5.5	3.0	
	4688, 4698	6.25	1600		1300	5.5	0.58	7.4	14	35		23, 24
	4689, 4699	10	1000		850	7.5	0.49	6.6	14	55		25, 26
	4681,4691	18.75	530		450	13	0.49	6.1	13	95		27, 28
	4682, 4692	30	330		280	19	0.46	5.5	13	150		29, 30
	4683, 4693	50	200		170	27	0.41	4.9	12	230		31, 32
	4684, 4694	70	140		120	39	0.42	5.0	10 [3]	310		33, 34
	4685, 4695	102.08	100		86	51	0.42	4.5	8 [3]	390		35, 36
	4686, 4696	131.25	79		68	63	0.40	4.4	6 [3]	470		37, 38
	4687, 4697	150	68		59	73	0.41	4.4	6 [3]	560		39, 40

Notes:

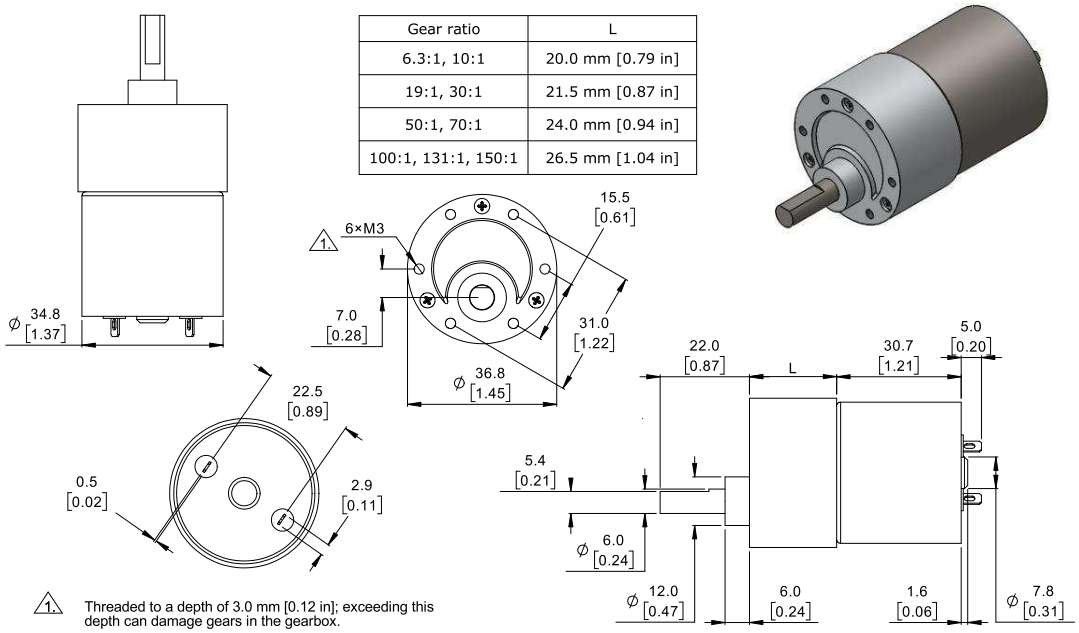
(1) Max efficiency data and performance graphs currently unavailable for the motors without gearboxes (items #4750 and #4690).

(2) Listed stall torques and currents are theoretical extrapolations; units will typically stall well before these points as the motors heat up. Stalling or overloading gearmotors can greatly decrease their lifetimes and even result in immediate damage. The recommended upper limit for continuously applied loads is 100 kg·mm, and the recommended upper limit for instantaneous torque is 250 kg·mm. Stalls can also result in rapid (potentially on the order of seconds) thermal damage to the motor windings and brushes; a general recommendation for brushed DC motor operation is 25% or less of the stall current.

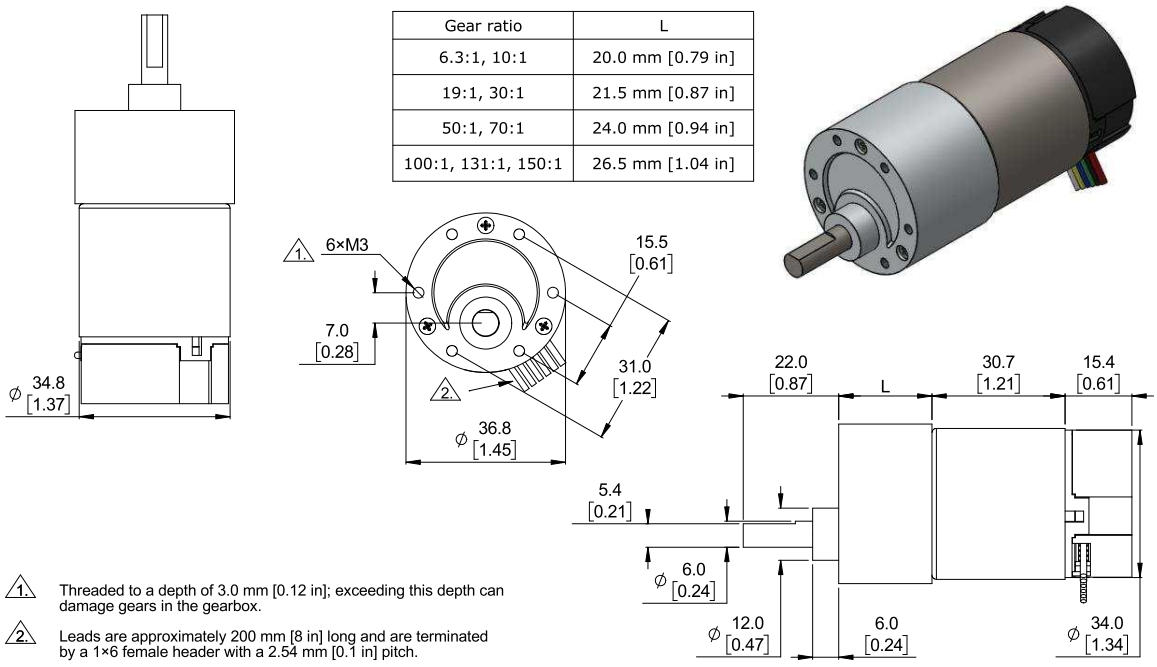
(3) Output power for these units is constrained by gearbox load limits; spec provided is output power at max recommended load of 100 kg·mm.

Dimensions (units: mm over [inches])

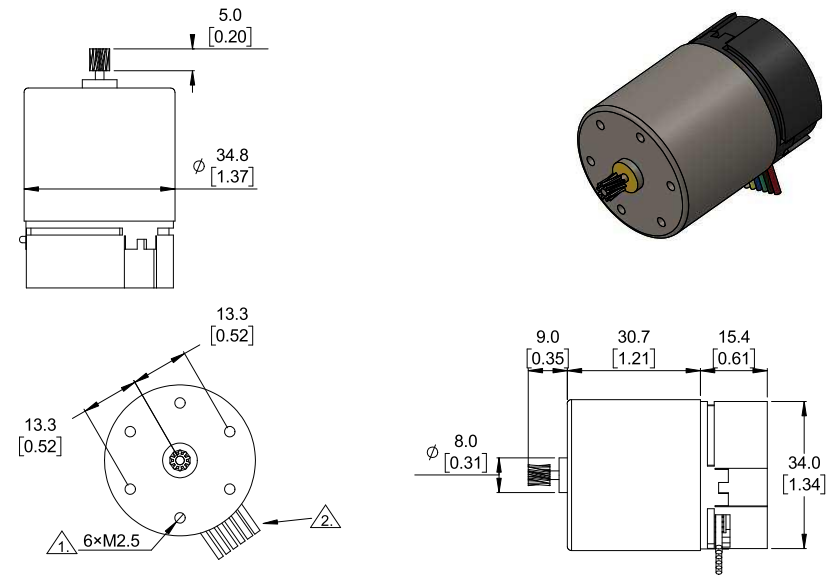
Gearmotor versions without encoders (items #2829, 4681–4689, 4741–4748) weight: 175 g to 195 g



Gearmotor versions with encoders (items #2828, 4691–4699, 4751–4758) weight: 190 g to 210 g



Motor with encoder and no gearbox (items #4690, 4750) weight: 110 g



1. Threaded to a depth of 3.5 mm [0.14 in]; exceeding this depth can damage the motor.
2. Leads are approximately 200 mm [8 in] long and are terminated by a 1×6 female header with a 2.54 mm [0.1 in] pitch.

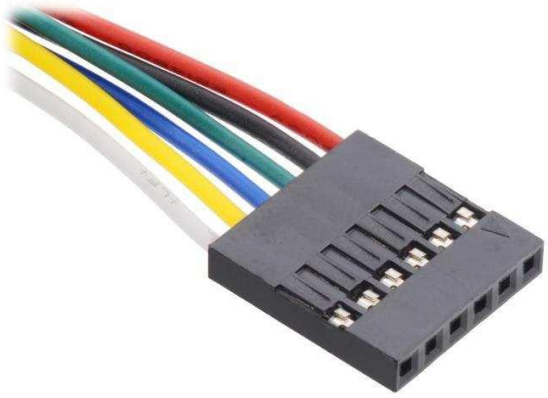
Using the encoder

Versions with encoders have additional electronics mounted on the rear of the motor. Two Hall-effect sensors are used to sense the rotation of a magnetic disc on a rear protrusion of the motor shaft. The encoder electronics and magnetic disc are enclosed by a removable plastic end cap. The following pictures show what the encoder portion looks like with the end cap removed:

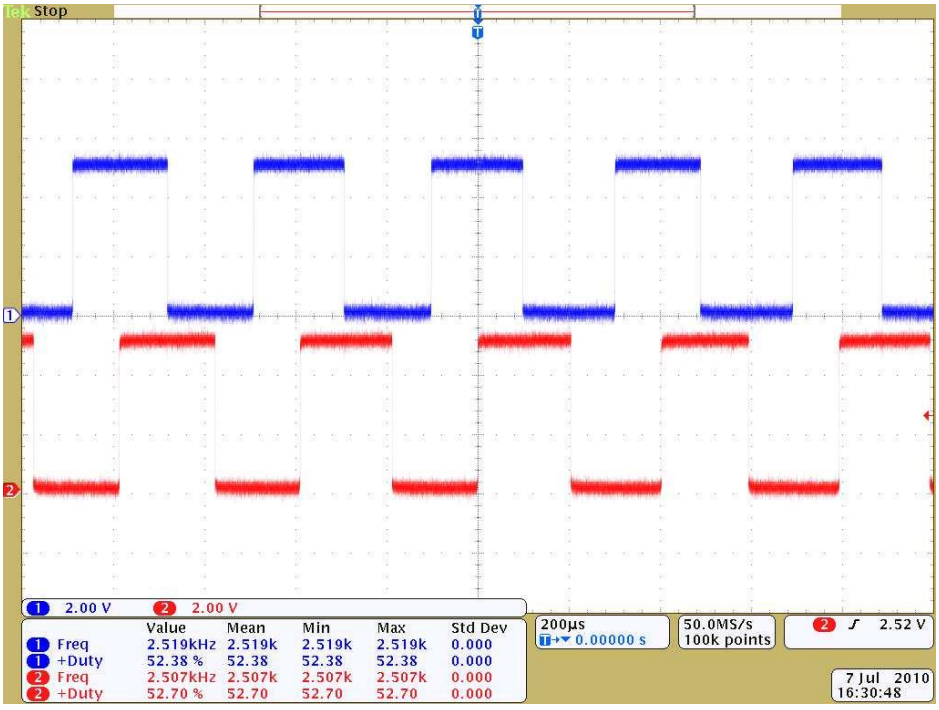
The quadrature encoder provides a resolution of 64 counts per revolution (CPR) of the motor shaft when counting both edges of both channels. To compute the counts per revolution of the gearbox output, multiply the gear ratio by 64.

The motor/encoder has six color-coded, 20 cm (8″) leads terminated by a 1×6 female connector with a 2.54 mm (0.1″) pitch. This connector works with standard 0.1″ male breakaway headers and Pololu male premium jumper and precrimped wires. If this header is not convenient, the crimped wires can be pulled out of the 1×6 housing and used with different crimp connector housings instead (e.g. 1×2 for the motor power and 1×1 housings for the other four leads), or the connectors can be cut off entirely.

Lead Color	Function
Red	Motor power
Black	Motor power
Green	Encoder ground
Blue	Encoder Vcc (3.5 V to 20 V)
Yellow	Encoder A output
White	Encoder B output



The Hall sensors require an input voltage, Vcc, between 3.5 V and 20 V and draw a maximum of 10 mA. The A and B outputs are square waves from 0 V to Vcc approximately 90° out of phase. The speed of the motor can be determined from the frequency, and the direction of rotation can be determined from the order of the transitions. The following oscilloscope capture shows the A and B (yellow and white) encoder outputs using a 12 V motor at 12 V and a Hall sensor Vcc of 5 V:



Counting both the rising and falling edges of both the A and B outputs results in 64 counts per revolution of the motor shaft. Using just a single edge of one channel results in 16 counts per revolution of the motor shaft, so the frequency of the A output in the above oscilloscope capture is 16 times the motor rotation frequency.

深圳飞特模型有限公司

SHENZHEN FEETECH RC MODEL CO.,LTD.

产品规格书

PRODUCT SPECIFICATION

客　　户(CUSTOMER)　　：

型　　号(MODEL)　　：FS5115M

名　　称(PRODUCT NAME)：6V 15kg.cm数码180度塑料壳铜齿轮铁芯舵机

版　　本(EDITION)　　：A/0

在您下单采购本产品之前，请阅读本规格书并在下列确认栏内签字回传。

Sign back before you place an order to purchase this product,
please read this specification and in the following admit column.

飞特确认/ FEETECH APPROVED

编写 EDIT	审核 CHECKED	确认 APPROVED
李永传	巫耿炎	冯潭新

客户确认/CUSTOMER APPROVED

审阅 REVIEW	审核 CHECKED	确认 APPROVED

地址：广东省深圳市龙岗区横岗镇六约埔厦路60号2楼
ADD:Floor 2,No 60,PuXia Road,LiuYueHengGang Town, Long Gang District,
ShenZhen,518173,China
电话/Tel：0755-89335266　网址/Website：www.feetech.cn

本规格书包含以下内容 CONTENT OF PRODUCT

Revise Record变更记录表

序号 No.	变更内容Content Of Change	版本Edition	更变日期Date	备注Remark
1	新板规格书初次发行	A/0	20210802	

1.使用环境条件 Apply Environmental Condition:

No.	项目(Item)	规格(Specification)
1-1	保存温度Storage Temperature Range	-30℃～80℃
1-2	运行温度Operating Temperature Range	-20℃～60℃

2.测试环境 Standard Test Environment:

No.	项目(Item)	规格(Specification)
2-1	温度 Temperature Range	25℃ ±5℃
2-2	湿度 Humidity Range	65%±10%

3.安全特性与认证 Secure Performance and Certificate

No.	项目/Item	认证Certificate	备注（remarks）
3-1	ASTM F963	No	美国玩具检测标准
3-2	EN71	No	欧盟市场玩具类产品的规范标准
3-3	FCC	No	对电子、电器中的电磁干扰，管理和控制无线电频率范围，保护电信网络、电器产品的正常工作。
3-4	EMC	Yes	对任何的物质而言，不给其无法容许的电磁干扰波，且在电磁环境中还需能具有满足其功能的机器，装置或系统的能力。
3-5	ROHS	Yes	电子，电器设备中对人体有害物质的含量。
3-6	REACH	NO	欧盟法规《化学品的注册、评估、授权和限制》

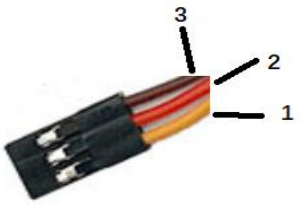
4.外观检查 Appearance Inspection:

No.	项目(Item)	规格(Specification)
4-1	外观尺寸OutlineDrawing	尺寸见附件(第12项) See the appendix
4-2	外观Appearance	无损坏，不允许影响功能 No damage and mustn’t affect functions.

5.电气特性 Electrical Specification

No.	项目(Item)	规格(Specification)	规格(Specification)	备注Remark
5-1	工作电压 Operating Voltage	4.8V	6V	
5-2	空载速度 No Load Speed	0.25sec/60° (40RPM)	0.2sec/60° (50RPM)	
5-3	空载电流 Running Current (no load)	180mA	220mA	
5-4	堵转扭力 Stall Torque (at locked)	14kg.cm	15.5kg.cm	
		194.8oz.in	215.6oz.in	
5-5	堵转电流 Stall Current (at locked)	1.3A	1.7A	
5-6	静态电流 Idle Current (at stopped)	6mA	6mA	
5-7	工作电压 Input Voltage	4V-8.4V		
5-8	额定负载 Rated Torgue	4.5kg.cm	5kg.cm	
5-9	额定电流 Rated Current	450mA	600mA	
5-10	Kt常数	9kg.cm/A		

6 机械规格 Mechanical Specification:

No.	项目(Item)	规格(Specification)		备注 (remarks)
6-1	外观尺寸 Size	40.1X20.1X38.9mm		见图纸See drawings
6-2	机构极限角度 Limit Angle	200°		
6-3	外壳材质 Case material	PA+Fiberglass		
6-4	齿轮材质 Gear material	铜 Copper		
6-5	轴承类型 Bearing type	滚珠轴承 Ball bearings		
6-6	角度传感器 Angle Sansor	类型Type	Carbon-Film Potentiometer	
		角度 Angle	220°	
		寿命 Life	1000000Cycles Min	
6-7	连接器和电缆 Connector and Cable	类型 Type	JR	
		Material材质	PVC	
		线长度 Length	30CM	
		引脚定义 Pin Definition	1 Signal	
			2 Vcc	
			3 GND	
6-8	重量The Weight	60.7± 1g		
6-9	出力轴 Horn Type	25T/5.9mm		
6-10	减速比Gear Ratio	1/300		
6-11	齿轮虚位Back Lash	≦0.5°		
6-12	摇臂虚位 The rocker phantom	0°		
6-13	出力轴螺丝 The rocker screw	M3X6		
6-14	马达Motor	Core Motor		

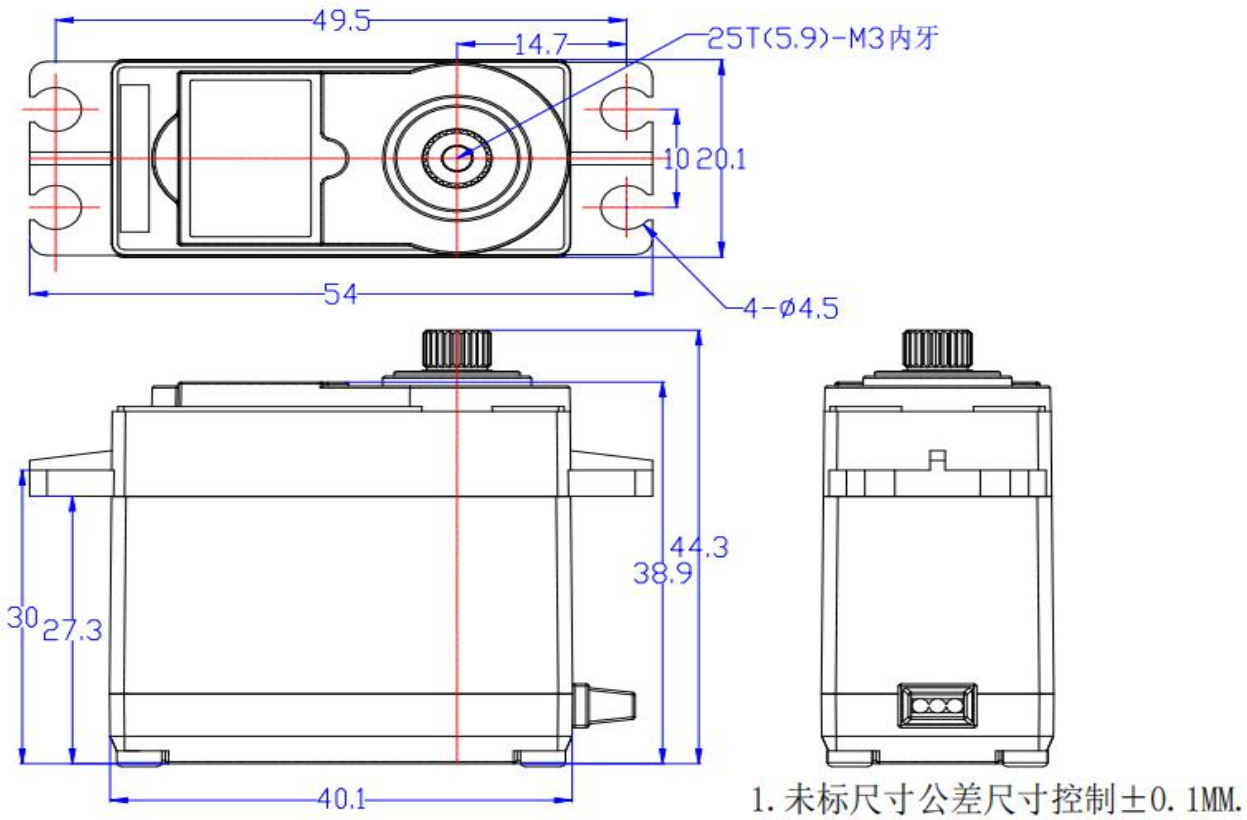
7 控制特性Control Specification:

No.	项目(Item)	规格(Specification)	备注 (remarks)
7-1	控制信号 Command Signal	Pulse width modification	
7-2	控制系统类型 Control System Type	Digital comparator	
7-3	操作角度 Operating Travel	180° (at 500→2500μsec)	
7-4	两边角度差 Left&Right Travelling Angledeviation	≤ 5°	
7-5	回中差 Centering Deviation	≦1°	
7-6	中立位置 Neutral Position	1500 μsec	
7-7	死区宽度 Dead Band Width	≤4 μsec	
7-8	旋转方向 Rotating Direction	逆时针 Counterclockwise(在1500→2000 μsec)	
7-9	脉波宽度范围 Pulse Width Range	500→2500 μsec	

8 可靠性测试 Reliability Testing

No.	项目(Item)	规格 (Specification)	备注（remarks）
8-1	负载寿命测试Life test	＞100000次	正转60度0.25S，停0.5S,反转60度0.25S，停0.5S为一次，持续循环工作，负载1/5堵转扭矩
8-2	马达噪音The motor noises	45±5dB	环境噪音40-45 dB　距离噪音仪30cm处测试
8-3	舵机噪音Steering gear noises	70±5dB	环境噪音40-45 dB　距离噪音仪30cm处测试-1/3空载速度测试电压6V
8-4	防水性能Waterproof performance	NO	

9 外观尺寸 Outside Dimension(单位Unit：mm)



1.未标尺寸公差尺寸控制±0.1MM.

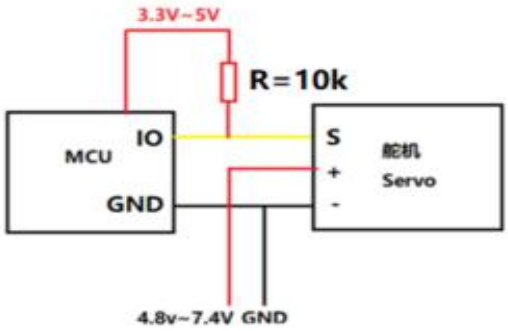
10 配件 Accessories

No Accessories

11 接口定义 Interface Definition

控制信号参数（Control signal parameter）

信号周期 Signal Period	20ms	
脉冲宽度 Pulse Width	500us-2500us	
信号高电平电压 Signal high Voltage	2V-5V	
信号低电平电压 Signal Low Voltage	0.0V-0.45V	



12 包装出货简图 The Packaging Detail

E Enlaces de los componentes

- **Componentes electrónicos**
  - Arduino Due

https://store.arduino.cc/en-es/products/arduino-due?srsltid=AfmBOoqK_tJsPK4jc8mQvFKwGdU4rarRKlPPID-KlMz1gGUjKLo96c_w

  - Adafruit Motor Shield V2

https://www.adafruit.com/product/1438

  - Módulo Bluetooth HC-06

https://tienda.bricogeek.com/modulos-bluetooth/1351-modulo-bluetooth-hc-06.html?srsltid=AfmBOoqeJVt0mQV3M7ehxQTav55JgRly7R7aag3sOhWXtg5IT27LSSy7

  - Motor DC

https://tienda.bricogeek.com/motores-dc/815-motor-con-reductora-701-con-encoder.html?vt_campaign=1022310&vt_content=10024707&vt_product=815&vt_alg=UpSell

  - Servomotor

https://tienda.bricogeek.com/servomotores/1321-servo-feetech-15kg-fs5115m-fb-con-feedback.html?search_query=servomotor+feetech&results=8

  - Batería Li-Po

https://tienda.bricogeek.com/baterias-lipo/1401-bateria-lipo-3500mah-3s-25c-11-1v.html?search_query=bateria+lipo+&results=128

  - Cargador batería Li-Po

https://tienda.bricogeek.com/baterias-lipo/1149-cargador-balanceador-imax-b6-de-80w-y-alimentador.html?utm_source=tienda&utm_medium=click&utm_campaign=prodrel

  - Conversor DC-DC

https://tienda.bricogeek.com/convertidores-de-voltaje/1399-conversor-dc-dc-ajustable-5a-75w-xl4015.html?search_query=conversor+dc&results=60

- **Otros componentes**

  - Rodamiento 15x35x11 *mm*

https://es.rs-online.com/web/p/rodamientos-de-bolas/6190187

  - Conector alimentación Arduino

https://tienda.bricogeek.com/cables/331-cable-adaptador-9v-jack.html?srsltid=AfmBOooFc0GPSvERrn9Wy37kYJRM_l1jZGdxzR-id3ky471jFwqmZwdd

  - Interruptor XT60

https://es.aliexpress.com/item/1005003120462424.html

  - Cable Dupont MM 40 *cm*

https://tienda.bricogeek.com/cables/1578-cables-dupont-macho-macho-40-cm-40-unidades.html?search_query=cable+dupont&results=38

  - Cable Dupont MH 40 *cm*

https://tienda.bricogeek.com/cables/1577-cables-dupont-macho-hembra-40-cm-40-unidades.html?search_query=cable+dupont&results=38

  - Cable Dupont MM 20 *cm*

https://tienda.bricogeek.com/cables/1361-cables-dupont-macho-macho-20-cm-40-unidades.html?search_query=cable+dupont&results=38

  - Cable Dupont MH 20 *cm*

https://tienda.bricogeek.com/cables/1362-cables-dupont-macho-hembra-20-cm-40-unidades.html?search_query=cable+dupont&results=38