-Trabajo de Fin de Grado-

“Diseño e implementación de un robot móvil con sistema de movimiento alternativo y de su sistema odométrico”

ESCUELA SUPERIOR DE INGENIERÍA Y TECNOLOGÍA

Grado en Ingeniería Electrónica Industrial y Automática

Autor: Alberto Díaz Rodríguez

Tutor: Jonay Tomás Toledo Carrillo

La Laguna, a 12 de junio de 2019

D. **Jonay Tomás Toledo Carrillo**, con N.I.F. 78.698.554-Y profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor

C E R T I F I C A

Que la presente memoria titulada:

"Diseño e implementación de un robot móvil con sistema de movimiento alternativo y de su sistema odométrico"

ha sido realizada bajo su dirección por D. **Alberto Díaz Rodríguez**, con N.I.F. 51.166.741-K.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 12 de junio de 2019.

Agradecimientos

A mis padres, a mis hermanos y a toda mi familia,

que siempre ha estado a mi lado ante las adversidades.

A mis compañeros y amigos, por el apoyo

que me han ofrecido para recorrer este camino.

A los profesores, por todo lo que me han enseñado.

A mi tutor Jonay, por su inestimable ayuda y

enseñanzas que se ven reflejadas en este trabajo.

Personas que han estado a mi lado todo este tiempo.

Gracias.

Resumen.

En el presente proyecto, se ha dotado de un sistema de gran maniobrabilidad a un robot móvil fabricado a partir de piezas impresas en 3D. Con respecto a esta idea, ha sido necesario realizar un diseño singular, que le permitiese al robot describir cualquier movimiento de rotación, de traslación, o una combinación de ambos.

Al mismo tiempo, se ha implementado un sistema para el control del movimiento que respondiese a las necesidades enunciadas, el cual se comunica de forma inalámbrica con el microcontrolador instalado en el robot. En concreto, se utiliza una conexión Wifi para el envío de las velocidades de los motores, las cuales parten de un software definido en ROS (Sistema Operativo Robótico).

Durante la navegación, el prototipo desarrollado es capaz de estimar su posición gracias a un modelo de odometría, el cual se ha diseñado atendiendo a las características particulares del modelo implementado. Se genera de este modo un mensaje con la pose del robot, el cual se envía de vuelta a ROS, donde se procesa la información con el objetivo de generar una representación gráfica, en la que se muestra la navegación del robot en tiempo real.

Abstract.

In this project, it has been equipped with an innovative manoeuvrable system, a mobile robot manufactured from 3D printed parts. Regarding this idea, it has been necessary to perform a remarkable design in order to allow the robot to describe any movement of rotation, translation, or a combination of both.

At the same time, it has been implemented an adequate control system for the aforementioned needs, which communicates wirelessly with the microcontroller installed in the robot. Specifically, a Wi-Fi connection is used to send the speeds of the motors from a software defined in ROS (Robot Operating System).

During navigation, the developed prototype can estimate its position thanks to an odometry model, which has been designed according to the characteristics of the built robot. Thereby, a message with the pose of the robot is generated, which is sent back to ROS, where the information is processed in order to generate a real time graphic representation, where the robot navigation is shown.

Índice de contenidos

Índice de figuras

Índice de tablas

Capítulo I. Introducción

En este Trabajo de Fin de Grado se ha querido atender a dos aspectos de gran relevancia dentro del mundo de la ingeniería. En primer lugar, resulta indudable que uno de los grandes objetivos de la ingeniería, tanto referida a un entorno industrial como a la sociedad en general, es la mejora continua de las tecnologías que están a disposición de las personas. En este sentido, y particularizando la idea en este proyecto, se ha estudiado un sistema de movimiento que se alejase de lo convencional, ofreciendo al sistema en el que se instale, una gran maniobrabilidad. En concreto se ha querido implementar un conjunto de ruedas omnidireccionales en un robot fabricado a partir de piezas impresas en 3D. El prefijo *omni-* significa todo, y es que el objetivo que se ha perseguido es que el robot pudiese moverse hacia cualquier dirección y sentido dentro de un plano. En el sentido de las ideas con las que se ha comenzado, las conclusiones que se obtengan a partir de este estudio podrían tener aplicación tanto en los robots móviles que estén presentes en un entorno industrial, a los que se les quiera dotar de este sistema de movimiento, como en los objetos dotados de movimiento que utilizan las personas en sus vidas cotidianas.

En segundo lugar, se ha pretendido atender a una tendencia de rigurosa actualidad como lo es el concepto del *internet of things*, lo cual consiste en la interconexión de los objetos cotidianos a través de una red, de modo que puedan interaccionar con otros elementos del entorno. Atendiendo al robot que se ha pretendido desarrollar en este proyecto, y dado que desde un primer momento se consideró el utilizar la plataforma Arduino para el control del sistema, existen diversas opciones para dotar estas placas de la capacidad para conectarse a una red Wifi. Es en este aspecto en el que se encuentra inmersa la idea planteada, ya que este tipo de dispositivos pueden aplicarse en una gran variedad de proyectos, aumentando de este modo las posibilidades que ofrece la plataforma Arduino.

Resulta conveniente mencionar que, junto a todo lo anterior, ha sido objeto del proyecto, el diseño de un sistema para el estudio de la odometría del robot diseñado, de modo que se pudiese monitorizar la posición de este durante la navegación.

1.1. **Antecedentes**

En la actualidad, los microcontroladores Arduino están cada vez más presentes en el desarrollo de proyectos como al que se atiende en esta memoria. Cada vez son más los módulos que se pueden incorporar a estas placas, aumentando las posibilidades que estas ofrecen.

En ciertos casos, es posible encontrar modelos de Arduino en los que estos añadidos ya están incorporados. Es el caso de Arduino WeMos D1, que incluye un chip de comunicaciones Wifi. No obstante, es posible añadir módulos con esta misma función en otro tipo de placas. Este panorama, en el que se ofrecen todas estas posibilidades, supone una oportunidad para poder escoger aquel hardware que mejor se adapte a las características del proyecto que se esté realizando.

En otro sentido de las ideas, en el momento en el que se redacta esta memoria, resulta muy común encontrar robots en la industria, algunos de ellos operando de forma autónoma, y otros auxiliando a los trabajadores con sus tareas. Algunos de estos sistemas son móviles. En esos casos, cada vez resulta más importante que posean un sistema de movilidad que les confiera una destacada maniobrabilidad, de modo que se puedan desplazar con comodidad por las instalaciones, de igual forma que puedan satisfacer el requerimiento por el que se cuenta dicha máquina. En relación a esta idea, existen muchas soluciones, una de ellas es el diseño de los denominados robots omnidireccionales. Este concepto se desarrollará en detalle en los siguientes apartados, pero como idea general se puede decir que estos equipos permiten desplazarse hacia cualquier dirección dentro de un plano, del mismo modo que realizar movimientos en los que intervenga la rotación. Este es el sistema que se ha querido estudiar en este proyecto.

Junto a todo lo anterior, ROS se ha estandarizado como uno de los métodos de mayor éxito para el control de sistemas robóticos, el cual, además, ofrece numerosas herramientas que podrían utilizarse para integrar el sistema de control del robot que se desarrolla en este Trabajo de Fin de Grado.

1.2. **Presentación del proyecto**

A partir de una serie de ideas iniciales, inspiradas en los planteamientos presentados al inicio del documento, se ha implementado un robot con un sistema de movimiento alternativo, el cual le proporciona una gran movilidad. Para ello, se ha diseñado una estructura con forma de triángulo equilátero, y se ha colocado en cada lado una rueda omnidireccional. Una rueda omnidireccional consiste en una estructura en la que se aloja el orificio para la conexión al eje de un motor, y una serie de soportes, cuyo número varía dependiendo del diseño concreto, y que permiten situar unos rodillos alrededor de la periferia de la rueda, de modo que sean estos los que entren en contacto con el suelo.

Se ha escogido este tipo de rueda debido a que permiten realizar tanto movimientos de rodadura, como cualquier otra rueda convencional, como deslizamientos perpendiculares al eje, lo cual es una clara mejora respecto a otros modelos. Observando la siguiente fotografía [1] se puede entender mejor el concepto que se ha tratado de expresar.

Figura 1. Movimientos con las ruedas omnidireccionales

Todo esto ha permitido que el robot pueda moverse hacia cualquier dirección y sentido dentro de un plano, realizando tanto traslaciones o rotaciones puras como movimientos combinados.

Se ha mencionado que se escogió una estructura en forma de triángulo equilátero. El motivo por el que se tomó esta decisión es que esta distribución es la que permite realizar todos los movimientos que requería el proyecto, con el mínimo número de ruedas omnidireccionales posible. Existen distintos modelos de ruedas omnidireccionales, los cuales se analizarán en el siguiente capítulo, pero se puede adelantar que el modelo seleccionado permite realizar los movimientos básicos que se muestran en la siguiente imagen gracias a la distribución escogida.

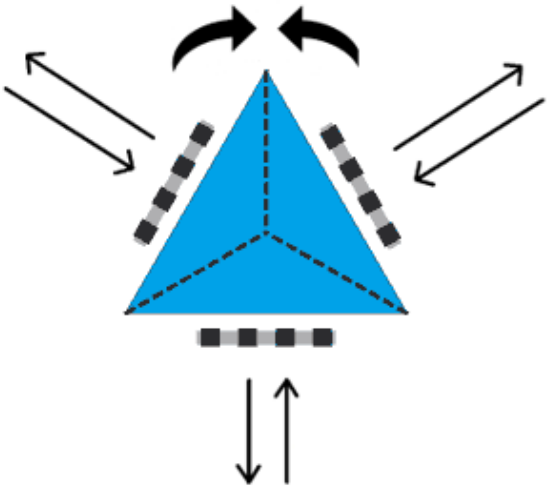


Figura 2. Estructura de robot omnidireccional implementada

Adicionalmente, y actuando sobre la velocidad de giro de cada una de las ruedas es posible realizar cualquier desplazamiento que se desee en el plano.

Tanto la estructura del robot como las ruedas se han diseñado para una posterior fabricación empleando una impresora 3D. El material que se ha utilizado es el PLA

mayoritariamente, pero también se ha utilizado filaflex para aquellas zonas de las ruedas que entran en contacto con el suelo, ya que este material presenta una serie de singularidades que lo hacían más adecuado para esas partes.

Se han utilizado distintas placas Arduino a lo largo del proyecto, elaborando distintos modelos de control según las posibilidades que ofrecían cada una de ellas. En este sentido, hubo que descartar la posibilidad de utilizar el Arduino UNO ya que este no permite realizar una conexión a una red Wifi. También se desechó la opción de utilizar un Arduino WeMos D1 ya que, a pesar de que este solucionaba los problemas del modelo anterior, no contaba con el número de pines que eran necesarios para controlar el robot. Finalmente se ha instalado un Arduino MEGA, el cual, junto a un módulo de comunicaciones wifi ESP-01, han permitido que el control del robot se realice de forma inalámbrica. Al otro lado de esta comunicación se encuentra un sistema implementado en ROS (Sistema operativo robótico) que incorpora aquellos nodos que son necesarios para tomar los datos de un mando para videojuegos, procesar la información, determinando de este modo la velocidad a la que se debe mover cada uno de los motores que incorpora el robot, de forma que este describa el movimiento que se solicita. Enviando esta información a través de una conexión Wifi mediante un sistema denominado Ros_Arduino_Bridge. El protocolo que se ha utilizado en el proyecto es el UDP, ya que es el que permitía un mejor funcionamiento del sistema.

Junto a todo lo anterior, se ha elaborado un modelo para el cálculo de la odometría del robot, es decir, el estudio de la posición en la que se encuentra el sistema, a partir del movimiento de los motores. En este sentido, se ha considerado como la mejor alternativa, colocar motores paso a paso en vez de motores de continua, ya que estos primeros permiten realizar un control del giro de las ruedas de forma más precisa y sencilla.

Es la placa Arduino la que calcula la posición en la que se encuentra el robot, de modo que, para poder monitorizar la navegación en el ordenador, ha sido necesario que el Arduino envíe un mensaje con la pose del robot. Este mensaje lo recibe un nodo de ROS en el que se ha implementado un socket UDP. Este nodo procesa la información recibida, generando un mensaje ROS de tipo Odom, el cual permite que cualquier aplicación pueda acceder, en un formato estándar, a la posición del robot. Un ejemplo de utilización es la aplicación Rviz, que es una herramienta de visualización 3D que incluye ROS, la cual muestra el movimiento que describe un robot en tiempo real.

De los anteriores planteamientos de pueden deducir los distintos hitos que se han logrado en este proyecto, los cuales son:

1. Diseño y fabricación de una estructura de robot con una impresora 3D.
2. Fabricación de las ruedas omnidireccionales.
3. Control del movimiento del robot con motores paso a paso.
4. Electrónica necesaria y batería.
5. Diseño de un sistema de control adaptado a las características del robot.
6. Sistema de envío de mensajes con la velocidad de los motores mediante el protocolo UDP.

7. Sistema para la recepción de los mensajes con la pose del robot mediante el protocolo UDP.
8. Control inalámbrico del robot a través de ROS.

1.3. Contenido de la memoria

En esta memoria se describen aquellos contenidos necesarios para documentar el desarrollo realizado en este Trabajo de Fin de Grado. Para ello, se ha definido una estructura de 6 capítulos claramente diferenciados.

En primer lugar, se ofrece una introducción en la que se contextualiza el trabajo, y en la que se incluye un primer acercamiento al prototipo diseñado.

Posteriormente, se expresa desde una perspectiva puramente teórica, los conceptos relacionados con el proyecto. En este caso, se atenderá tanto al software como al hardware utilizado, así como a los protocolos de comunicación que se han podido estudiar.

En el tercer capítulo se describe el desarrollo, a nivel práctico, que se ha llevado a cabo, exponiendo los distintos modelos que se han implementado con el propósito de alcanzar los objetivos a los que se ha hecho referencia en el inicio de este capítulo. En este mismo sentido de las ideas, se ilustrarán los modelos que se han diseñado en 3D para la posterior fabricación del robot.

Habiendo definido todo lo anterior, se expondrán las conclusiones a las que se ha llegado en el proyecto, aportando ideas de cómo se podría mejorar el prototipo, así como aquellos aspectos en los que el desarrollo realizado puede ser de utilidad para futuros proyectos.

Seguidamente, se ha reservado un capítulo para los anexos necesarios, entre los que se incluirán los planos de los sólidos modelados, los códigos implementados y las hojas de datos de los elementos hardware utilidades.

Se cierra el documento con la bibliografía que ha sido necesaria, tanto para documentar el proyecto, como para el desarrollo del robot omnidireccional.

1.4. Desarrollo temporal

En este apartado se pretende enmarcar las distintas fases por las que ha pasado el proyecto, en el momento en el que se llevaron a cabo, ofreciendo de este modo una visión aproximada de qué se ha desarrollado en cada momento.

En primer lugar, y coincidiendo con el inicio del curso, se comenzó una primera fase de documentación, en la que se comprendieron los conceptos de mayor relevancia para el proyecto, tales como las ruedas y robots omnidireccionales, la plataforma Arduino o la electrónica que iba a ser necesaria. Paralelamente, se comenzaron a diseñar las piezas que constituirían el prototipo.

De este modo, a lo largo del primer cuatrimestre, se fueron imprimiendo los sólidos. Gracias a ello, se pudieron identificar defectos en los diseños, de modo que se generaban nuevos modelos que se adaptasen mejor a las necesidades del proyecto. Fabricar las piezas a partir de una impresora 3D es un proceso lento, pero que no impidió que se pudiese comenzar a realizar pruebas con los componentes que se instalarían posteriormente en el robot. En relación a esto último, se comprendió el funcionamiento de los motores paso a paso, realizando ensayos en el laboratorio en los que se pudo determinar la velocidad a la que estos podían girar, atendiendo a la alimentación con la que contaría el robot. Fue en este momento del desarrollo en el que se descartó la posibilidad de utilizar motores de corriente continua.

En el mes de noviembre ya se contaba con un primer modelo, controlado por un Arduino UNO a través de una comunicación serial. Se pudo identificar de este modo que las ruedas diseñadas respondían razonablemente bien en la práctica. Como consecuencia de haber obtenido estos resultados, se decidió sustituir el microcontrolador por un Arduino WeMos D1, introduciendo de este modo el concepto de la comunicación Wifi en el proyecto.

Se comprendió el funcionamiento de esta nueva placa, y se diseñó un sistema de control adaptado a ella, basado en la creación de un servidor web. Sin embargo, no fue posible continuar trabajando con el Arduino WeMos D1 ya que este no contaba con el número de pines que se necesitaban para el control de los motores. A pesar de que existían algunas posibilidades para solucionar este problema, se decidió que lo mejor para el proyecto era sustituir de nuevo el microcontrolador, instalando en este caso un Arduino MEGA.

En las siguientes semanas, se adaptó el sistema de control anterior al nuevo microcontrolador, al que se incorporó un módulo de comunicaciones Wifi ESP-01. De mismo modo, se mejoró el diseño del robot, modelando una estructura más robusta y resistente, además de aumentar el tamaño de esta para que hubiese espacio para alojar todos los componentes. De este modo, el nuevo prototipo estaba listo a principios de febrero. Los principales problemas que presentaba aquel modelo eran que la unión entre el eje de los motores y las ruedas no era precisa, y que los rodillos de las ruedas patinaban sobre las superficies lisas, afectando a la navegación del robot.

Atendiendo a las circunstancias expuestas, y de forma paralela a una mejora en el diseño de las ruedas, y a la sustitución del material de fabricación de los rodillos, se comenzó a estudiar la posibilidad de que el siguiente modelo de robot se controlase mediante un mando. Para ello, existía la posibilidad de utilizar el entorno ROS, con el que no se había trabajado hasta el momento por lo que fue necesario aprender a utilizar las herramientas que este ofrece.

A finales de febrero, tras haber intentado sin éxito, adaptar una serie de librerías para emplearlas en el modelo, se encontró una nueva posibilidad, modificar el paquete Ros_Arduino_Bridge, que permite establecer una comunicación entre un ordenador y un microcontrolador Arduino, para que el envío de los datos se realizase a través del módulo

ESP-01. A mitad de marzo ya se había conseguido avanzar con este nuevo modelo, pero no fue hasta la segunda semana de abril cuando ya se podían enviar mensajes desde el sistema ROS hasta el Arduino de forma inalámbrica, para lo cual fueron necesarios numerosos ensayos e incluso un cambio de protocolo en la comunicación. También, en aquel momento ya se había logrado que el sistema implementado en ROS reconociese un mando genérico que se conectase al ordenador, obteniéndose como resultado la velocidad a la que debía girar cada uno de los motores paso a paso.

Se comenzó entonces a desarrollar un código, basado en interrupciones software, para el control del movimiento de los motores, con el que se pudiesen aprovechar las ventajas que ofrecía el diseño de robot omnidireccional que se había fabricado. Junto a esto, se enlazaron los dos sistemas que se habían implementado en ROS, de modo que la información que partía del uso del mando llegase a la placa Arduino, donde se activaría el movimiento de los motores. Coincidiendo con el comienzo del mes de mayo, se habían logrado importantes avances en el sistema, pero aún había aspectos que se debían mejorar, lo cual demoró el proyecto una semana más.

El siguiente objetivo del proyecto era diseñar un modelo teórico para su posterior implementación, de la odometría del robot omnidireccional. No fue hasta mediados de mayo cuando se logró que el funcionamiento fuese el adecuado.

Por último, se implementó un sistema que permitía que el Arduino enviase la pose calculada del robot al ordenador, donde se procesa la información y se genera una representación gráfica en 2 dimensiones de la navegación del Robot, basada en el uso de la aplicación Rviz.

En relación a la redacción de este documento, se han ido documentando aquellos aspectos relacionados con la realización práctica del proyecto en el momento en el estos se realizaban. En cuanto a los contenidos teóricos, se han ido redactando de acuerdo a la disponibilidad de tiempo derivada de la carga de trabajo presentada por el resto de las asignaturas. Una vez finalizado el proyecto a nivel práctico, se comenzaron a redactar aquellos apartados que faltaban para la correcta documentación del proyecto.

Capítulo II. Marco teórico

En este capítulo se presentarán los distintos contenidos que se han podido trabajar en el desarrollo de este proyecto, desde una perspectiva teórica.

Se sentarán de este modo las bases necesarias para una posterior comprensión del modelo experimental que se ha fabricado. Atendiendo a este propósito, se mostrarán tanto aquellos aspectos que afectan al diseño realizado, como los que guardan relación al software que se implementará.

2.1. **Ruedas omnidireccionales**

Sobre la base de lo planteado en el primer capítulo de este documento, uno de los principales objetivos a los que se pretende atender en este trabajo, es lograr que el robot cuente con una gran maniobrabilidad. La idea fundamental consiste en que el robot debe disponer de los medios necesarios para poder realizar giros puros sobre sí mismo y traslaciones puras hacia las direcciones perpendiculares a cada una de las 3 ruedas con las que cuenta. Posteriormente, y aplicando técnicas de control se podrán describir movimientos más complejos derivados de las posibilidades que ofrece el diseño realizado.

Hecha la observación anterior, y con el propósito de lograr lo descrito, el diseño de las ruedas debía ser singular. En concreto se eligió utilizar las denominadas ruedas omnidireccionales, también conocidas como ruedas suecas o ruedas mecanum.

El prefijo “omni-“ significa todo, en este caso, omnidireccional haría referencia a todas las direcciones, y es que la principal característica distintiva en este tipo de piezas es que permiten realizar, tanto el movimiento habitual de una rueda, con el que se puede avanzar perpendicularmente al eje del motor, como realizar un movimiento paralelo a dicho eje. En el caso de este segundo tipo de movimiento, es necesario que cada una de las ruedas con las que cuenta el robot realicen el movimiento adecuado para que el desplazamiento resultante sea en la dirección y sentido deseados.

Figura 3. Modelo de rueda omnidireccional con rodillos a 45 grados

Figura 4. Modelo de rueda omnidireccional con una hilera de rodillos

Figura 5. Modelo de rueda omnidireccional con dos hileras de rodillos

En las anteriores imágenes, se muestran distintos diseños con los que es posible realizar las maniobras descritas. De entre estas opciones se ha escogido la última que se ha mostrado.

La elección de ese diseño radica en que es el más adecuado para las características del robot construido. Esta afirmación se basa en que el diseño A resulta más adecuado para un montaje en el que se incluyan cuatro ruedas con una colocación idéntica a la de un coche convencional, ya que, en esa posición, y haciéndolas girar de la manera adecuada, la fuerza resultante permite realizar desplazamientos paralelos a las direcciones de los ejes de los motores. Este comportamiento no parece muy eficiente y, además, la fabricación resulta más compleja. Con el segundo diseño la dificultad de fabricación se ve reducida en gran medida, pero presenta un gran inconveniente. El que queden espacios entre las pequeñas ruedas que entran en contacto con el suelo, favorece la posibilidad de que dichos espacios entren en contacto con la superficie sobre la que se mueve el robot, dificultando el movimiento paralelo al eje de dicha rueda omnidireccional. De todos los planteamientos presentados se deduce que el diseño más adecuado es el tercero, ya que

corrige los inconvenientes de los anteriores, y es el más adecuado a la disposición de las ruedas en el robot diseñado. La fabricación de la pieza no es compleja, lo cual es importante ya que en el presente trabajo se pretende obtener las mismas utilizando una impresora 3D. Además, el comportamiento de este tipo de rueda omnidireccional es adecuado para una implementación en la que se coloquen tres de ellas en cada uno de los lados de un triángulo equilátero, que es la estructura con la que cuenta el robot. Por último, al contar con dos filas de pequeñas ruedas, siempre una de ellas estará en contacto con el suelo dada la disposición de estas.

Una vez estaba clara la forma que debían tener las ruedas, se podía proceder a la realización del diseño en un programa de modelado de sólidos en 3D, para posteriormente poder imprimir las distintas piezas que fuesen necesarias. El software utilizado fue el Autodesk Inventor, en concreto la versión para estudiantes del mismo.

2.2. **Impresión 3D**

En este apartado se expondrán una serie de conceptos que se deben conocer para trabajar con esta tecnología con garantías.

2.2.1. Impresoras 3D

Las impresoras 3D son equipos que permiten generar cuerpos sólidos tridimensionales, diseñados a partir de un software CAD.

El uso de estos dispositivos en proyectos como el descrito en esta memoria es algo que no era posible hasta hace unos pocos años, y el haber contado con esta posibilidad, ha sido fundamental para poder fabricar un robot que se ajuste perfectamente a los requerimientos derivados de los componentes que se iban a instalar en la estructura. También ha sido de gran ayuda para fabricar el tipo de rueda que se necesitaba, debido a que no es común encontrar este tipo de piezas en las tiendas locales, dada la singularidad de estas.

Concretando aún más lo planteado en el párrafo anterior, aunque es cierto que la tecnología que utilizan las impresoras 3D actuales, la Deposición por material fundido (FDM), data de finales de los años 80 [2], no fue hasta el año 2009 cuando caducó la patente de esta tecnología. Este hecho fue el que provocó que el precio de las impresoras bajase sustancialmente al incorporarse al mercado nuevas empresas que comenzaron a fabricar estas máquinas, acercando este producto a un gran número de personas.

Desde aquel año, las impresoras 3D se están popularizando cada vez más, hasta el punto de que en la actualizad es común poder acceder a una de ellas en los centros enseñanza, como es el caso de las universidades, o incluso el contar con ella en un domicilio particular.

El proceso completo [3] hasta la obtención del objeto final, parte de la realización de un diseño en un software de modelado de sólidos en 3D de la pieza que se desea. El siguiente paso es generar el archivo con extensión .stl. En este proyecto, para estas

labores, se utilizó el Inventor Professional. El siguiente paso es elaborar un archivo que contenga los parámetros de impresión que se deseen, tales como la posición de la pieza, velocidad, temperatura de impresión, y otros muchos ajustes que se pueden determinar, lo cual se aclarará en el apartado destinado a explicar el Ultimaker Cura, que es el software que se ha empleado en este caso. El resultado es un archivo con extensión .gcode que contiene la información de que movimientos que debe realizar la impresora para conformar cada una de las capas que son necesarias para obtener el sólido final. Por tanto, y como se deduce de estas palabras, la impresora 3D trabaja depositando sobre una superficie lisa, que puede ser calefactable o no, la primera capa de material, y a partir de ella se van incorporando las sucesivas capas, una encima de otra hasta obtener el resultado final.

Hay que mencionar en este punto que existe una gran variedad de materiales con los que se puede producir la pieza, cada uno con diferentes características. Para la fabricación del robot presentado en este Trabajo de Fin de Grado se han utilizado dos materiales distintos, los cuales se presentarán en el siguiente apartado.

Una vez se ha expuesto en qué consiste esta tecnología, se puede presentar el que ha sido el equipo fundamental para la obtención de las distintas piezas que componen el robot omnidireccional, tanto la estructura como las ruedas, la WitBox 2 de la marca BQ.

Figura 6. Impresora 3D BQ Witbox 2

Se trata de una impresora completamente cerrada, con una puerta que permite el acceso a la superficie de impresión, y que ofrece la posibilidad de imprimir piezas de dimensiones de hasta 297 x 210 x 200 mm, lo cual era suficiente para los sólidos que se han fabricado para el robot tras dividir la estructura triangular en 3 partes.

2.2.2. Filamentos 3D

Existen distintos materiales que se pueden utilizar para la generación de sólidos en las impresoras 3D, cada uno de ellos presenta distintas características y precios. Además, se suelen ofrecer en un rango de colores muy variado. Algunos de estos materiales son el PLA, el ABS, el PETG y el Filaflex (filamento flexible). A parte de los mencionados, existen otros materiales cuya utilización es mucho menos común, y cuyo empleo es mayoritariamente en industria, en impresoras con un precio mucho más elevado. Como ejemplo de esto último se pueden mencionar materiales como resinas o metales.

En el caso de este proyecto se han utilizado dos de los materiales mencionados, el PLA y el Filaflex.

El PLA [4] es un material termoplástico. Es el más utilizado en el ámbito de las impresoras 3D que trabajan con la tecnología FDM. Es un componente ideal para la fabricación de la gran mayoría de piezas que se suele fabricar en este tipo de equipos. No es una excepción a esto el robot diseñado, y es por ello por lo que prácticamente la totalidad de las piezas que componen el mismo se han producido con PLA. Esto se debe a que presenta, junto a un precio que no es muy elevado, unas muy favorables propiedades físicas, ya que es un material muy duradero y resistente. Es posible conseguir piezas con una elevada resistencia estableciendo un relleno que no necesariamente tiene que ser excesivo. Los datos exactos que avalan esta afirmación dependen del proceso de fabricación seguido, es por ello por lo que no todos los objetos producidos con este material son exactamente iguales. En relación con lo anterior, existen variaciones que presentan unas mejores características, pero con un mayor coste.

Figura 7. Bobina de PLA negro

Por otro lado, el filaflex [5] es un material que satisface unas necesidades muy específicas, y que tiene un precio sustancialmente superior. Es un filamento con base de poliuretano mezclada con ciertos aditivos que, al contrario que el PLA, es un material

blando y menos resistente. Por otro lado, y como se intuye por el nombre, es flexible, lo cual es su mayor distinción y virtud. Es la mejor opción cuando se quiere fabricar una pieza que sea deformable, pero que recupere rápidamente su forma. Del mismo modo que para el caso anterior, los datos específicos varían en función del modelo concreto que se utilice. Es importante mencionar en este punto, que la dificultad para fabricar piezas con filaflex es bastante superior. Esto se debe, sobre todo, a que la velocidad de impresión hay que reducirla considerablemente, a que resulta más complicado establecer unos ajustes con los que se logre un buen acabado en el resultado final, y a que no es posible activar la opción de retracción del extrusor, lo cual es un problema ya que implica que cuando este se desplaza no absorbe el material que hay en la boquilla, imposibilitando de este modo el poder imprimir varias piezas de forma simultánea, y al mismo tiempo aumentando las probabilidades de que el resultado final contenga apreciables imperfecciones. Por otro lado, otro problema que se puede dar al trabajar con este tipo de filamento, aunque este no se manifestó en la realización de este proyecto, es que existe la posibilidad de que el filamento se atasque en el extrusor, lo cual es común para algunos modelos de impresoras 3D.

En el caso de las ruedas diseñadas para el robot, se necesitaba que el material que entra en contacto con el suelo presentase una buena adherencia, que impidiese que estas patinasen cuando el robot estuviese en movimiento. El filaflex satisface este requerimiento al tener un alto coeficiente de fricción.

Figura 8. Bobina de filaflex amarillo

2.2.3. Ultimaker Cura

Resulta razonable comenzar este apartado mencionando que existe una gran diversidad de programas destinados a la preparación de los modelos 3D, que posteriormente se cargarán en la impresora. El Ultimaker Cura ha sido el software utilizado en este Trabajo de Fin de Grado, ya que es una herramienta gratuita, fácil de utilizar, y que ofrece un gran número de posibilidades, las cuales se pasan a describir a continuación.

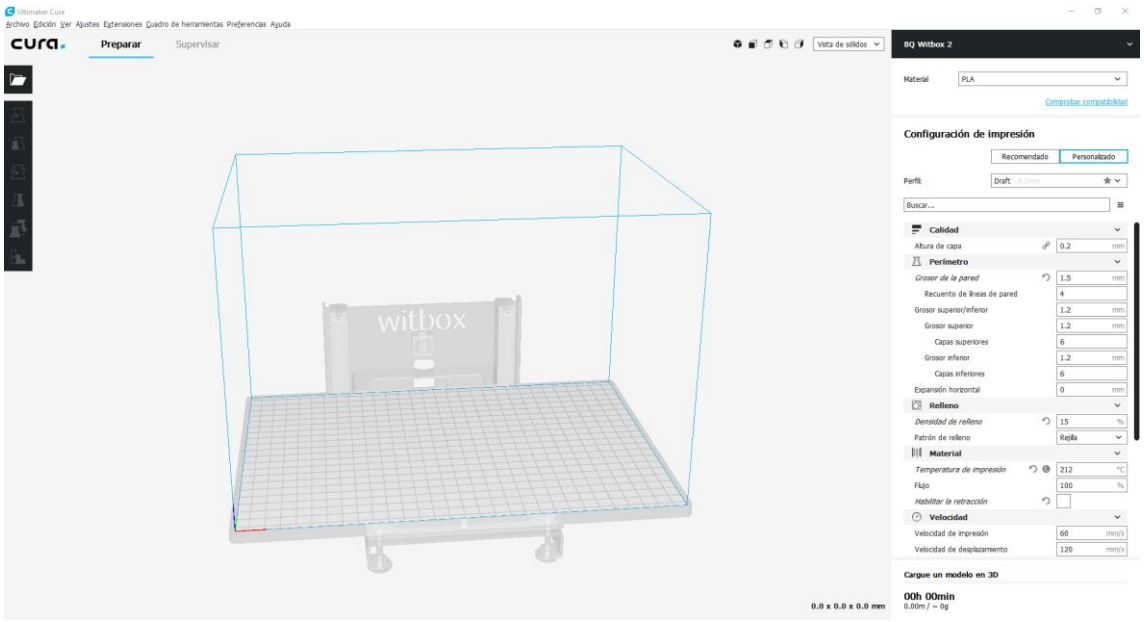


Figura 9. Ventana de trabajo Ultimaker Cura 3.5.0

En primer lugar, se ha de cargar el modelo 3D que se desea preparar, tras lo cual ya es posible seleccionar en qué parte de la superficie de impresión se quiere ubicar la pieza, la orientación, y también es posible reescalar el sólido, modificando su tamaño sin alterar las proporciones.

En el panel de configuración de impresión se ofrecen una serie de ajustes, los cuales se pueden modificar según requiera el usuario, ya que los valores escogidos dependerán de muchos factores, además del material con el que se vaya a realizar la impresión. En este apartado se explicarán tanto aquellos ajustes considerados por el programa como básicos, como otros que se han modificado adicionalmente para obtener un mejor resultado.

En primer lugar, la altura de capa determina indirectamente la resolución de la pieza final, y con ello la calidad del acabado. Cuando menor sea el valor, mayor será el grado de detalle logrado en la pieza final. En este caso, para cualquiera de los sólidos que ha sido necesario fabricar, ha sido suficiente con una altura de capa de 0.15 mm.

El siguiente desplegable del software, el cual contiene los ajustes relacionados con el perímetro, no es necesario que sea modificado en exceso respecto a los parámetros por defecto. En el caso concreto del número de capas inferiores, si es conveniente aumentar el número de ellas para evitar que la pieza se pueda romper al tratar de separarla de la superficie de impresión. En el caso particular de este proyecto se decidió aumentar al doble del valor predefinido, es decir unas 8 capas en total.

La densidad de relleno es uno de los parámetros que más influencia tiene sobre el resultado final. Determina, como su nombre indica, la densidad de la pieza que se fabrique, y se trabaja con porcentajes. Para piezas impresas en PLA, con un 15% o 20% ya es más que suficiente para que el resultado sea una pieza dura y resistente. En el caso del filaflex es necesario aumentar bastante este valor para lograr que el resultado sea

bueno. Por tanto, es necesario establecer un equilibrio entre la facilidad de impresión y la flexibilidad que se quiere obtener, ya que con un porcentaje muy alto el resultado es una pieza poco deformable. En el proyecto se utilizó este material para la parte en contacto con el suelo de las ruedas, y con un valor del 40% se consiguió un balance adecuado. En cuento al patrón de relleno, este no tiene un efecto muy significativo en el resultado obtenido.

La temperatura de impresión es otro de los factores fundamentales a la hora de utilizar una impresora 3D. El valor seleccionado dependerá de lo que indique el fabricante para cada material y variedad de este. El PLA se imprimió a una temperatura de 212 ºC y en el caso del filaflex, para conseguir que las capas presentasen pocas imperfecciones, fue necesario aumentar esta temperatura hasta los 240 ºC . En cuanto a la habilitación de la retracción, que consiste en que el extrusor retrae el filamento cuando se desplaza por una zona en la que no tiene que depositar material, y que consigue que el acabado de la pieza sea destacablemente mejor. Este ajuste fue posible utilizarlo para el PLA, pero no para el filaflex, ya que al ser este un filamento flexible, existe la posibilidad de que se atasque en la zona en la que se empuja el material.

La velocidad de impresión y de desplazamiento son las que más directamente fijan el tiempo que va a tardar el proceso hasta la obtención del sólido completo. El rango de valores entre los que se puede escoger depende principalmente de la impresora que se vaya a utilizar y del material. En el caso del presente proyecto, para las piezas fabricadas con PLA se podía trabajar con una velocidad de impresión de 60 mm/s y de desplazamiento de 100 mm/s, mientras que para el filaflex no era posible obtener un resultado bueno si no se reducía la velocidad de impresión hasta los 20 mm/s y la de desplazamiento a 50 mm/s.

La opción de generar soporte permite poder realizar formas en las que haya que depositar material en altura, es decir, sin que debajo haya material. Para ello, la impresora crea una estructura debajo del espacio donde se quiere imprimir que sea fácil de retirar posteriormente. Hay que mencionar que en la práctica no es tan sencillo retirar este material sobrante, por ello es mejor evitar tener que utilizar esta opción.

El resto de los parámetros no se modificaron respecto a los valores predefinidos que ofrece el software para cada material, y en el caso del tipo de adherencia de la placa de impresión, el cual, si se elige alguno de los ofrecidos, se favorece el que, en el inicio del procedimiento, las primeras capas no se despeguen de la superficie de impresión. En el caso del filaflex no es necesario utilizar este ajuste, y para el PLA, habiendo previamente impregnado la superficie de impresión con una fina capa de laca, con seleccionar el tipo de adherencia denominado falda es suficiente, ya que este crea una primera deposición de material alrededor de la pieza, sin llegar a tocarla, pero que permite que cuando se comience con lo que realmente es la pieza ya no haya imperfecciones en la colocación del material.

De forma adicional a todo lo expuesto en este apartado, es posible mencionar que el Ultimaker Cura ofrece también otro tipo de información de gran utilidad, como es una

previsión del tiempo que llevará la impresión del sólido, la cantidad de material, medida en gramos, que se empleará, y también se permite realizar una vista de capas en la que poder visualizar cómo la impresora va a componer el modelo internamente, o las estructuras que va a crear si éstas son necesarias y se ha habilitado la opción correspondiente.

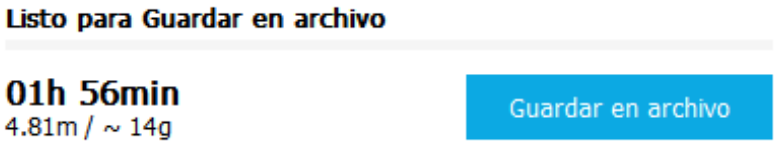


Figura 10. Cuadro de información de consumo y tiempo estimado del Ultimaker Cura

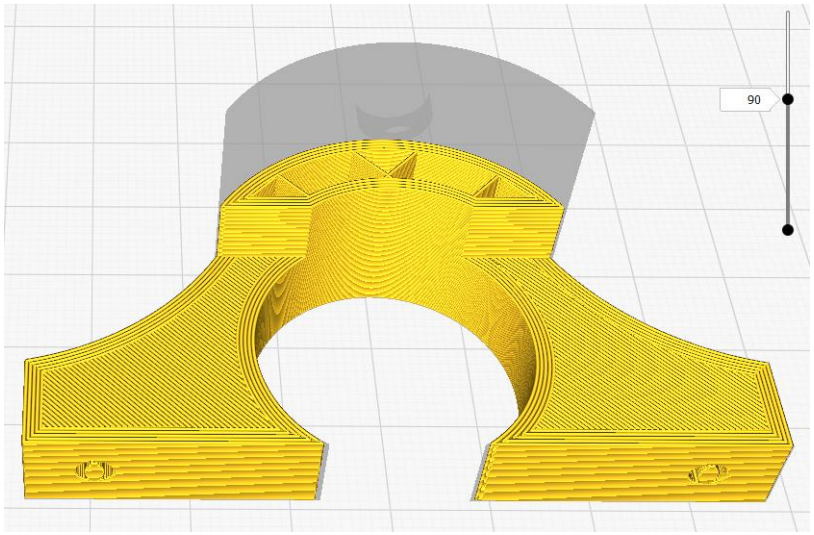


Figura 11. Vista de capas en el Ultimaker Cura

Antes de concluir este apartado, resulta conveniente explicar que la orientación en la que se coloque la pieza en la superficie de impresión es un factor muy importante. Eligiendo una buena colocación de una pieza se puede reducir considerablemente la cantidad de material que se necesita, ya que, en aquellas zonas en las que no haya material por debajo, la impresora creará una estructura, de modo que se pueda depositar el material de impresión a la altura que se desea. En la siguiente imagen se puede observar, a la izquierda una buena colocación, y a la derecha, una disposición en la que el consume de material es mayor.

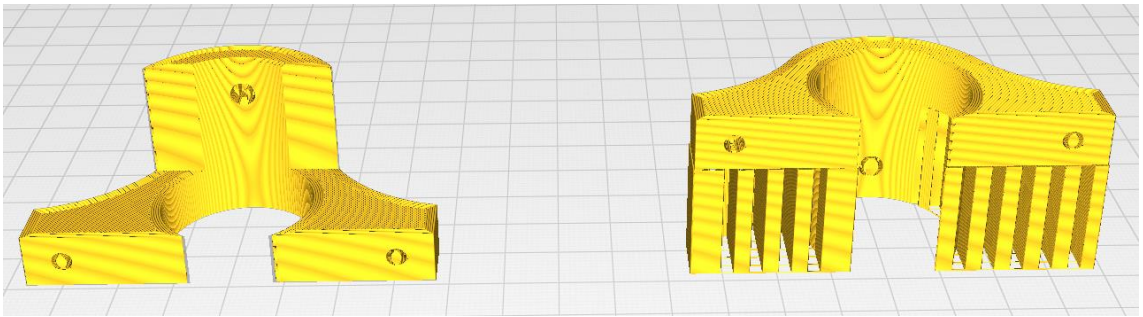


Figura 12. Correcta colocación de las piezas en el Ultimaker Cura

2.3. **Autodesk Inventor Professional**

Es una herramienta software para el diseño de sólidos en tres dimensiones, la cual era necesaria para el modelado de las distintas piezas que constituirían el robot. Existen numerosos programas que permiten realizar este tipo de trabajo, no obstante, existían algunas razones por las que se seleccionó esta aplicación de entre las posibles.

No se trata de una herramienta gratuita, sin embargo, Autodesk ofrece licencias de sus productos para estudiantes, que salvo por algún obstáculo respecto a la versión de pago, permite el acceso a todas las posibilidades que ofrece el software. Por otro lado, es un programa que ya se había utilizado en algunas asignaturas de la carrera, por lo que ya era conocido el manejo del mismo.

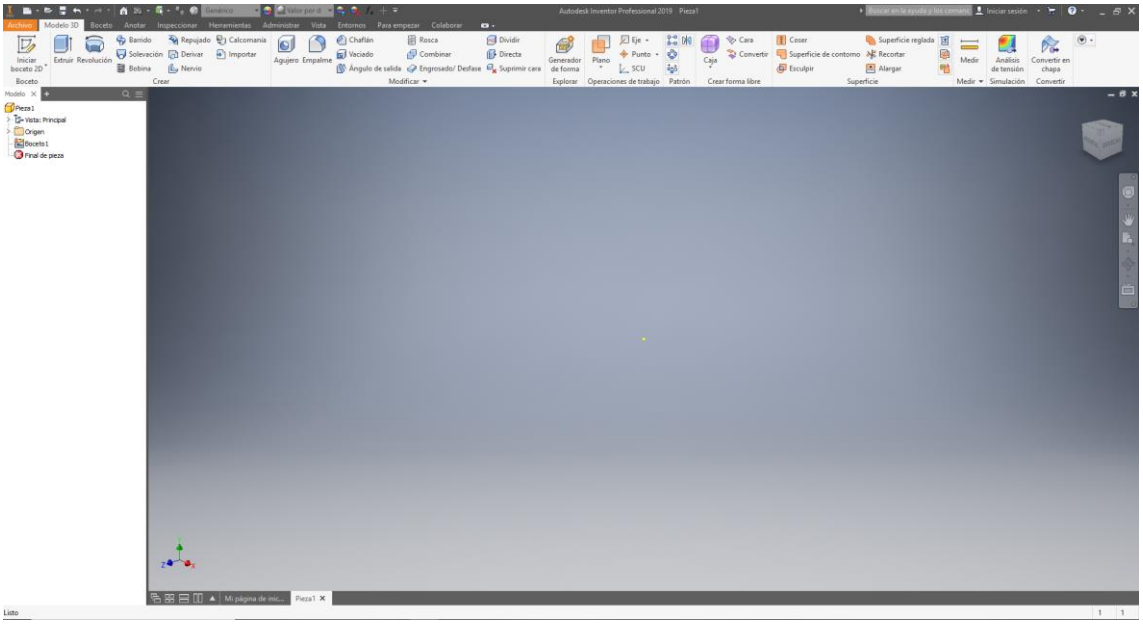


Figura 13. Entorno de trabajo Autodesk Inventor Professional

Junto a todo lo anterior, ofrece una gran variedad de herramientas para poder realizar un diseño que se ajuste con una muy alta precisión al sólido diseñado. De esta forma fue posible realizar con una gran libertad cada una de las piezas que eran necesarias para conformar el robot.

El modo de trabajo se basa en realizar dibujos en 2D mediante dibujo libre o ajustando formas predefinidas (como circunferencias o arcos), a partir de los cuales realizar operaciones que le agreguen volumen. Las operaciones más utilizadas han sido la extrusión, que consiste en dar volumen al cuerpo de forma perpendicular al dibujo realizado, la revolución, que permite crear un sólido a partir de la rotación del dibujo 2D en torno a un eje, y el empalme, el cual permite redondear los bordes de las piezas. En el siguiente capítulo de la memoria se explicará con más detenimiento la creación de las distintas piezas que han sido necesarias.

El inventor también permite crear planos en los que mostrar la información de las medidas de las piezas diseñadas y las vistas que se deseen mostrar. Esta herramienta se utilizará para mostrar, en el tercer capítulo, los sólidos diseñados en este proyecto.

De la misma manera que en la mayoría de software del ámbito de la ingeniería, existe la posibilidad de realizar un estudio, previo a la fabricación de la pieza, con el que poder identificar fallos en el diseño para corregirlos antes de proceder a la realización práctica. En este caso, es recomendable utilizar la herramienta de ensamblaje, con la que cerciorar que las piezas que están destinadas a ir juntas encajen como estuviese previsto. Por ejemplo, esto ha sido de gran utilidad en este trabajo para ratificar que las muescas, dispuestas en los distintos sólidos para que puedan encajar entre sí, estaban realizadas correctamente. Para ello se utiliza la herramienta "Restringir", la cual permite seleccionar aquellas superficies que se deben juntar.

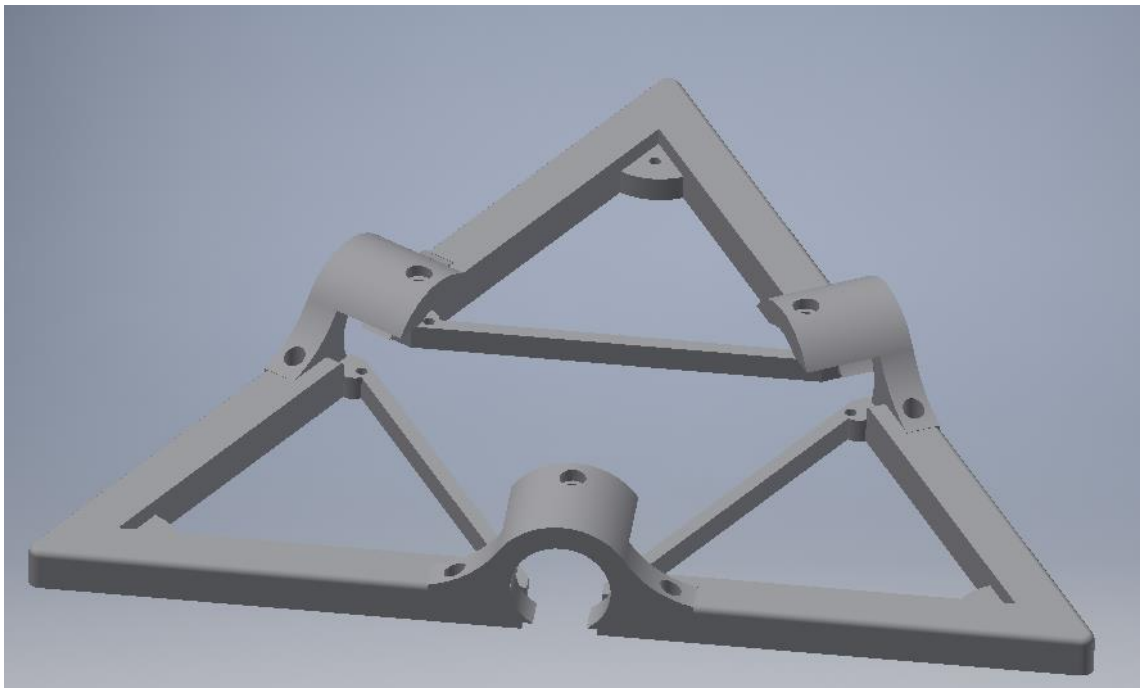


Figura 14. Ensamblaje en Inventor

El inventor Professional ofrece muchas otras posibilidades, pero en este apartado se ha querido dar a conocer aquellas que más se han utilizado para los diseños de las distintas piezas.

2.4. **Arduino**

Arduino [6] es una plataforma destinada a la implementación de electrónica, y que se puede utilizar como nexo entre multitud de sensores y actuadores para que estos trabajen en conjunto. Resulta de gran relevancia mencionar que esta placa se basa en la utilización de hardware y software libre, lo cual posibilita que existan multitud de variantes de tarjetas que presentan características diferentes a las de los modelos originales.

La plataforma ha ido reuniendo una gran importancia dentro del ámbito de la ingeniería, y en especial para el caso particular de la electrónica, ya que ofrece un medio en el que se pueden implementar un gran número de proyectos de diferentes características, tales como el presentado en este trabajo. Todo lo mencionado, junto a un

bajo precio en el caso de algunos modelos, son las razones del creciente éxito de estas placas, las cuales ya se utilizan tanto en pequeños proyectos como en otros más ambiciosos.

En relación con lo anterior, y para el caso particular de este proyecto, gracias a la incorporación de una placa Arduino ha sido posible integrar los elementos fundamentales del robot, lo cual se refiere al módulo de comunicación a través de WIFI y el control de los motores paso a paso.

Arduino también es un medio que resulta bastante adecuado para que cualquier persona, con interés por la electrónica, incluso desde tempranas edades, se introduzca en este entorno. Esta afirmación se basa en que el entorno de programación está pensado desde el punto de vista de ser lo más simple y claro para el usuario.

Una característica que ha sido de gran utilidad es el que Arduino sea multiplataforma, lo cual ha hecho que sea posible desarrollar parte del código del proyecto tanto en un entorno Windows como en una distribución Linux, dependiendo de las posibilidades del lugar donde se estuviese trabajando.

Como se ha mencionado en párrafos anteriores, existen multitud de placas Arduino, las cuales utilizan distintos microcontroladores y microprocesadores, presentan distintas características e incluso algunas incorporan módulos adicionales. En este proyecto se ha trabajo con 3 modelos diferentes, lo cual se debe a que, en cada momento del proyecto, las necesidades a satisfacer han sido diferentes. En concreto se han utilizado las placas Arduino UNO, WeMos y MEGA, siendo esta última la que finalmente forma parte de la electrónica del robot.

2.4.1. Arduino IDE

Es el entorno de desarrollo de código abierto de Arduino. En él se han elaborado todos los programas para las placas Arduino que han sido necesarios a lo largo del transcurso del proyecto. El lenguaje de programación que se utiliza está basado en C++, de hecho, es posible utilizar los comandos estándar de este lenguaje. El entorno también permite, una vez preparado el código, subirlo a la placa Arduino que se utilice, para lo cual se ha de disponer una conexión física entre la tarjeta y el ordenador, la cual se hace mediante una conexión USB.

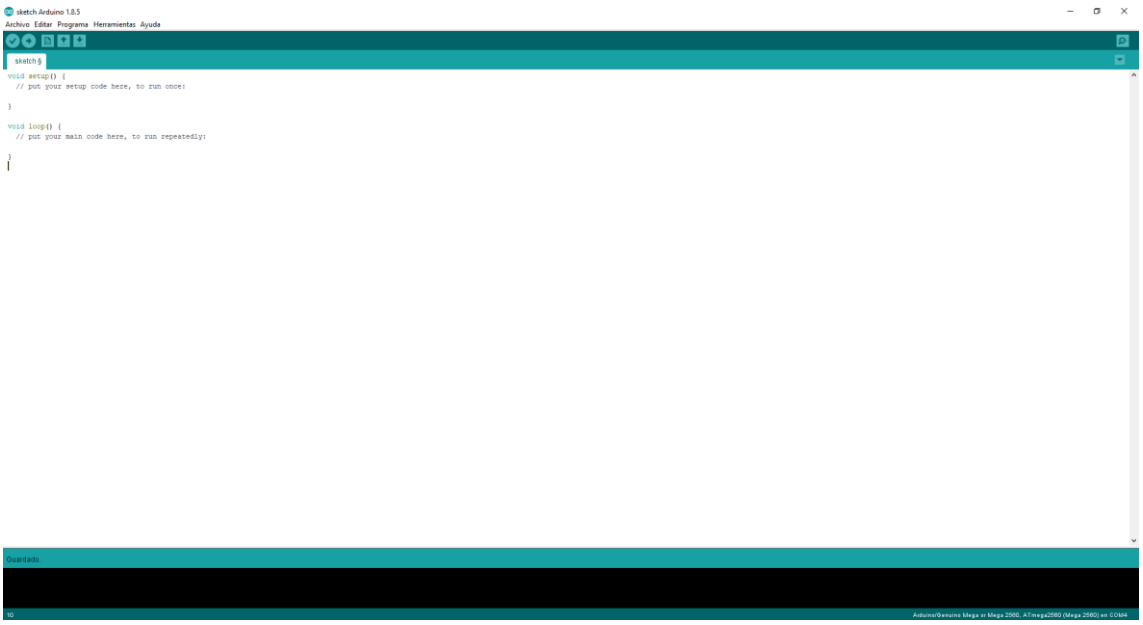


Figura 15. Entorno de trabajo Arduino IDE 1.8.5

Se puede acceder a las distintas funciones, variables y estructuras que incluye el lenguaje en la web de la plataforma [7].

Una vez se ha cargado el código en la memoria del microcontrolador, se puede utilizar el Arduino sin que esté conectado al ordenador, alimentándolo con una pila, una batería o a través de algún puerto de alimentación que incorpore la placa. No obstante, en ciertas ocasiones resulta muy conveniente mantener la conexión mediante USB al ordenador, ya que el Arduino IDE incorpora un monitor que permite visualizar una comunicación serial entre el microcontrolador y el ordenador, lo cual es de gran utilidad para comprobar el funcionamiento del código, o para buscar errores. Para poder hacer esto último se ha de incorporar la instrucción Serial.print(), junto con el mensaje que se quiere enviar, en aquella zona del código en la que se desee hacer depuración. Este mensaje se podrá leer desde el monitor.

Este software se puede utilizar tanto en Windows como en Linux y Mac, siendo estos dos primeros los entornos en los que se ha trabajado.

En este Trabajo de Fin de Grado, se han utilizado tres modelos distintos de placas Arduino, el UNO, el MEGA y el WeMos, siendo necesario para este último, añadir el siguiente enlace en el panel de gestor de URLs adicionales de tarjetas para poder trabajar con esta placa Arduino:

`http://arduino.esp8266.com/stable/package_esp8266com_index.json.`

Esto se debe a que el Arduino WeMos no está incluido por defecto en el entorno de desarrollo.

Otra característica adicional que incluye el software analizado en este apartado es un listado de códigos de ejemplo, lo cuáles pueden ser de gran utilidad para conocer las

posibilidades que ofrecen las tarjetas Arduino por defecto, y también las de las librerías que se añadan al entorno.

El proceso de carga del código en el microcontrolador es tan sencillo como conectar la tarjeta al ordenador mediante el cable USB adecuado (en el caso del WeMos es un cable MicroUSB, y para el UNO y MEGA se utiliza un USB tipo B), e indicar en el programa la placa que se va a utilizar, el procesador que esta incorpora, el puerto del ordenador al que se ha conectado y seleccionar el programador, que por defecto es el AVRISP mkII. Prácticamente todos estos ajustes se establecen solos una vez que el ordenador detecta la tarjeta que se ha conectado. Antes de que el programa se transfiera al Arduino, este es verificado y compilado si estas acciones no se han realizado manualmente en algún momento previo.

De los anteriores planteamientos se puede deducir que el Arduino IDE es un entorno de programación muy adecuado para las necesidades de proyectos como el explicado en esta memoria. Además, gracias a la particularidad de ser multiplataforma, se han podido realizar ensayos con el robot en múltiples espacios, favoreciendo de este modo el poder avanzar con el desarrollo del prototipo.

2.4.2. Arduino UNO

Esta fue la primera placa que se utilizó en el proyecto para las primeras pruebas de funcionamiento de los componentes que se instalarían en el robot, tales como los motores paso a paso o el módulo de pantalla LCD. Cumplía con los requerimientos para poner en funcionamiento, de forma simultánea, los tres motores que contiene el robot, ya que cuenta con suficientes pines de salida digitales. Por esta razón, esta fue la placa con la que se implementó el primer modelo de robot, en el que se controlaban una serie de movimientos predefinidos que podía realizar el robot, mediante una comunicación serial. No es el modelo más básico de la marca, pero si el más conocido y el más fácil de encontrar en las tiendas.

No es necesario realizar soldaduras para utilizar los pines de la placa ya que cuenta con zócalos en los que se puede conectar cómodamente los terminales de los cables tipo DuPont.

Si se realiza un análisis más técnico del Arduino UNO [8], se puede mencionar que se trata de un microcontrolador basado en el ATmega328P, que cuenta con 14 pines de entrada y salida digital, de los cuales, 6 se pueden utilizar como salida PWM. También incorpora 6 entradas analógicas, un cristal de cuarzo de 16 MHz, un conector USB tipo B, un puerto de alimentación tipo Jack de 9 Voltios, un ICSP (programación serial en circuito) y un botón de reinicio, el cual, al ser presionado, inicia una nueva ejecución el programa.

Figura 16. Placa Arduino UNO

En cuanto a las especificaciones técnicas, estas se detalla en la página web oficial de Arduino y son las que se muestran en esta tabla.

Microcontrolador	ATmega328P
Voltaje de operación	5 V
Voltaje en pin de entrada recomendado	7 - 12 V
Voltaje en pin de entrada limites	6 - 20 V
Pines digitales	14 (6 de ellos con posibilidad de utilizar señales PWM)
Pines de entrada analógica	6
Corriente en los pines	20 mA (Corriente continua)
Corriente en el pin de 3.3 V	50 mA (Corriente continua)
Memoria Flash	32 KB
SRAM	2 KB
EEPROM	1 KB
Velocidad del reloj	16 MHz
Largo	68.6 mm
Ancho	53.4 mm
Peso	25 g

Tabla 1. Especificaciones del Arduino UNO

Finalmente hubo que descartar este microcontrolador cuando surgió la necesidad de dotar al robot de la capacidad de comunicarse a través de una red Wifi, lo cual no era posible con el Arduino UNO. De este modo se comenzó a trabajar con el Arduino WeMos D1.

2.4.3. Arduino WeMos D1

Se trata de un microcontrolador [9] basado en el ESP-8266EX, que no forma parte del catálogo principal de la marca, pero que es compatible con el resto de las herramientas de esta, y que cuenta con 11 pines de entrada y salida digitales que operan a 3.3 V. Todos ellos permiten trabajar con interrupciones, con señales PWM y con el protocolo I2C, excepto el pin D0. Adicionalmente, cuenta con una entrada analógica con un voltaje

máximo de 3.2 V, también con un conector Micro USB y un puerto de alimentación tipo Jack que puede suministrar de 9 a 24 V.

Se comenzó a utilizar este modelo de Arduino en el proyecto cuando fue necesario incorporar una comunicación WIFI en el sistema. El WeMos D1 resultaba muy adecuado para este propósito debido a que, manteniendo un tamaño y disposición de pines muy similar al Arduino UNO, incorporaba un chip basado en el ESP-8266, en concreto el ESP-12, el cual permitía que se trabajase con este tipo de comunicación.

En este modelo tampoco es necesario realizar soldaduras para utilizar los pines de la placa, ya que cuenta con zócalos destinados a ello.

Figura 17. Placa Arduino WeMos D1

En cuanto a las especificaciones técnicas, estas se detalla en la siguiente tabla:

Microcontrolador	ESP-8266 (Modelo ESP-12)
Voltaje de operación	3.3 V
Pines digitales	11
Pines de entrada analógica	1 (Máximo voltaje de entrada de 3.2 V)
Velocidad del reloj	Programable a 80 MHz o a 160 MHz
Memoria Flash	4 MB
Largo	68.6 mm
Ancho	53.4 mm
Peso	25 g

Tabla 2. Especificaciones del Arduino WeMos D1

El problema principal que presentaba esta placa es que no disponía de un número de pines suficiente para controlar los 3 motores paso a paso, lo cual se debía a que el módulo ESP-8266 utilizaba parte de los pines disponibles en su funcionamiento.

No obstante, existían algunas opciones con las que hubiese sido posible continuar trabajando con esta placa, como emplear un circuito de puertas lógicas con las que controlar, mediante los pines disponibles, un número mayor de señales, o lo que es más común, emplear registros de desplazamiento TTL. Estos dispositivos permiten realizar una conversión serie a paralelo o viceversa, de los datos. En este caso, se tendría que

utilizar un registro de desplazamiento con una entrada serie, de modo que se almacenen los valores en unos biestables internos del integrado (cada uno guarda un bit), obteniendo una salida paralela en la que se cuenta con los valores necesarios para cada uno de los pines de los controladores de los motores. El tratamiento de los datos es secuencial, necesitando de este modo una señal de reloj externa que sincronice los datos. Con este método se podría contralar cada motor incluso con un único pin de la placa Arduino.

Sin embargo, a pesar de que existía esta posibilidad de utilizar lógica externa, un cambio de placa a una nueva con la que se contase con más recursos, resultaba más cómodo y sencillo en cuanto a la implementación del control del robot. De este modo, y de igual forma que con el Arduino UNO, hubo que descartar la posibilidad de emplear esta placa en el robot.

2.4.4. Arduino MEGA

Es un microcontrolador [10] basado en el ATmega2560, el cual está pensado para proyectos en los que se necesita un número de pines mayor al ofrecido por tarjetas como las presentadas en los apartados anteriores. En concreto, cuenta con 54 pines de entrada y salida digitales, de los cuales 15 se pueden utilizar con señales PWM. Incorpora también 16 entradas analógicas, 4 puertos para UART y un oscilador de 16 MHz. Al igual que el Arduino UNO, este trabaja a 5 V e incluye un conector USB tipo B, un circuito programador serial (ICSP), la posibilidad de alimentar la placa mediante un conector Jack y un botón de reinicio. Cabe mencionar que incluye zócalos para la conexión de los cables.

Se incorporó este modelo al robot como sustitución a las placas anteriores para solucionar el problema de la falta de pines para el control de los motores paso a paso. El punto negativo del Arduino MEGA respecto al WeMos D1 es que, al contrario que el segundo, no incluye un chip para comunicación WIFI, lo cual supuso que fuese necesario añadir un módulo externo, el ESP-01. Esta es la placa con la que cuenta el modelo final del robot.

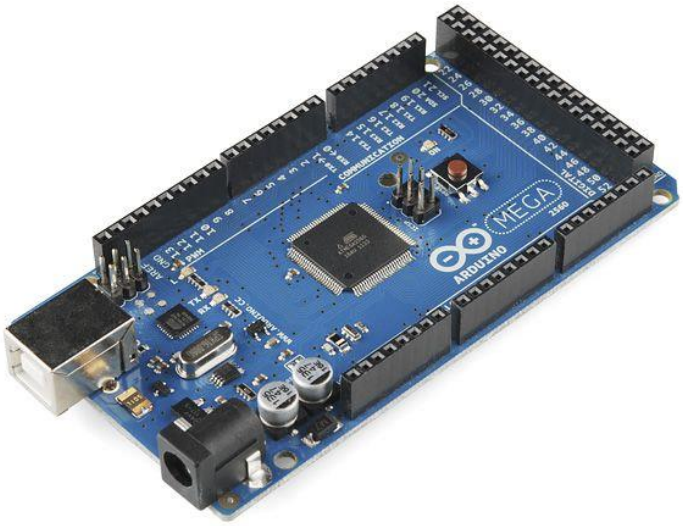


Figura 18. Placa Arduino MEGA

En cuanto a las especificaciones técnicas, estas se detallan en la página web oficial de Arduino, y son las que se muestran en la siguiente tabla:

Microcontrolador	ATmega2560
Voltaje de operación	5 V
Voltaje en pin de entrada recomendado	7 - 12 V
Voltaje en pin de entrada limites	6 - 20 V
Pines digitales	54 (15 de ellos con posibilidad de utilizar señales PWM)
Pines de entrada analógica	16
Corriente máxima en los pines	20 mA (Corriente continua)
Corriente máxima en el pin de 3.3 V	50 mA (Corriente continua)
Memoria Flash	256 KB
SRAM	8 KB
EEPROM	4 KB
Velocidad del reloj	16 MHz
Largo	101.52 mm
Ancho	53.3 mm
Peso	37 g

Tabla 3. Especificaciones del Arduino MEGA

2.5. **Otros componentes hardware**

En este apartado se mostrar los componentes electrónicos que, junto al microcontrolador, se han instalado en el robot omnidireccional.

2.5.1. Alimentación y regulador de tensión MP1584EN

En el robot descrito en este trabajo, la comunicación se realiza de forma inalámbrica mediante un módulo WIFI. Junto a esto, si la alimentación del sistema se realiza mediante baterías, se consigue eliminar cualquier tipo de unión física entre el robot y otro equipo.

En concreto, se ha optado por una batería de polímero de litio (LiPo) de dos celdas de 3.7 V, que ofrecen un nivel de tensión total de 7.4 V, y capacidad de 1500 mAh. La tasa de descarga es de 25 C en el modo de funcionamiento continuo y de 40 C en ráfaga.

Se trata una batería de baja capacidad, pero es suficiente para poner en funcionamiento el robot que se ha fabricado. Es importante mencionar que en este tipo de baterías de polímero de litio, no se debe descargar la misma por debajo de 3.5 V ya que se podrían dañar las celdas de forma irreversible.

Las primeras pruebas de funcionamiento del robot se realizaron alimentando la placa Arduino mediante un cable conectado por USB al ordenador, siendo dicha placa la que alimentaba a los controladores de los motores y al resto de componentes. Una vez se había logrado que el funcionamiento del sistema fuese el correcto se incorporó la batería al sistema. Para ello, el terminal negativo se conectó a la tierra de la electrónica del robot y el positivo se llevó a un interruptor, con el que se controlará la alimentación de todo el

sistema. Tras eso, se unió el otro terminal del interruptor a la entrada de un regulador de tensión. En concreto se ha instalado el modelo MP1584EN.

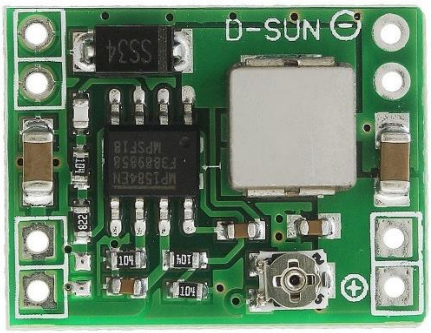


Figura 19. Regulador de tensión MP1584EN

Este regulador es un convertidor reductor muy interesante en cuanto a que permite, trabajando en el rango recomendado de operación, para una tensión de entrada que puede ser desde 4.5 V hasta 28 V, obtener una salida de entre 0.8 V y 20 V. La salida se regula a partir de un potenciómetro que viene incorporado, el cual se puede ver en la imagen que se ha presentado del componente. Para el caso estudiado se ajustó para obtener una salida regulada de 5 Voltios, con la que se alimenta toda la electrónica del robot. La frecuencia de operación también es programable, y se puede ajustar desde 100 kHz hasta 1.5 MHz.

Tiene unas dimensiones de 22 mm x 17 mm x 4 mm, y la temperatura de funcionamiento está comprendida entre – 45 ºC y 85ºC. Otro dato importante es que suministra una corriente de 3 Amperios como máximo a la salida.

Como ocurre en todos los reguladores de tensión, estos tienen una eficiencia. En el caso del MP1584EN esta depende de la tensión de entrada y de la corriente de salida según la siguiente curva.

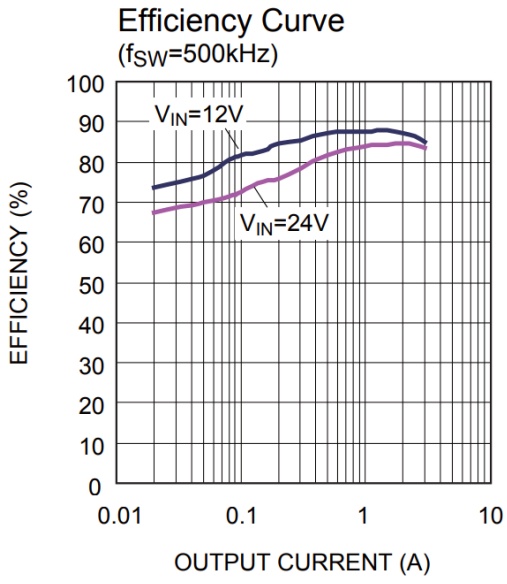


Figura 20. Curva de eficiencia del regulador de tensión

Con esa tensión de salida se alimentó la placa Arduino, empleando para ello el pin destinado a tal fin, es decir, el de 5 Voltios. También se podría alimentar la placa por el pin Vin, pero en ese caso la tensión tendría que pasar por el regulador interno de la placa Arduino. También se conectó a los pines positivos de alimentación de cada uno de los tres controladores de los motores paso a paso. Con esta misma tensión se alimentan el resto de los componentes presentes en el robot como es el caso del módulo para comunicación WIFI y la pantalla LCD.

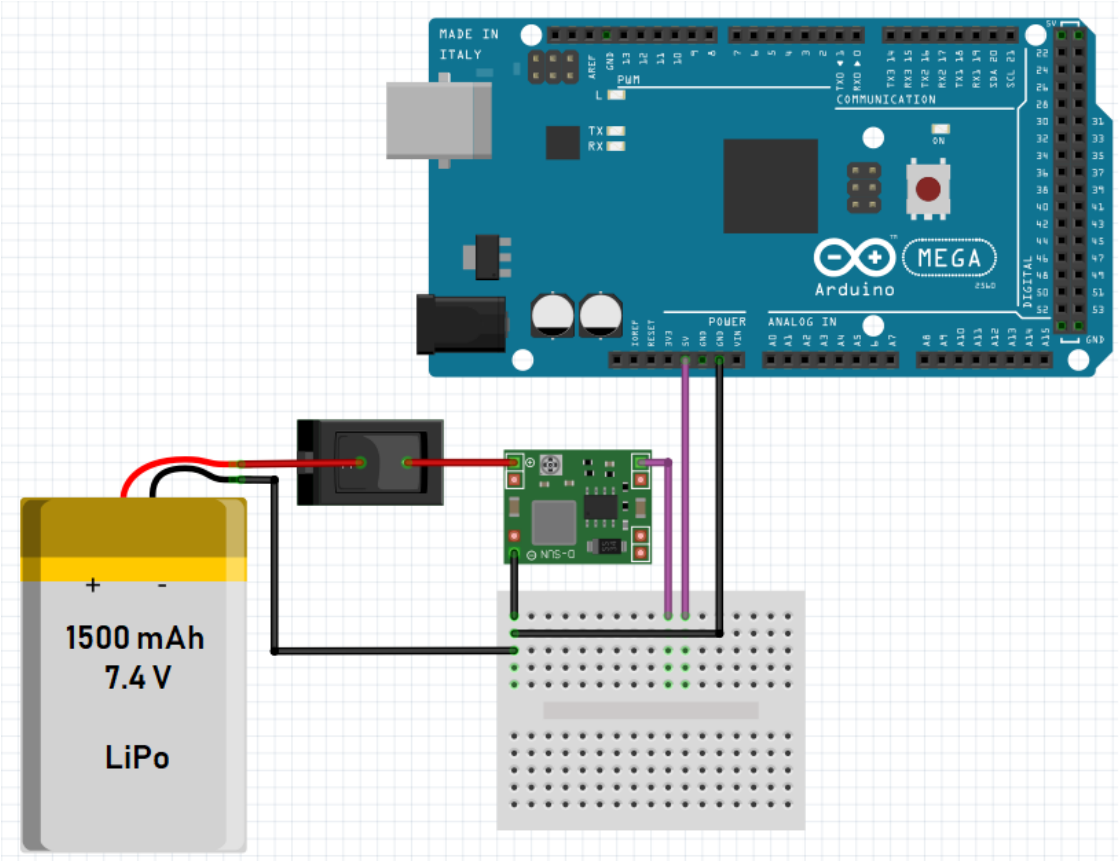


Figura 21. Esquema de la alimentación de la placa Arduino MEGA

2.5.2. Módulo LCD IIC / I2C 1602

En el marco de los aspectos que se están describiendo en este apartado, resulta oportuno mostrar el que ha sido el medio elegido para mostrar información sobre el robot directamente al usuario, una pantalla LCD que informa de que el robot está listo para ser utilizado. Un componente como este era necesario ya que al encender el robot este tarda un tiempo en establecer la comunicación con el punto de acceso Wifi, y era necesario que se mostrase al algún modo cuando ha concluido este proceso.

El módulo cuenta con una pantalla de 2.7 pulgadas, en la que se pueden mostrar dos filas de datos, cada una con hasta 16 caracteres. Una de sus principales ventajas es que incorpora luz de fondo, y se puede controlar, mediante un potenciómetro que en este modelo viene incorporado, el nivel de contraste. Está preparada además para funcionar a un nivel de tensión de 5 Voltios.

La mayor virtud que presenta este modelo es que soporta el protocolo I2C, cuyas siglas significan circuito interintegrado. Es un protocolo de comunicación serial basado en el SPI y el UART, que envía la información en una única señal, un bit detrás del otro sincronizados por una señal de reloj. Se trata por tanto de un protocolo síncrono. Gracias a esto, es posible enviar la información que se quiere mostrar desde el Arduino utilizando únicamente 4 pines, de los cuales 2 son para la alimentación y tierra del dispositivo. Los otros dos pines son el SDA, por donde se envían los datos, y el SCL, por el que transita la señal de reloj.

Figura 22. Módulo LCD IIC / I2C 1602

Para su utilización, lo más conveniente es utilizar la librería LiquidCrystal_I2C, la cual se puede añadir al código mediante la directiva **#include <LiquidCrystal_I2C.h>**. Tras ello se pueden utilizar funciones como LiquidCrystal_I2C **lcd(0x3F,16,2);**, que crean el objeto lcd con una dirección determinada (en este caso 0x3F). También hay que indicar como parámetros el número de columnas y filas de la pantalla.

Cuando se quiera iniciar el LCD se ha de llamar a la función **lcd.init();**, y para activar la luz de fondo **lcd.backlight();**.

Para escribir un texto en la pantalla se ubica el cursor en una determinada posición mediante la función **lcd.setCursor(0, 1);**, donde el primer parámetro, en este ejemplo 0, es la columna, y el segundo, en este caso 1, es la fila.

Con todo lo anterior preparado se puede mostrar una cadena de caracteres seleccionándola como parámetro en la llamada a la función **lcd.print();**.

El conexionado de este módulo en el robot se muestra en la siguiente imagen.

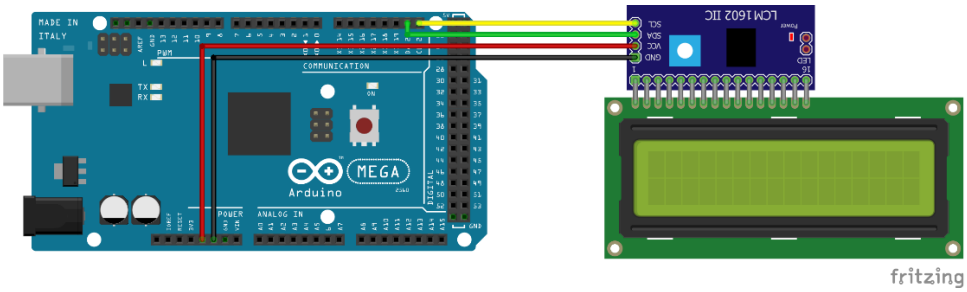


Figura 23. Conexionado del módulo LCD IIC / I2C 1602

2.5.3. Motores

En la realización del presente proyecto se han considerado dos posibilidades para la elección de los motores con los que se dotaría de movilidad al robot.

La primera opción contemplada fue la utilización de motores de corriente continua [11] ya que tanto su utilización como su adquisición es sencilla. En este sentido, el modelo más común comercializado es el que se muestra en la siguiente imagen.

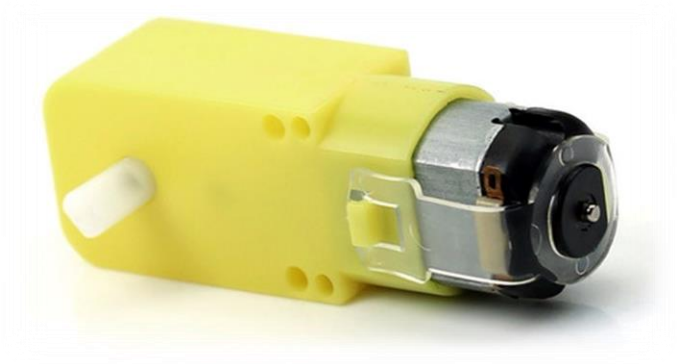


Figura 24. Motor de corriente continua

El funcionamiento de este tipo de motores se basa en el alineamiento de dos campos magnéticos, el del estator, que es la parte fija del motor, en el que se coloca un imán permanente que genera uno de los campos magnéticos, y el del rotor, que consiste en una espira por la que se hace pasar la corriente. De este modo, se induce un segundo campo magnético, que al estar desfasado respecto al primer campo, genera un par de giro. El rotor es móvil, y gracias a ello, el par generado provoca el giro de este, con el consiguiente movimiento del eje del motor.

Si los campos magnéticos se consiguiesen alinear, el motor se dejaría de mover. Para evitar que esto ocurra hay que invertir uno de los campos magnéticos, cambiando el sentido de la corriente en la espira del rotor para ello. Con el objetivo de que esto ocurra, se utiliza un colector de delgas que está dividido en dos mitades, de forma que cada una de ellas entra en contacto con una de las escobillas. Por cada 180º de giro, cada mitad del colector de delgas cambia de escobilla. De este modo se consigue una inversión mecánica del sentido de la corriente.

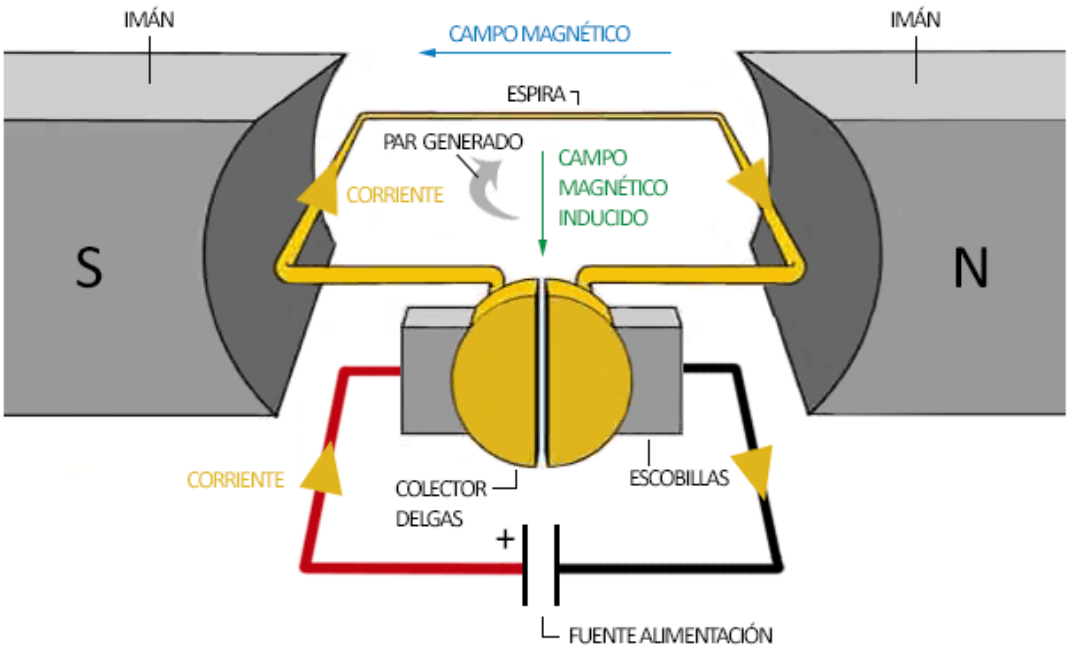


Figura 25. Esquema de funcionamiento de los motores de corriente continua

Para controlar la velocidad de este tipo de motores en las placas Arduino, se utiliza la técnica de la modulación por ancho de pulso (PWM) que permiten algunos de los pines de la placa. Según la cantidad de tiempo que esté la señal en alta dentro de un periodo de la señal variará el tiempo en el que alimenta la espira del rotor. Gracias a esto se puede aumentar o reducir la velocidad de giro del eje.

Algunas de las características más importantes del motor mostrado en la imagen al inicio del apartado, el cual incorpora una reductora, se muestran en la siguiente tabla.

Dimensiones del motor	70 mm x 22 mm x 18 mm
Peso	50 g
Reductora	48:1
Rango de alimentación	3 V - 12 V

Tabla 4. Características del motor de corriente continua

Para el control de este tipo de motores con una placa Arduino es necesario utilizar electrónica de adaptación. En este sentido, se pueden utilizar controladores de motores, con los que también se puede modificar el sentido de giro. Para realizar el control de los motores se utilizó el controlador L293D, que se puede conseguir en forma de placa adicional de conexión.

Figura 26. Placa de conexiones del L293D

Este dispositivo permite controlar la velocidad y el sentido de giro de los motores de continua, los cuales se pueden alimentar con una fuente de tensión externa a la placa Arduino. La placa tiene unas dimensiones muy similares al Arduino UNO ya que está pensado para colocarse encima conectando todos los pines, en concreto mide 6.8 cm x 5.5 cm x 2 cm. Permite utilizar motores que requieren una alimentación de entre 4.5 V y 10 V de continua. El resto de las características se presentan en la hoja de datos del componente. Este controlador permite trabajar con hasta 4 motores de forma simultánea si solo se requiere que estos giren en un único sentido, pero si se necesita poder seleccionar el sentido de giro de los motores, el número de motores con los que se puede trabajar de forma simultánea se ve reducido a únicamente 2. Esto último, junto a los planteamientos que se van a presentar en las siguientes líneas, llevaron a considerar la búsqueda de otro tipo de motores que instalar en el robot.

En este punto hay que recordar que, uno de los objetivos del proyecto es realizar un estudio de la posición del robot, de forma estimada, durante su navegación, es decir, la odometría.

En el caso de utilizar motores de continua, hay que introducir en el modelo algún sistema que permita conocer cuántos grados ha girado la rueda, de modo que a partir de esa información poder llegar al conocimiento de en qué posición se encuentra el robot. Los encoders son dispositivos que permiten generar una señal de pulsos a partir el movimiento de giro que se produce en un eje. El modelo más utilizado de encoder es el incremental [12], cuyo funcionamiento, de un modo muy sintetizado, se basa en que un emisor de luz, habitualmente LED, proyecte una señal luminosa que llegue a un disco rotatorio. Este disco incorpora unas ranuras por las que puede pasar la luz, y de este modo llegar a un fotorreceptor.

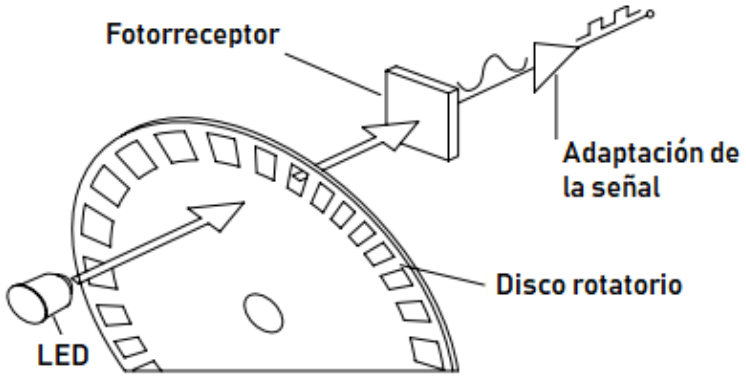


Figura 27. Esquema de funcionamiento de un encoder incremental

Como se muestra en la imagen, solo podrá pasar la luz por aquellos huecos del disco rotatorio en los que así lo permita el disco rotatorio. La señal se hace pasar por un circuito electrónico que adapta la señal generando pulsos cuadrados. Contando el número de pulsos que se generan se puede conocer los grados que ha girado el motor.

El método planteado resulta bastante complejo en comparación con el que se podría llevar a cabo con la utilización de motores paso a paso. Además, estos encoders tienen un precio bastante elevado, pudiendo conseguir un modelo básico a partir de los 50 euros. Es por estos motivos por los que se decidió avanzar en el proyecto con este otro tipo de actuador.

Los motores paso a paso [13] permiten el movimiento de un eje de forma discretizada, es decir, que permite conocer en todo momento la cantidad de pasos que se han dado y el sentido en el que se han realizado. El control de este tipo de actuadores es muy distinto al de los motores de continua ya que en vez de alimentarlos a través de dos cables (uno de ellos para la tierra), es necesario activar una serie de líneas en una secuencia concreta para que se produzca el movimiento. Por esto último, resulta conveniente utilizar estos motores junto a un controlador, y por esa razón se suelen encontrar en las tiendas kits que incluyen ambas partes.

En este sentido, el modelo que se ha utilizado ES EL 28BYJ-48, controlado por un integrado ULN2003APG, el cual permite generar las secuencias para el movimiento del motor empleando para ello únicamente 4 pines.

Figura 28. Motor paso a paso 28BYJ-48 y su controlador

Este dispositivo ya incorpora una reductora, de 1 / 64, la cual, al tratarse de un motor paso a paso, genera que exista una muy alta precisión a la hora de realizar movimientos.

Figura 29. Reductora del motor paso a paso

Las características principales del motor a paso a paso son las siguientes:

Modelo	28BYJ-48
Tensión nominal de alimentación	5V - 12 V
Numero de fases	4
Resistencia	50 Ω
Par motor	34 N/m
Consumo aproximado	55 mA
Pasos por vuelta	64 (en configuración de medios pasos)
Reductora	1 / 64
Ángulo de paso	5.625º (sin reductora)

Tabla 5. Características del motor paso a paso

El dato del ángulo de paso merece una explicación a parte, ya que, debido a que el motor da 64 pasos por vuelta, cuando trabaja con la configuración de medios pasos, y que además existe un reductor 1/64, la precisión pasa a ser de 0.0879º, la cual es bastante elevada. En la implementación de este robot no se ha utilizado esta configuración de medios pasos para simplificar el código ya que de todos modos la precisión sigue siendo muy alta (0.1758º de ángulo de paso).

Una configuración de pasos completos consiste en que se activen cada una de las bobinas que generan el par de rotación en el eje del motor una a una, de forma que la secuencia de activación de las bobinas contaría con 4 etapas. Por otro lado, en una configuración de medios pasos, la secuencia consta de 8 etapas, en las que además de activar cada bobina una a una, en medio de estas operaciones se mantiene activa la última bobina, al mismo tiempo se activa la siguiente, de modo el eje del motor rota hasta situarse en un punto intermedio entre ambas bobinas. Es posible comprender mejor esto último atendiendo a las siguientes imágenes [1].

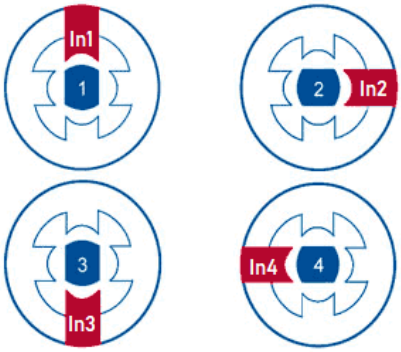


Figura 30. Configuración de pasos completos de los motores

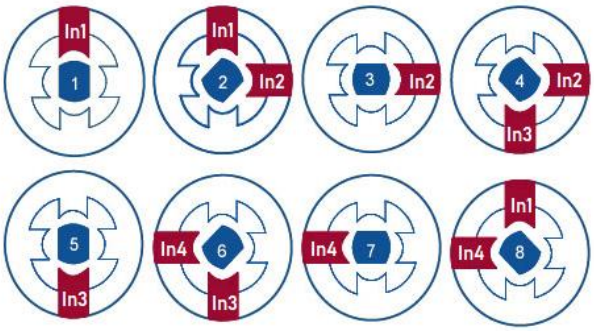


Figura 31. Configuración de medios pasos de los motores

Por otro lado, el controlador presenta las siguientes especificaciones:

Modelo	ULN2003
LEDS	4 (Uno para cada una de las líneas que se pueden activar)
Alimentación	5 V - 12 V
Pines de control	4
Dimensiones	42 mm x 30 mm

Tabla 6. Especificaciones del controlador del motor paso a paso

Se ha mencionado en los párrafos anteriores, que existe la posibilidad de utilizar este motor en una configuración de medios pasos o de pasos completos. Según la que se elija, se deberá realizar una secuencia de activación de los pines de control, de forma que el rotor avance en intervalos de mayor recorrido o menor.

En la configuración de medio paso, son necesarias 8 etapas de activación para dar un paso completo en el giro del motor.

Secuencia	IN1	IN2	IN3	IN4
1	High	Low	Low	Low
2	High	High	Low	Low
3	Low	High	Low	Low
4	Low	High	High	Low
5	Low	Low	High	Low
6	Low	Low	High	High
7	Low	Low	Low	High
8	High	Low	Low	High

Tabla 7. Secuencia de activaciones en configuración de medio paso del motor

En la anterior tabla, IN1, IN2, IN3 e IN4, hacen referencia a los 4 pines de control de la placa. En casa secuencia solo deberán estar activos aquellos pines en los que así se indique mediante la palabra *High*, mientras que *Low* se reserva para indicar un nivel bajo de tensión. Por su parte, en la siguiente tabla se representa, con la misma representación, la secuencia para el modo de pasos completos, en el que hay 4 etapas distintas.

Secuencia	IN1	IN2	IN3	IN4
1	High	Low	Low	Low
2	Low	High	Low	Low
3	Low	Low	High	Low
4	Low	Low	Low	High

Tabla 8. Secuencia de activaciones en configuración de pasos completos del motor

La secuencia anterior es la que se realiza activando en cada etapa una única bobina, pero si se utilizan dos se consigue un par mayor sin modificar la cantidad de etapas de la secuencia. Esta segunda configuración es la recomendada por el fabricante.

Secuencia	IN1	IN2	IN3	IN4
1	High	High	Low	Low
2	Low	High	High	Low
3	Low	Low	High	High
4	High	Low	Low	High

Tabla 9. Secuencia de activaciones recomendada por el fabricante

Sobre la base de las consideraciones anteriores y a modo de resumen, se ha optado en el desarrollo del robot, por utilizar motores paso a paso, dadas las ventajas que estos

ofrecen para realizar un control de la navegación del mismo. Para ello han sido necesario utilizar 12 pines del Arduino MEGA, 4 de ellos para cada uno de los motores paso a paso.

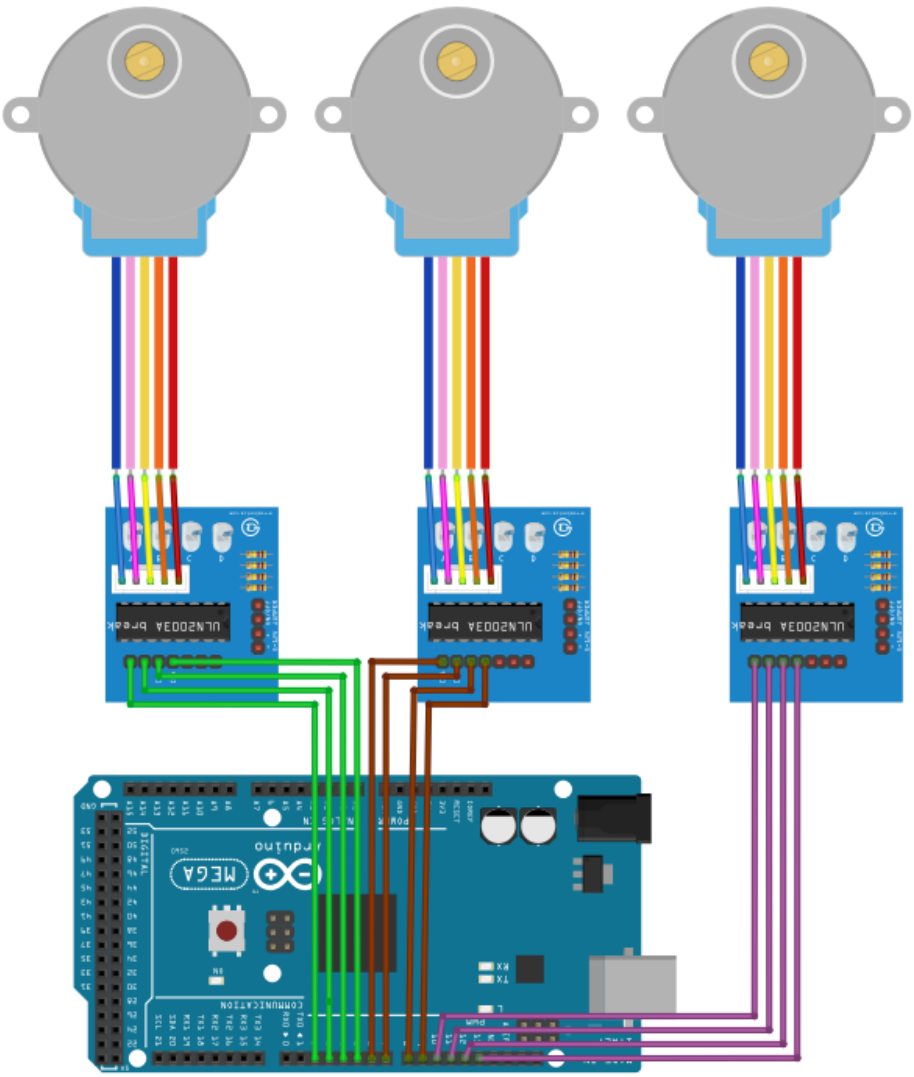


Figura 32. Esquema de conexionado de los motores paso a paso

2.5.4. Chip ESP-8266

De acuerdo con los razonamientos que se han ido planteando, existía la necesidad de incorporar a la electrónica del robot algún dispositivo que permitiese que la tarjeta Arduino que se instalase se pudiese conectar a un punto de acceso Wifi.

En ese mismo sentido, se trató de buscar una tarjeta Arduino que incorporase un chip destinado a tal propósito. De esta forma se comenzó a trabajar con el Arduino WeMos, que está basado en el ESP-12.

Antes de continuar, resulta conveniente realizar una serie de aclaraciones de modo que la información que se presente en esta memoria, y que tenga relación con lo que aquí se expone, se pueda entender con claridad. El ESP-8266 es una familia de chips Wifi de bajo coste que permite utilizar el protocolo TCP/IP y el estándar IEEE 802.11 b/g/n. En

la página web de la marca es posible consultar todos los modelos que existen y las distintas novedades que se van presentando.

Modelos de ESP-8266
ESP-WROOM-02
ESP-01
ESP-02
ESP-03
ESP-04
ESP-05
ESP-06
ESP-07
ESP-08
ESP-09
ESP-10
ESP-11
ESP-12
ESP-12F
ESP-12E/Q
ESP-12S
ESP-13
ESP-14
WT8266-S1

Tabla 10. Listado de variantes del ESP-8266

En la anterior tabla se ofrece una lista de los modelos ESP-8266 existentes en el momento de la redacción de esta memoria, la cual se ha elaborado a partir de la información existente en la web de la marca [14].

En el caso particular de este proyecto, se ha mencionado que el Arduino WeMos incorporaba la versión ESP-12, el cual cuenta con 16 pines, motivo por el cual se consideró viable que pudiese servir en el robot diseñado, aunque finalmente no fue así ya que muchos de estos pines no estaban disponibles para poder utilizarlos para el control de los motores.

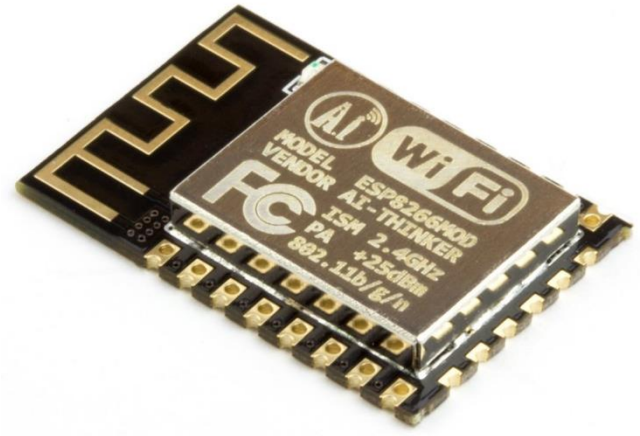


Figura 33. Módulo ESP-12

Este chip permitía utilizar la librería ESP8266Wifi a la hora de programarlo, lo cual facilitó el trabajo cuando se empezó a trabajar con el Arduino WeMos, ya que gracias a ellos era sencillo elaborar un código que se conectase a una red de área local. De este modo se comenzó a diseñar un método de envío de órdenes de movimiento basado en un servidor Web, el cual se expondrá en el siguiente capítulo de la memoria. El sistema implementado finalmente en el robot dista en gran medida de todo esto, pero se puede afirmar que se pudo contemplar todas las virtudes de este dispositivo, las cuales suponen que, aunque no se haya podido utilizar en este proyecto, es un chip muy útil para multitud de aplicaciones.

Figura 34. ESP-12 en el Arduino WeMos D1

Tras descartar el Arduino WeMos, porque éste no ofrecía un número de pines suficiente para este proyecto, hubo que buscar una nueva solución ante la necesidad de implementar una comunicación Wifi entre el robot y el ordenador. En las tiendas locales se podía conseguir el modelo ESP-01, con lo que se decidió probar con él junto a un Arduino MEGA. De este modo, se contaba con la posibilidad de utilizar la comunicación Wifi al mismo tiempo que se disponía de pines suficientes para controlar los motores y la electrónica que se pudiese incorporar más adelante.

El lado negativo de este cambio fue que, como el ESP ya no era el microcontrolador principal, sino que se debía programar el Arduino MEGA y que este se comunicase con el ESP-01 para enviar los mensajes a través de la red Wifi. En este mismo sentido, quedó atrás la posibilidad de poder utilizar la librería ESP8266Wifi, puesto que para ello se requería que la programación se hiciese directamente en el ESP-01.

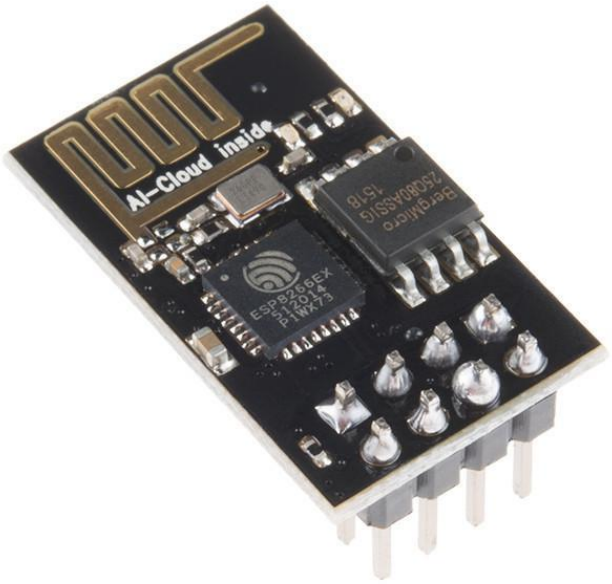


Figura 35. Módulo ESP-01

Este módulo es el que incorpora el modelo final del robot. El principal inconveniente de este dispositivo, en comparación al anterior y debido a la manera en la que se utiliza, es la circunstancia ya descrita de no poder trabajar con librería específicas.

En cuando a las características técnicas de ambos modelos, están vienen detalladas en la misma web de la que se habló antes en este apartado. En la siguiente tabla se muestra un resumen de estas especificaciones.

Modelo	Numero de pines	Contiene indicador LED	Colocación de la antena	Dimensiones (mm)	Tamaño memoria flash
ESP-01	8	Sí	Grabada en la PCB	14.3 x 24.8	512 KB
ESP-12	16	Sí	Grabada en la PCB	24.0 x 16.0	4 MB

Tabla 11. Especificaciones del ESP-01 y del ESP-12

Una aclaración que resulta pertinente es que dentro del propio ESP-12 existen distintas variedades, como son el ESP-12F, el ESP-12-E/Q o el ESP-12S. En la placa WeMos D1 utilizada en el proyecto no se indica claramente cuál es la que incorpora, y si se busca la información en la red se encuentran distintos modelos, aunque el más repetido es el ESP-12EX. En la práctica, esto no ha supuesto un problema puesto que todos estos modelos basados en el ESP-12 presentan las características que se han presentado en la última tabla, y el resto de las especificaciones satisfacen los requerimientos del robot implementado.

Para utilizar las características del ESP-01 hubo que utilizar los denominados comandos Hayes, también muy comúnmente llamados comandos AT. Es un lenguaje de comandos para la configuración de dispositivos que permiten la comunicación entre dispositivos.

Existe un gran número de comandos AT [15], pero de ellos, solo algunos fueron necesarios en este proyecto.

Comando AT	Utilidad
AT+CWMODE=	Para configurar el punto de acceso (modo de operación).
AT+CWJAP="ssid","contraseña"	Se indica la red a la que se quiere conectar y su contraseña
AT+CIPMUX=	Para configurar múltiples conexiones (1)
AT+CIFSR	Se obtienen las direcciones IP
AT+CIPSTART=ID,"UDP","direccionIP","puerto"	Se indica el identificador de la conexión (un entero), el tipo de conexión, la dirección IP y el puerto al que se va a conectar el dispositivo.
AT+CIPDINFO=	Se habilita (1) o deshabilita (0) la información remota
AT+CIPRECVMODE=1	Se establece el módulo en el modo de recepción de datos
AT+CIPSEND=	Se establece el módulo en el modo de envío de datos

Tabla 12. Comandos AT utilizados en el proyecto

En relación al conexionado del módulo ESP-01, este se ha instalado en el robot del modo en el que se muestra en la siguiente imagen.

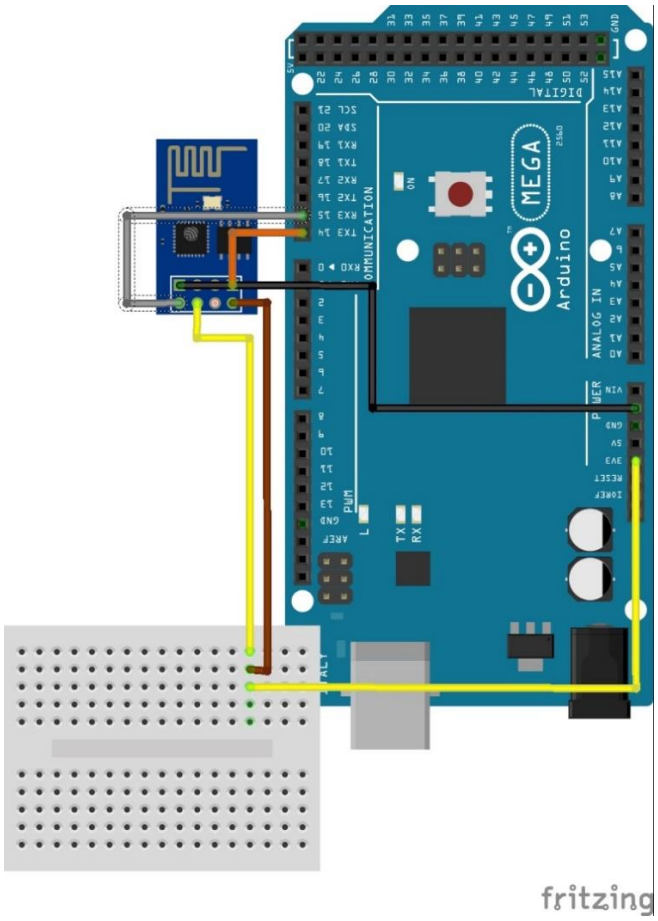


Figura 36. Esquema de conexionado del ESP-01

2.6. **Comunicación serial**

La comunicación serial, también llamada serie o secuencial, permite el envío de información en bits que se transmiten de uno en uno, de forma secuencial. Cada bit dura un determinado tiempo, el cual es conocido.

Resulta conveniente mencionar que existen dos tipos distintos de comunicación serial, la síncrona y la asíncrona, siendo esta última la que se emplea al trabajar con las placas Arduino.

En la asíncrona, la sincronización se realiza a partir de la propia señal. Tan solo es necesario establecer una determinada velocidad de transmisión, y el envío de información puede iniciarse en cualquier momento.

Todas las tarjetas Arduino cuentan con puertos para este tipo de comunicación [16], identificados mediante los pines RX y TX. En el caso particular del Arduino MEGA, a parte de los pines comunes al resto de placas, este cuenta con tres puertos adicionales, los cuales no están conectados al puerto USB de la tarjeta.

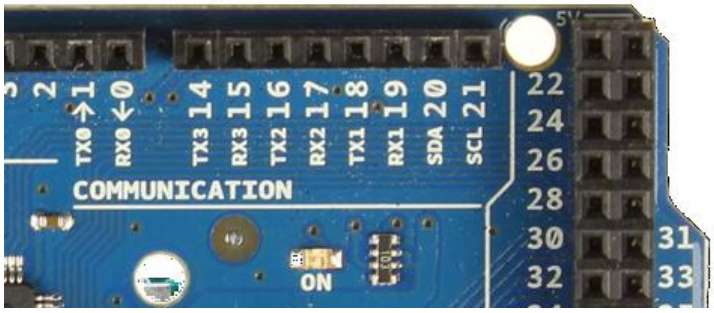


Figura 37. Pines RX y TX del Arduino MEGA

Cabe mencionar que fue necesario utilizar uno de estos puertos adicionales en el robot para la comunicación entre el Arduino MEGA y el módulo ESP-01, concretamente los pines RX3 y TX3.

Todas las placas Arduino que se han utilizado a lo largo del desarrollo del robot descrito en este documento, disponían de al menos una unidad UART. Todas ellas operan a un nivel TTL de 0 Voltios a 5 Voltios. También estas tarjetas contaban con conectores USB, o Micro USB, que es el caso de Arduino WeMos.

Resulta preciso explicar a qué hace referencia el término UART [18]. Las siglas significan Transmisor Receptor Asíncrono Universal. Este sistema permite utilizar una transmisión serie asíncrona como la que se ha descrito en los párrafos anteriores. Para ello, se incorporan los bits del mensaje entre los bits de inicio y parada. Además, existe la posibilidad de realizar un control de errores mediante paridad.

En el caso concreto de Arduino existe un bit de inicio que siempre estará en baja, luego se incorporan los bits de datos. Este tramo puede variar desde 5 bits hasta 9 bits, aunque se suele trabajar con 8 bits. Después se encuentra el bit de paridad, que puede ser

configurado para trabajar con paridad par o impar, pero que por defecto está desactivado. Por último, se añade el bit de parada, el cual siempre estará en alta.

Este tipo de comunicación es el que se ha empleado, por ejemplo, para visualizar información en el monitor Serie del Arduino IDE [17] cuando se realizaban pruebas con el robot conectado al ordenador a través de un cable USB.

Para ello se utiliza la instrucción Serial.begin(BAUDRATE);

El parámetro "BAUDRATE" indica el número de bits que se pueden enviar por segundo, y la unidad es el baudio. En el proyecto se han empleado principalmente dos valores, 9600 baudios y 115200 baudios. Conociendo este dato resulta sencillo calcular el tiempo de bit, es decir, el tiempo que se mantiene cada bit en la comunicación, ya que este es la inversa del número de baudios.

Para el segundo caso sería:

Ecuación 1

$$TiempoBit = \frac{1}{BAUDRATE} = \frac{1}{115200} \sim 8.68\ \mu s$$

Por otro lado, la instrucción para enviar la información, que puede ser tanto una cadena de caracteres como un valor numérico, es Serial.print();

2.7. **Comunicación Wifi**

Una de las características principales del robot es que la comunicación entre el ordenador y la placa Arduino se produce a través de una comunicación Wifi, tanto para el envío de la velocidad de los motores como para la recepción de la pose del robot en el ordenador.

Wifi significa *Wireless Fidelity*, o lo que es lo mismo, fidelidad inalámbrica. Se trata de una tecnología que permite a los dispositivos comunicarse entre sí, de forma inalámbrica, tanto de forma directa como a través de internet, necesitando en este segundo caso un punto de acceso inalámbrico.

Todos los productos que incorporan esta marca, como es el caso del chip ESP8266, deben satisfacer las especificaciones del estándar IEEE 802.11 , el cual conviene que sea descrito.

El estándar IEEE 802.1 [19] surgió con el objetivo de proporcionar conectividad a dispositivos electrónicos a las redes de acceso local inalámbricas (WLAN). Define el control de acceso al medio (MAC), es decir la capa de enlace, y la capa física, describiendo por tanto las capas 1 y 2 del modelo OSI. A lo largo de los años han ido surgiendo nuevas versiones de este estándar, los cuales han ido aumentando la velocidad de transmisión de los datos, han ido mejorando los métodos de tratamiento de los datos y han impulsado la expansión de esta tecnología a lo largo de todo el mundo. Sin entrar en más detalle en relación a esto último, ya que las necesidades del hardware implementado

en este sentido son bastante bajas, si es importante definir una serie de conceptos [20] para las posteriores explicaciones.

Red	Estándar	Banda de frecuencia	Rango nominal	Máxima velocidad de transmisión
WLAN	IEEE 802.11	2.4 / 5 GHz	100 m	1 Mbps
	IEEE 802.11a	5 GHz	100 m	48 Mbps
	IEEE 802.11b	2.4 GHz	100 m	11 Mbps
	IEEE 802.11g	2.4 GHz	100 m	54 Mbps
	IEEE 802.11n	2.4 / 5 GHz	250 m	600 Mbps
	IEEE 802.11ac	5 GHz	250 m	1.3 Gbps

Tabla 13. Características de los distintos estándares para redes WLAN

Una estación es cualquier dispositivo con capacidad para interferir en el medio inalámbrico, como, por ejemplo, un ordenador.

Un punto de acceso es el equipo que permite a los dispositivos inalámbricos realizar una conexión a una red cableada mediante, en este caso Wifi, pero también se puede utilizar para otros estándares. Debido a que el desarrollo del robot se ha llevado a cabo en distintos lugares, era necesario acceder a distintos puntos de acceso, cada uno de ellos con una dirección IP distinta. Esto era un problema ya que implicaba el tener que cambiar parte del software para poder utilizar el robot. La solución fue utilizar un punto de acceso portátil, de modo que la dirección IP no cambiase. Esto es una posibilidad que ofrecen, por ejemplo, los teléfonos móviles actuales.

Al trabajar con redes inalámbricas existen dos posibles arquitecturas que se pueden implementar. En el modo Ah hoc los distintos dispositivos se conectan directamente entre sí. Por otro lado, la otra posibilidad que existe, que es la que se ha utilizado en este TFG, es el modo infraestructura, en el que todos los dispositivos implicados en la comunicación se conectan en primer lugar a un punto de acceso común a todos ellos. Como se ha mencionado anteriormente, se ha utilizado un teléfono móvil como punto de acceso, el cual puede a su vez conectarse a una red de área local o a la red de datos móviles. De este modo se solucionaba el problema de tener que encontrar un *router* en cada lugar en el que se quería utilizar el robot.

2.7.1. Protocolo TCP/IP

Los protocolos definen una serie de normas que deben cumplir todos los elementos, tanto software como hardware, que intervienen en una comunicación, en la que se intercambia una determinada información. En caso contrario no sería posible interpretar el mensaje.

En este apartado se pretende dar a conocer el protocolo TCP/IP [21], el cual se utilizó en los primeros modelos de comunicación entre el robot y el ordenador. El protocolo TCP/IP está compuesto por las capas de aplicación, transporte, red, interfaz de red y hardware. Se define también cómo es el paso de información a través de todas estas capas. En concreto, los programas de aplicación envían la información a la capa de transporte, donde en este caso se encuentra el protocolo TCP, el cual divide los datos en

distintos paquetes, incorporando la dirección de destino antes de pasar a la siguiente capa, la de red. En esta capa se genera, para cada uno de los paquetes mencionados, un datagrama IP que incluye la cabecera y la cola de datagrama. Este datagrama se transfiere a la capa de interfaz de red, desde la cual se transmiten los datagramas IP, como un conjunto de hilos en paralelo, a través del medio físico que se vaya a utilizar, un cable ethernet por ejemplo.

En la siguiente imagen se presenta el flujo de la información desde el remitente hasta el host cuando se utiliza el protocolo descrito.

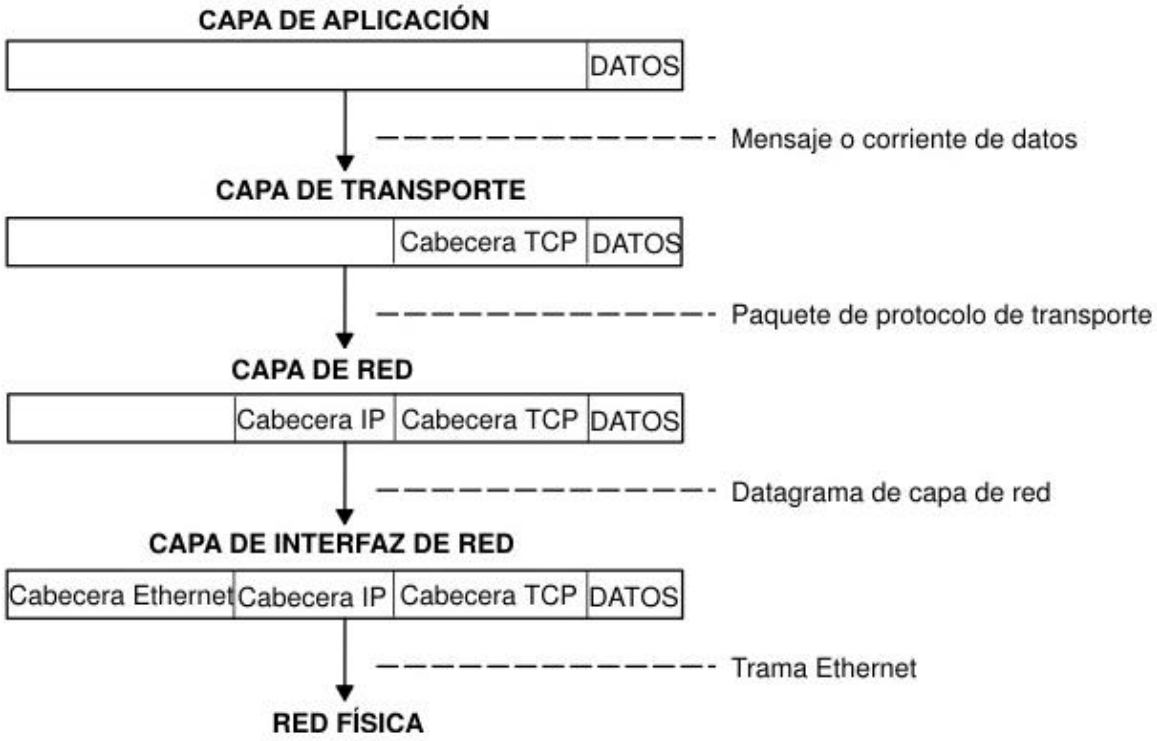


Figura 38. Protocolo TCP/IP desde el desde el remitente hasta el host

El avance entre capas se invierte cuando el sentido deseado de la comunicación es el contrario. De igual modo, los distintos pasos que se suceden son los opuestos a los planteados al inicio de este apartado del documento.

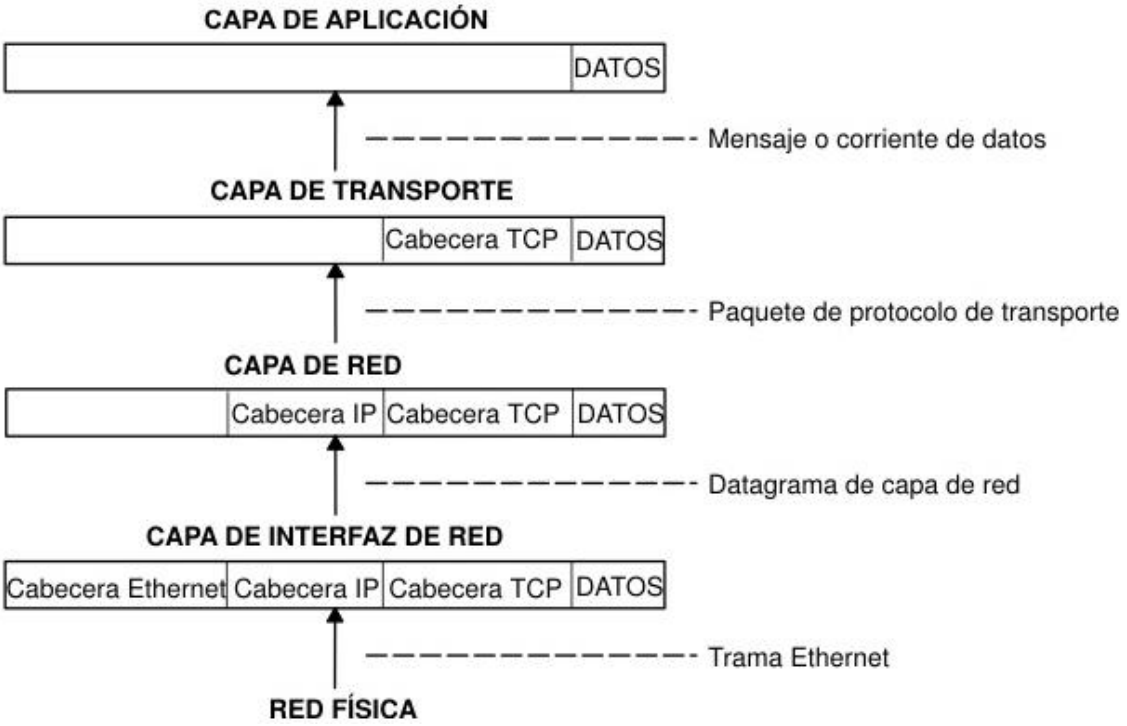


Figura 39. Protocolo TCP/IP desde el desde el host hasta el remitente

En el protocolo TCP/IP, con el objetivo de que la comunicación sea fiable, el destinatario debe confirmar la recepción del mensaje, por lo que el avance por las distintas capas es constante en ambos sentidos. A causa de esto, se puede plantear un esquema general como el que se presenta en la siguiente ilustración.

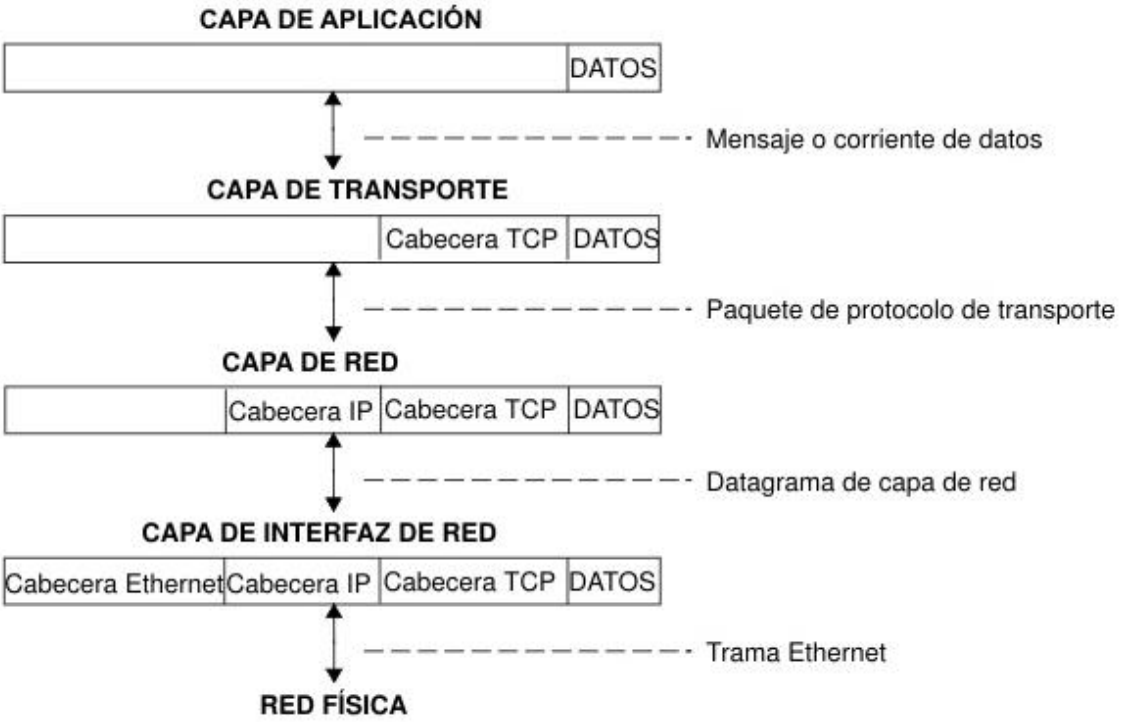


Figura 40. Esquema general del protocolo TCP/IP

Se ha mencionado que el protocolo es fiable, pero también presenta muchas otras características muy positivas. Esta afirmación se basa en que presenta una conexión ordenada ya que los bytes están numerados, lo cual posibilita una reconstrucción del mensaje aunque se produzcan pérdidas. Se incorpora un sistema de control de redundancia cíclica (CRC) para identificar errores. Otras características son la persistencia, para garantizar que la conexión está activa, y el control que se realiza del flujo, basado en que se regula la tasa de envío de la información evitando saturar el receptor, y en que se adapta la velocidad de procesado de la información para evitar la congestión de la red.

Adicionalmente se puede mencionar que existen numerosos servicios que se pueden utilizar con este protocolo. Uno de ellos es el denominado Telnet, el cual da acceso en modo terminal a otro equipo. Este servicio se utilizó en una fase intermedia del desarrollo del proyecto en la que era necesario comprobar que el módulo ESP-01 se había configurado correctamente para recibir datos que se enviasen desde un PC conectado a la misma red, y en el puerto adecuado.

De todos los planteamientos anteriores se puede deducir que utilizar este protocolo presenta un gran número de beneficios al ser utilizado, sin embargo, dado que el módulo ESP-01, presenta algunas limitaciones en cuanto a la velocidad a la que puede enviar y recibir los datos, el que se tuviese que confirmar la recepción del mensaje, provocaba que no se pudiese actualizar la velocidad de los motores a la frecuencia deseada. Para solucionar este problema se realizaron una serie de cambios en el modelo, entre los cuales se decidió cambiar de protocolo.

2.7.2. Protocolo UDP

El protocolo UDP [22] define, del mismo modo que el TCP/IP, una serie de normas y procedimientos que se deben cumplir para que un determinado mensaje pueda llegar al destinatario deseado y que se pueda interpretar correctamente. No obstante, también presenta algunas diferencias notables, las cuales resultaron determinantes a la hora de escoger este protocolo como el que se implementaría en el robot.

El UDP emplea los puertos de protocolo de destino, los cuales se identifican por enteros positivos, ya que, en este tipo de comunicación, los equipos que emiten el mensaje no pueden conocer los procesos que están activos. En el caso del código cargado en el Arduino MEGA del robot, hubo que habilitar la posibilidad de usar múltiples conexiones, de este modo, con el identificador 0 se reciben los mensajes con la velocidad de los motores, y con el identificador 1 se envía la pose del robot a un socket UDP.

Este protocolo no asegura la recepción del mensaje ya que el receptor del mensaje no envía ningún mensaje de confirmación. Esto que a priori podría parecer una desventaja, resulta muy favorable para los intereses particulares del proyecto, ya que, permite que la frecuencia de envío de la información sea sustancialmente más rápida. Esto último supuso un avance muy importante en el funcionamiento del robot debido a que supuso un gran aumento en la fluidez de respuesta del sistema.

La cabecera del UDP está formada por 32 bits. De ellos, los 16 primeros contienen el número del puerto de origen y la longitud del paquete, y los siguientes 16 informan del número del puerto de destino y una suma de comprobación. Este último es un parámetro que permite que, aunque el propio protocolo no lo asegure por defecto, las aplicaciones de envío y recepción puedan verificar que el mensaje se ha entregado correctamente. Para ello, se calcula la suma de la comprobación de los datos y también la cabecera, siendo esta suma a la que se hacía referencia en la explicación del contenido de los 32 bits de la cabecera.

Otras características de este este protocolo que resultan positivas es que también cuenta con un CRC para comprobar errores a nivel superior, y que es un sistema que genera poca sobrecarga. Es de gran utilidad en sistemas en los que no resulta necesario mantener la conexión, como es el caso de este trabajo.

En el modelo final del robot se ha establecido una recepción de valores de velocidades de los motores en el Arduino y un sistema de envío de los tres parámetros de la pose. En ambos casos se utiliza el protocolo UDP por las razones antes expuestas.

Resulta conveniente presentar, antes de concluir con este apartado, que se ha implementado un socket UDP para la recepción de la pose, escrito en Python. Este permite la recepción de un datagrama, en concreto una cadena de caracteres que incluye los tres parámetros de la pose del robot.

Las funciones que utiliza dicho socket son las siguientes:

Funciones	Significado
socket()	Se crea el punto de comunicación
bind()	Enlaza el socket a una dirección y puerto
recvfrom()	Se detiene la ejecución para recibir un mensaje

Tabla 14. Funciones utilizadas para el socket UDP que recibe la pose en el ordenador

Estas instrucciones coinciden con las que deben incluir tanto los servidores como los clientes UDP. Las instrucciones que no aparecen y podrían estar son sendto(), la cual no se incluye ya que el socket no tiene que enviar mensajes en ninguna circunstancia, y close(), ya que esta se encarga de cerrar la conexión, situación que no se va a dar en ningún momento en la propia ejecución del programa.

La implementación práctica específica se detallará en el apartado destinado a tal propósito.

2.8. **ROS**

2.8.1. Introducción a ROS

ROS [23], cuyas siglas significan Sistema Operativo Robótico, es un entorno de desarrollo de software, que actualmente es un estándar para la programación y control de

robots. Asociados a esta plataforma, existen multitud de paquetes software, algunos de ellos se utilizarán en este proyecto.

En este apartado de la memoria se pretende dar a conocer esta herramienta, basando las explicaciones en lo que se expone en la guía oficial de ROS [1], la cual es recomendable consultar antes de empezar a trabajar con la plataforma.

En primer lugar, se instaló el sistema, concretamente la distribución Melodic Morenia, que en aquel momento era la adecuada a la versión de Ubuntu en la que se iba a trabajar, la 18.04.2.

Es importante mencionar que, para utilizar algunos de los comandos asociados a este sistema operativo robótico, es necesario introducir en cada nueva ventana de comandos la siguiente instrucción:

```
$ source /opt/ros/melodic/setup.bash
```

Tras instalar el sistema se puede comenzar a trabajar con él. Resulta conveniente exponer una serie de conceptos antes de continuar con el resto de las explicaciones.

Los paquetes o *packages* son la unidad de organización del software que utiliza ROS. Junto a esto, se incluye la descripción de cada paquete en un archivo de nombre package.xml, el cual se denomina *manifest*, y donde se almacena toda la información del paquete.

Los nodos son ejecutables que se utilizan en ROS para establecer una comunicación entre distintos programas. Para ello se utilizan librerías como rospy (para clientes Python) o roscpp (para clientes C++).

Por otro lado, un tópico es el medio a través del cual los nodos envían los mensajes. Para ello un nodo publicador o *Publisher* debe enviar la información a través de dicho tópico, de modo que otro nodo, en este caso denominado *subscriber*, pueda recibir la información si este está subscrito al mismo tópico. De este modo se define el método básico de comunicación que se emplea en ROS. Sin embargo, también es posible utilizar un sistema de cliente y servidor, garantizándose en este caso la recepción del mensaje. De este modo, este segundo método es más apropiado para aquellos casos en los que es importante que el mensaje se reciba correctamente. Por el contrario, en modelos como el implementado en este Trabajo de Fin de Grado, en los que se publica de forma constante un mensaje, y el que aluno de ellos se pierda no supone un grave problema, se utiliza el método de *Publisher* y *subscriber*, ya que este consume menos recursos.

Tras haber presentado los anteriores conceptos, se puede comenzar a exponer la forma de operar con el sistema. En este sentido, para poder utilizar ROS se debe crear un entorno de trabajo catkin, que es el tipo con el que se trabaja en las versiones más actuales de este sistema.

En el caso de este proyecto, se creó un entorno de trabajo de nombre “ROS_WS”, para lo cual fue necesario abrir una nueva ventana de comandos e introducir las siguientes líneas:

```
$ mkdir -p ~/ROS_WS/src
$ cd ~/ROS_WS/
$ catkin_make
$ source devel/setup.bash
```

Por otro lado, para crear un paquete hay que definir como mínimo dos archivos, que consisten en el fichero de compilación package.xml que se mencionó antes, y el CMakeLists.txt que utiliza la herramienta catkin_make para compilar los códigos del paquete. No obstante, en este proyecto no fue necesario crear paquetes ROS, ya que, tanto para el sistema de intercambio de información de forma inalámbrica entre este sistema y el microcontrolador Arduino, como para la incorporación al modelo de un mando para realizar el control, se partió de software ofrecido en repositorios públicos.

En este punto resulta apropiado citar una instrucción que permite desplazarse en la ventana de comandos entre los distintos directorios y subdirectorios del entorno de trabajo. Se trata de roscd:

```
$ roscd [locationname[/subdir]]
```

Con todo lo mencionado en ellos párrafos anteriores preparado, ya se puede comenzar a trabajar con la plataforma. Para ello se ha de contar con aquellos nodos que resulten necesarios para la tarea que se quiere realizar. Estos nodos pueden estar escritos, tanto en Python como en C++, obteniéndose habitualmente códigos de menor extensión si se emplea el primer lenguaje. No obstante, en algunas asignaturas de la carrera se ha trabajado con C++ por lo que en este trabajo se han utilizado ambos lenguajes.

Como se mencionó anteriormente, una vez están listos los archivos, hay que compilarlos utilizando la instrucción catkin_make, de la siguiente forma:

```
$ cd ~/ROS_WS
$ catkin_make
```

De este modo ya sería posible iniciar los nodos que se han creado. En primer lugar, para poder trabajar con ROS hay que introducir la siguiente palabra en una ventana de comandos:

```
$ roscore
```

De este modo se inicia una serie de nodos y programas que son un requisito indispensable para poder utilizar un sistema basado en ROS. Esta ventana se puede minimizar, y se debe abrir un nuevo terminal para poder continuar.

Si se introduce la instrucción siguiente:

```
$ rosnode list
```

Se mostrará una lista de los nodos activos que existen actualmente en el ordenador que se está utilizando. Por otro lado, si lo que se desea es iniciar un nodo se debe utilizar la siguiente estructura:

```
$ rosrun [package_name] [node_name]
```

Por ejemplo, para iniciar el nodo “joy_node” del paquete “joy”, con el que en este proyecto se accede a la información de un mando que se conecte al ordenador mediante un cable USB, se escribiría la siguiente instrucción:

```
$ rosrun joy joy_node
```

Si adicionalmente se inicia el nodo subscrito al tópico “joy”, en el que el nodo anterior publica la información del estado de los botones y sensores del mando, del siguiente modo:

```
$ rosrun ros_arduino_python joy_to_speed
```

Y adicionalmente, en un nuevo terminal se introduce esta instrucción:

```
$ rosrun rqt_graph rqt_graph
```

Se mostrará lo siguiente por pantalla.

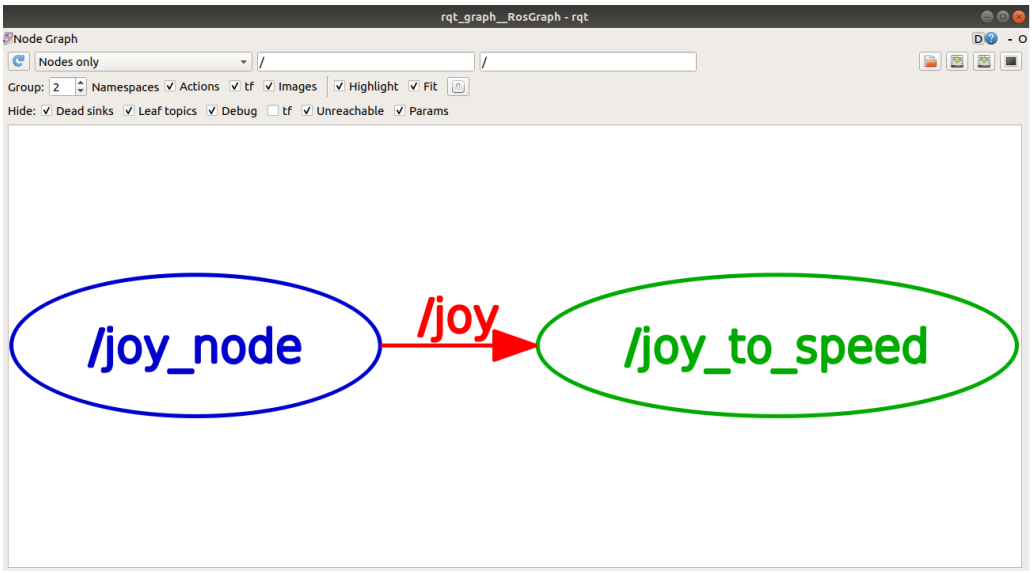


Figura 41. Primer tópico y nodos del sistema implementado en ROS

Se obtiene una representación gráfica de los nodos, en azul el *publisher* y en verde el *subscriber*, y del tópico a través del cual se envían los mensajes, que se muestra de color rojo.

Existe la posibilidad de visualizar por pantalla el contenido del tópico que se está enviando. Para ello, se envía en un nuevo terminar lo siguiente:

```
$ rostopic echo /joy
```

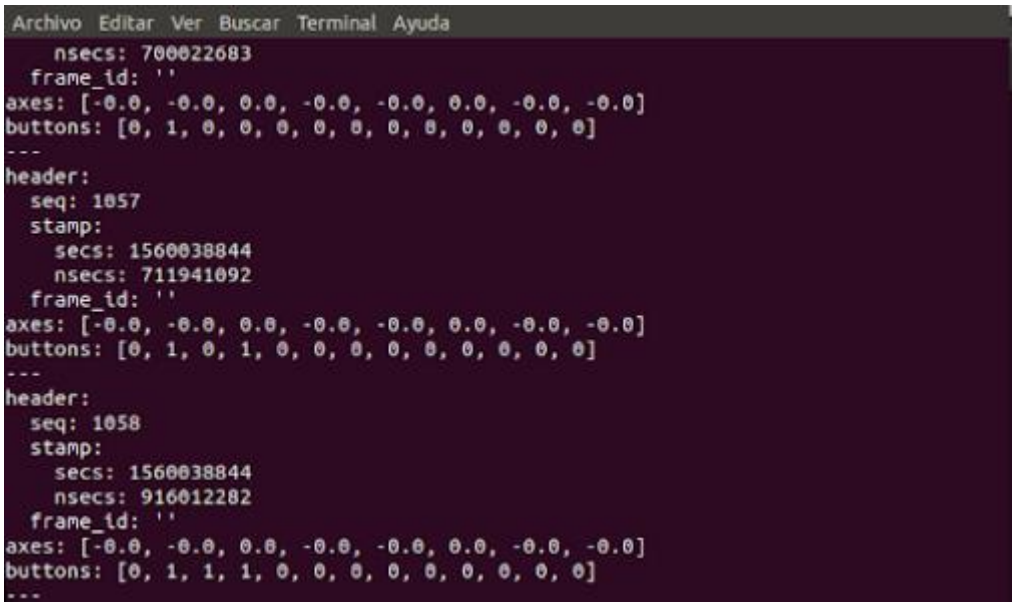


Figura 42.Parámetros del mando recogidos por ROS

En este caso se puede apreciar que se envían todos los parámetros referentes a los sensores del mando. En el momento en el que se tomó la imagen se estaban presionando 3 botones, lo cual se puede saber por los 3 unos que se muestran.

Si se recarga la representación en la que se pueden ver los nodos y tópicos activos, el comando rostopic echo utilizado está ahora subscrito al tópico joy para poder mostrar la información por pantalla.

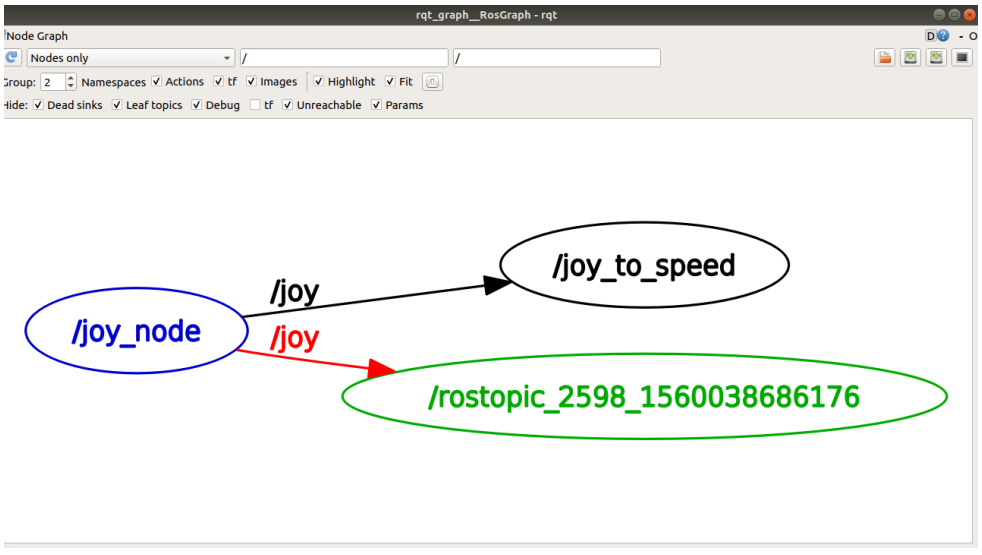


Figura 43. Subscripción mediante el comando rostopic echo

Para activar el siguiente nodo, que es el que se encarga de enviar las velocidades de los motores al microcontrolador a través de una conexión Wifi, se utiliza el siguiente texto:

```
$ rosrun ros_arduino_python joy_to_speed Arduino_node.py
```

Sin embargo, existe un método para poder iniciar varios nodos mediante una única instrucción, roslaunch. Para ello, es necesario definir previamente un archivo con extensión .launch en el que se indiquen los nodos que se quieren iniciar, y los paquetes en los que se encuentran. En el apartado 3.9 del siguiente capítulo, se expone el modo en el que se ha configurado este fichero para este proyecto.

Para utilizarlo, hay que introducir lo siguiente por la ventana de comandos:

```
$ roslaunch ros_arduino_python arduino.launch
```

En este caso solo es necesario un terminal. De este modo se inician 3 de los nodos que se han implementado en el proyecto. Si se vuelve a consultar la representación gráfica la que se ha hecho alusión en los párrafos anteriores, ahora se puede ver el sistema diseñado en ROS, el cual se comunica con el Arduino, enviando y recibiendo mensajes.

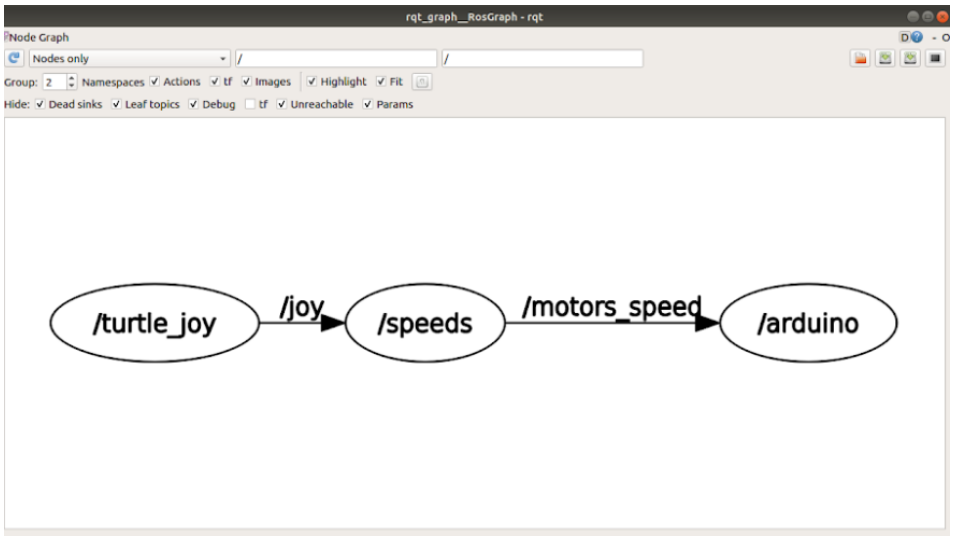


Figura 44. Nodos necesarios para el envío de las velocidades de los motores

2.8.2. Introducción a Rosserial

Rosserial [24] es un protocolo de comunicación serial que permite realizar un envío de datos, provenientes de tópicos o servicios de ROS, entre sistemas. De forma más específica, la librería rosserial_arduino, permite el intercambio de mensajes entre un sistema ROS y un microcontrolador Arduino. El inconveniente de este protocolo, a la hora de aplicarlo al robot diseñado, es que añade una sobrecarga de uso de memoria, de consumo de CPU y de comunicaciones, que impide que se utilice en una placa Arduino, salvo para tareas muy sencillas.

Por otro lado, el que la comunicación sea serial por defecto, se opone al objetivo de que el robot sea completamente inalámbrico, por lo que en un primer momento parecía que este método no era el más adecuado. No obstante, y tras analizar las posibilidades que ofrecía este protocolo, se encontró que ROS incorpora un método estandarizado para añadir nuevo hardware, el cual se presenta en la siguiente web oficial de ROS [25].

A partir de todos estos planteamientos, se contempló la posibilidad de utilizar un software que se adaptase al ESP-01, con el que se estaba trabajando en aquel momento. Esta idea se desarrollará en el apartado que corresponde a ello en el siguiente capítulo de la memoria, pero dado que, como se ha expuesto, se continuó el desarrollo en el sentido de las ideas planteadas, resulta conveniente profundizar más en el protocolo que describe el Rosserial.

Rosserial presenta una serie de limitaciones. En primer lugar, en la mayoría de los microcontroladores, el número de *Publishers* y *Subscribers* está limitado a 25, y el tamaño de los buffers de salida y entrada es de 512 bytes. Estos datos se ven reducidos en el caso del microcontrolador ATmega328P, que es el que incorpora por ejemplo el Arduino UNO, en el que el número de *Publishers* y *Subscribers* que se permiten es de 6, y los buffers de entrada y salida son de 280 bytes. En cualquiera de los casos, se satisfacían los requerimientos del proyecto.

Por otro lado, no se puede trabajar con datos de tipo float de 64 bits, viéndose estos reducidos automáticamente a float de 32 bits, con la pérdida de precisión que ello conlleva. Por otro lado, los string se almacenan como tipo unsingned char*, y al trabajar con arrays se debe conocer siempre el tamaño de estos. Ninguna de las limitaciones presentadas suponía inconveniente alguno en continuar con el desarrollo del proyecto mediante este sistema.

Atendiendo al principal impedimento que presenta Rosserial, el cual se comentó en el inicio de este apartado, solo el uso de este sistema, para la subscripción un tópico, consume aproximadamente un 70% de la memoria de una placa Arduino común. En el caso de tópicos como el Odom, que contiene la información relativa a la odometría del robot, no sería posible publicarlos por falta de memoria, salvo en el Arduino MEGA.

El protocolo, el cual define una comunicación serial, utiliza los mismos mensajes que emplea el ROS estándar, pero incluyendo un encabezado y cola del paquete que permiten que varios tópicos compartan un enlace serial común. El formato del paquete es el que se muestra en la siguiente tabla.

Primer byte	Bits de sincronización (Depende de la distribución de ROS)
Segundo byte	Bits de sincronización (Versión del protocolo)
Tercer byte	Tamaño del mensaje (Byte menos significativo)
Cuarto byte	Tamaño del mensaje (Byte más significativo)
Quinto byte	Suma de control de la longitud del mensaje
Sexto byte	ID del tópico (Byte menos significativo)
Séptimo byte	ID del tópico (Byte más significativo)
Siguientes bytes	Mensaje serial
Último byte	Suma de control del ID del tópico y datos del mensaje

Tabla 15. Contenido de los bytes en los mensajes de ROS

De forma previa al comienzo de la comunicación, el sistema ROS realiza una consulta al microcontrolador, en este caso el Arduino, para recopilar los nombres y los tipos de los tópicos que se van a publicar o subscribir.

En cuanto a la sincronización del tiempo, esta se maneja transfiriendo un mensaje del tipo std_msgs::Time en cada envío, de modo que junto a la información que contiene el mensaje, se puede conocer el momento en el que este se ha generado.

2.8.3. Ros_Arduino_Bridge

Es un paquete [26] que permite realizar una comunicación entre Ros y un microcontrolador Arduino, empleando un protocolo diseñado expresamente para ello, y simplificado respecto al modelo que presenta el Rosserial. Se reduce de este modo el costo computacional asociado a su utilización. Incorpora un nodo Python que se ejecuta en el ordenador, el cual se encarga de traducir estos mensajes al formato estándar de ROS.

A pesar de que no guarda ninguna relación con Rosserial, en el modelo básico del sistema, es necesario que exista una conexión física entre los equipos. En este caso mediante un cable USB tipo B.

El modelo base del paquete permite recopilar información sobre el estado de los pines digitales y analógicos del microcontrolador. También es posible controlar los pines de salida digital o incluso controlar servomotores.

Está pensado para controlar robots implementados en Arduino, a partir de los mensajes y servicios que ofrece ROS. Debido a esto, incluye un controlador base para el control de robots diferenciales, el cual, a partir de un mensaje de ROS tipo Twist, el cual expresa la velocidad lineal y angular deseada en cada uno de los 3 vectores posibles, permite controlar el movimiento de los motores, recopilando posteriormente la información de la odometría. Para que todo esto sea posible hay que indicar el controlador que se va a utilizar, el tamaño de las ruedas instaladas o los parámetros para el controlador PID, entre otros ajustes necesarios. Un robot diferencial como el que se ha mencionado es aquel que cuenta con dos ruedas motrices, una a cada lado del robot, que habitualmente van a compañas por una o dos ruedas giratorias para equilibrar el robot. Sin embargo, este modelo dista bastante del diseño de robot omnidireccional que se ha realizado, por lo que no fue posible sacar provecho a estas características.

De los planteamientos anteriores se puede deducir que, si se quería utilizar este paquete para controlar el movimiento del robot, iba a ser necesario realizar importantes modificaciones para poder adaptarlo a los requisitos del diseño, en especial a que la comunicación se produjese a través se señales Wifi.

Para poder utilizar esta herramienta de comunicación serial, hay que disponer en el sistema operativo del paquete *Python Serial*, el cual se puede instalar en el caso de Ubuntu, empleando la siguiente instrucción.

```
$ sudo apt-get install python-serial
```

Por otro lado, la versión del Arduino IDE debe ser la 1.6.6 o superior.

Junto a todo lo anterior, es necesario que el usuario del sistema operativo en el que se conectará el cable USB, esté añadido al grupo "dialout", otorgando de este modo los permisos para usar el dispositivo conectado al puerto USB. La instrucción a utilizar es la siguiente.

```
$ sudo usermod -a -G dialout <usuario>
```

Donde <usuario> es el nombre de usuario que se utiliza para iniciar sesión en el sistema operativo.

El último paso de la preparación para poder utilizar el Ros_Arduino_Bridge es instalar este paquete en el ordenador. Para ello, desde la ventana se comandos, se accede a la carpeta src del espacio de trabajo catkin, y se introducen las instrucciones que se presentan a continuación.

```
$ git clone https://github.com/hbrobotics/ros_arduino_bridge.git
$ cd ../..
$ catkin_make
```

También hay que instalar la librería *ROSArduinoBridge* en el Arduino IDE. Para ello se accede a la carpeta en la que se guardan habitualmente los códigos de Arduino, y se introduce la instrucción de la siguiente línea.

```
$ cp -rp `rospack find ros_arduino_firmware`/src/libraries/ROSArduinoBridge ROSArduinoBridge
```

Con todo lo anterior preparado ya se podía subir el código a la placa Arduino. Por su parte, en ROS hay que iniciar el nodo ros_arduino_python, lo cual se puede hacer a partir de un *launch file*.

```
$ roslaunch ros_arduino_python arduino.launch
```

De este modo ya es posible acceder a la información de la placa Arduino mediante los servicios que ofrece la herramienta Ros_Arduino_Bridge. Por ejemplo, para leer el valor analógico del pin 2 de la placa se utilizaría:

```
$ rosservice call /arduino/analog_read 2
```

Se puede acceder a toda esta información a través del propio monitor serial del Arduino IDE. Para ello hay que configurar dicha ventana a 57600 baudios, estableciendo el retorno de carro como "Ambos NL & CR". En la siguiente tabla se muestran la lista de comandos que ofrece el paquete por defecto.

ANALOG_READ	'a'
GET_BAUDRATE	'b'
PIN_MODE	'c'
DIGITAL_READ	'd'
READ_ENCODERS	'e'
MOTOR_SPEEDS	'm'
PING	'p'
RESET_ENCODERS	'r'
SERVO_WRITE	's'
SERVO_READ	't'
UPDATE_PID	'u'
DIGITAL_WRITE	'w'
ANALOG_WRITE	'x'

Tabla 16. Órdenes predefinidas en Ros_Arduino_Bridge

De este modo, si como en el ejemplo anterior se desea hacer leer el valor analógico del pin 3 de la placa, se introduciría por el monitor serie "a 2".

Para conocer más información de este paquete se puede consultar el enlace de descarga [27] del mismo en GitHub.

2.9. **Odometría**

La odometría de un robot móvil como el que se ha diseñado, consiste en la estimación de su posición durante la navegación. Esta información se puede obtener mediante diversas técnicas, pero en el caso de este proyecto, aprovechando que se han instalado motores paso a paso, resulta sencillo conocer cuánto ha girado cada rueda.

Por otro lado, la pose del robot es el conjunto de parámetros que son necesarios para situar el sistema en el espacio. Estos parámetros en el caso de este proyecto, y dado que el robot se mueve dentro de un plano, consisten en la posición en el eje x, la posición en el eje y, y la rotación respecto a un eje, en este caso el eje x. Todo esto último habiendo considerado previamente un sistema de referencia cartesiano. De este modo, el primer paso para elaborar un modelo de odometría para un robot consiste en establecer una posición inicial respecto al sistema de coordenadas. La pose será un vector de 3 variables.

Pose = { X , Y , Θ}

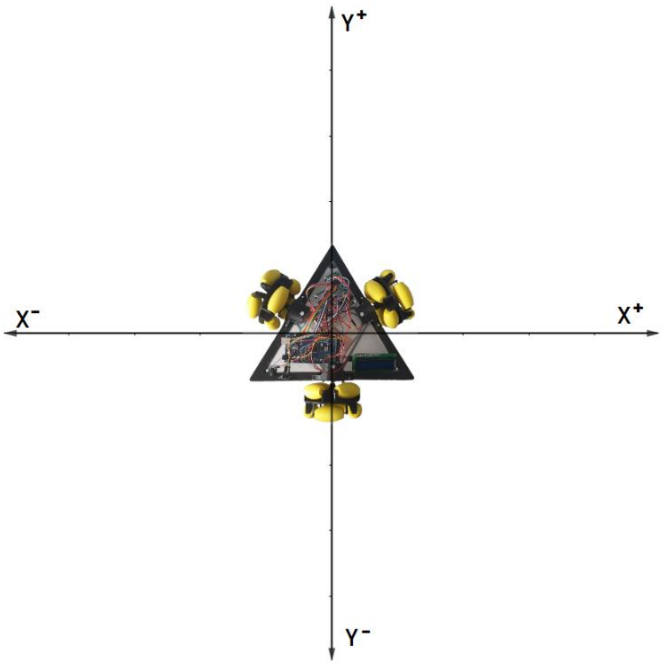


Figura 45. Referencia del sistema odométrico

La referencia que se ha escogido es la que se muestra en la imagen. Al iniciar la ejecución del programa, se definirá un sistema de coordenadas de modo que el robot esté ubicado en el punto (0, 0) del mismo. De este modo, la parte delantera del robot, que es el vértice en el que se han instalado los controladores de los motores, estará en la dirección del eje y, en el sentido positivo del mismo. Esta consideración supone que haya que establecer un ángulo inicial, respecto al eje x, de 90 º. Este ángulo se considera positivo ya que se ha definido como positivo el sentido de rotación antihorario, y como negativo el sentido horario.

Un dato que resulta fundamental conocer antes de proceder a la obtención de las distintas expresiones, con las que se realizará la estimación de la posición, es el avance que realiza cada rueda por cada paso. Este cálculo se puede realizar conociendo que los motores paso a paso 28BYJ-48, por cada 512 pasos, producen una rotación en el eje de 360º.

Ecuación 2

$$Avance = 1\ paso \cdot \frac{360º\ eje}{512\ pasos} \cdot \frac{2 \cdot \pi \cdot r}{360º\ eje}$$

Donde r es el radio de la rueda omnidireccional, el cual, en base al diseño realizado, es de 5.2 cm. De este modo se obtendría el avance en centímetros:

Ecuación 3

$$Avance = 1\ paso \cdot \frac{360º\ eje}{512\ pasos} \cdot \frac{2 \cdot \pi \cdot 5.2\ cm}{360º\ eje} \cong 0.0638\ cm$$

En otro sentido de las ideas, el movimiento de las ruedas omnidireccionales puede contribuir a que el robot realice rotaciones puras, traslaciones puras, o movimientos mixtos. El modelo que se ha elaborado consiste en separar estos dos primeros tipos de

movimientos para analizarlos por separado. De este modo, se estudiará la rotación y la traslación del robot por separado, asegurando que las expresiones que se obtengan para un tipo de movimiento no afecten al otro.

En relación con la rotación, cada rueda describe un arco respecto al centro geométrico del robot.

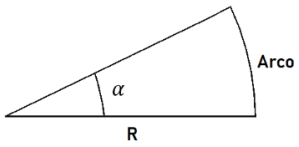


Figura 46. Arco que describen las ruedas respecto al centro del robot

En un movimiento de rotación pura, todas las ruedas describirán un arco de circunferencia, el cual se puede obtener a partir de la siguiente expresión:

Ecuación 4

$$Arco = 2 \cdot \pi \cdot R \cdot \frac{\alpha}{360º}$$

En relación a esta expresión, dado que se calculará esta contribución por cada paso del robot, el Arco se corresponde con el valor antes calculado para el avance. Por su parte, R es la distancia del centro geométrico del robot al punto de apoyo en cada una de las ruedas. Este punto varía durante la navegación, ya que en algunos momentos entra en contacto con el suelo los rodillos de una hilera, y en otros los de otra hilera de la rueda omnidireccional.

Figura 47. Hileras de las ruedas omnidireccionales diseñadas

De este modo, y teniendo en cuenta que cada hilera entrará en contacto con la superficie sobre la que se esté moviendo el robot la mitad del tiempo, la distancia R se tomará hasta el centro de la rueda omnidireccional. Este valor, atendiendo al diseño elaborado, es de 12.1 cm. Introduciendo en la expresión este valor en centímetros, de igual forma que para el Avance, se obtiene el ángulo descrito en el movimiento de rotación en grados, que es la unidad en la que se quiere almacenar esta información.

Ecuación 5

$$\alpha = \frac{Avance \cdot 360º}{2 \cdot \pi \cdot R} = \frac{0.0638\,cm \cdot 360º}{2 \cdot \pi \cdot 12.1\,cm} \cong 0.3021º$$

Para que se produzca un cambio de ángulo en el robot como el descrito, es necesario que giren todas las ruedas en el mismo sentido. Por este motivo, la contribución que se asignará a cada motor para la rotación será un tercio del valor obtenido.

Ecuación 6

$$\alpha m = \frac{\alpha}{3} = \frac{0.3021º}{3} \cong 0.1007º$$

Si los tres motores provocan un giro del robot en sentido antihorario:

Ecuación 7

$$\frac{\alpha m}{3} + \frac{\alpha m}{3} + \frac{\alpha m}{3} = \alpha$$

Si, por el contrario, se provoca un giro en sentido horario:

Ecuación 8

$$-\frac{\alpha m}{3} - \frac{\alpha m}{3} - \frac{\alpha m}{3} = -\alpha$$

Como se ha podido comprobar, el modelo es válido para los movimientos de rotación pura, pero es necesario comprobar que esto no afecta a los movimientos de traslación. En este tipo de movimientos, mientras que un motor gira en un sentido, otra gira en sentido contrario, generando un desplazamiento del robot en la dirección del eje de la tercera rueda, la cual debe estar parada. Cuando ocurre esto, la suma de contribuciones a la rotación puede ser:

Ecuación 9

$$\frac{\alpha m}{3} - \frac{\alpha m}{3} + 0 = 0$$

O, si se desplaza en el sentido contrario:

Ecuación 10

$$-\frac{\alpha m}{3} + \frac{\alpha m}{3} + 0 = 0$$

De este modo, solo resta analizar el caso de desplazamiento puro en el eje horizontal. Para este tipo de movimiento se mueven dos ruedas en un sentido y la tercera en el otro. Sin embargo, también en este caso se compensan las contribuciones ya que esta tercera rueda gira a mayor velocidad que las otras. El caso más sencillo de analizar es el del movimiento del robot sobre el eje X cuando el ángulo de la pose es de 90º, es decir en condiciones iniciales. En este caso las dos primeras ruedas giran en un sentido, a la misma velocidad, mientras que la tercera, que gira en sentido contrario, lo hace al doble de velocidad. Debido a esto, al ir sumando las contribuciones en el momento de la

ejecución, se sumará el doble de contribuciones por parte de este tercer motor. Este comportamiento se puede representar mediante la siguiente expresión:

Ecuación 11

$$\frac{\alpha m}{3} + \frac{\alpha m}{3} - \frac{2 \cdot \alpha m}{3} = 0$$

O, en el caso del desplazamiento en el sentido contrario:

Ecuación 12

$$-\frac{\alpha m}{3} - \frac{\alpha m}{3} + \frac{2 \cdot \alpha m}{3} = 0$$

Esta compensación descrita se produce para todos los movimientos de traslación que permite realizar el robot, lo cual se puedo comprobar posteriormente de forma experimental. En este punto, resulta conveniente mencionar que el movimiento de una única rueda, estando las otras dos detenidas, no puede provocar una rotación pura. Esta situación no ha presentado ningún problema en este caso, ya que, en ningún movimiento de los que puede describir el robot, se activa una única rueda.

Se concluye por tanto con la determinación de la contribución a la rotación de cada motor, tras haber verificado que el modelo elaborado no se ve afectado por los movimientos, que no son de rotación pura, y que puede realizar el robot.

El movimiento de traslación debe modificar los valores x e y de la pose. En este caso, la contribución será en el sentido de avance de la rueda, el cual también se verá afectado por el ángulo respecto al eje x en el que se encuentre el robot en cada momento.

A continuación, se describirán las expresiones con las que se realizarán los cálculos para determinar, en cada momento, la contribución que genera para la traslación el movimiento de cada motor.

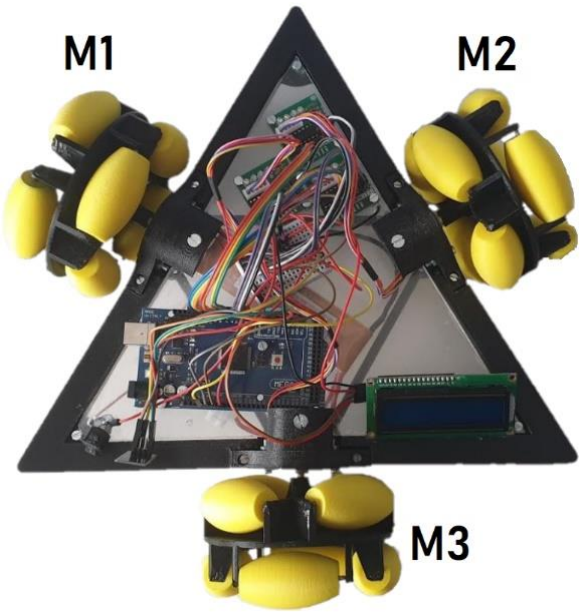


Figura 48. Disposición de los motores en el robot omnidireccional

Antes de continuar, conviene recordar que por cada paso se recorre una distancia que venía dada por el término que se denominó Avance. Por otro lado, las contribuciones se calcularán para la referencia que se ha escogido, es decir, con el robot centrado en el eje de coordenadas y con un ángulo respecto al eje x de 90º. Debido a esto, cuando sea necesario el ángulo respecto al eje x en las expresiones que se presenten a continuación, se deberá sumar la variación en el ángulo que se haya producido hasta el momento, por la navegación del robot. Esta diferencia se puede calcular sumándole al ángulo que se utilice en las expresiones, el ángulo recogido en la pose actual y restándole los 90º desde los que se parte. Se obtendrán las expresiones trabajando en grados, pero de cara a una futura implementación en el robot, se tendrá que trabajar en radianes, por lo que se expondrán ambas expresiones.

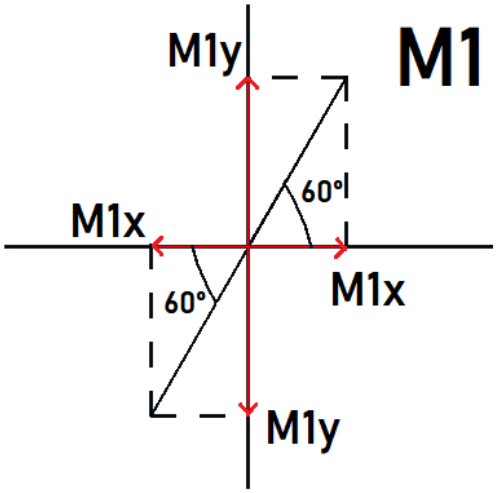


Figura 49. Posibles contribuciones al desplazamiento del motor 1

En la anterior representación se muestra, para la posición de referencia en la que se ha situado el robot, las posibles contribuciones que se puede producir, descompuestas en los ejes principales. En este caso el desplazamiento tendrá lugar con un ángulo de 60º respecto al eje de las abscisas, mientras que el signo dependerá del sentido en el que se mueva el robot.

En cuanto al primer motor, si éste avanza hacia el primer sector:

Ecuación 13

$$M1x = Avance \cdot \cos(60º + \theta - 90º) = Avance \cdot \cos(\theta - 30º) \text{ [cm]}$$

Ecuación 14

$$M1x = Avance \cdot \cos\left(\theta - \frac{\pi}{6}\right) \text{[cm]}$$

Ecuación 15

$$M1y = Avance \cdot \text{sen}(60º + \theta - 90º) = Avance \cdot \text{sen}(\theta - 30º) \text{ [cm]}$$

Ecuación 16

$$M1y = Avance \cdot \text{sen}\left(\theta - \frac{\pi}{6}\right) \text{[cm]}$$

Por otro lado, en el caso de avanzar hacia el tercer sector del eje de coordenadas:

Ecuación 17

$$M1x = -Avance \cdot \cos(60º + \Theta - 90º) = -Avance \cdot \cos(\Theta - 30º) \text{ [cm]}$$

Ecuación 18

$$M1x = -Avance \cdot \cos\left(\Theta - \frac{\pi}{6}\right) \text{[cm]}$$

Ecuación 19

$$M1y = -Avance \cdot \text{sen}(60º + \Theta - 90º) = -Avance \cdot \text{sen}(\Theta - 30º) \text{ [cm]}$$

Ecuación 20

$$M1y = -Avance \cdot \text{sen}\left(\Theta - \frac{\pi}{6}\right) \text{[cm]}$$

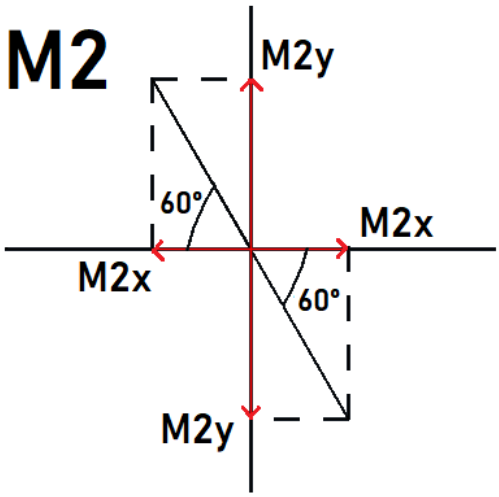


Figura 50. Posibles contribuciones al desplazamiento del motor 2

En este caso, la representación de las contribuciones al desplazamiento que se pueden producir, para este segundo motor, es muy similar al caso anterior, pero utilizando en este caso el ángulo de 30º, resultante de restar 60º a los ángulos rectos, para obtener la descomposición de los desplazamientos. De nuevo, los signos de estas contribuciones dependerán del sentido en el que se desplace el robot.

Por otro lado, atendiendo al segundo motor, si se avanza hacia el segundo sector:

Ecuación 21

$$M2x = -Avance \cdot \text{sen}(30º + \Theta - 90º) = -Avance \cdot \text{sen}(\Theta - 60º) \text{ [cm]}$$

Ecuación 22

$$M2x = -Avance \cdot \text{sen}\left(\Theta - \frac{\pi}{3}\right) \text{[cm]}$$

Ecuación 23

$$M2y = Avance \cdot \cos(30º + \Theta - 90º) = Avance \cdot \cos(\Theta - 60º) \text{ [cm]}$$

Ecuación 24

$$M2y = Avance \cdot \cos\left(\Theta - \frac{\pi}{3}\right) \text{[cm]}$$

Si, por el contrario, la rueda realiza un movimiento hacia el cuarto sector:

Ecuación 25

$$M2x = Avance \cdot \text{sen}(30º + \Theta - 90º) = Avance \cdot \text{sen}(\Theta - 60º) \text{ [cm]}$$

Ecuación 26

$$M2x = Avance \cdot \text{sen}\left(\Theta - \frac{\pi}{3}\right) \text{[cm]}$$

Ecuación 27

$$M2y = -Avance \cdot \cos(30º + \Theta - 90º) = -Avance \cdot \cos(\Theta - 60º) \text{ [cm]}$$

Ecuación 28

$$M2y = -Avance \cdot \cos\left(\Theta - \frac{\pi}{3}\right) \text{[cm]}$$

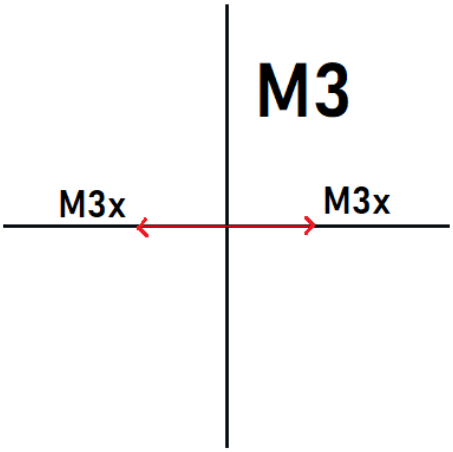


Figura 51. Posibles contribuciones al desplazamiento del motor 3

En el caso del tercer motor del robot omnidireccional, para la colocación que se ha tomado como referencia, solo se puede producir un desplazamiento en el eje de las abscisas, cuyo signo dependerá de si el robot se traslada hacia la derecha o hacia la izquierda. Sin embargo, esto solo ocurre en esta posición concreta, mientras que cuando se produce una rotación en el robot, y en este caso atendiendo a los valores absolutos, la componente del eje de ordenadas irá aumentando al mismo tiempo que la componente sobre el otro eje se ve reducida. Esta situación da lugar a las expresiones que se presentan a continuación.

Por último, y en este caso describiendo el comportamiento del tercer motor, si este produce un movimiento del robot hacia el sentido positivo del eje X:

Ecuación 29

$$M3x = Avance \cdot \cos(\Theta - 90º) \text{ [cm]}$$

Ecuación 30

$$M3x = Avance \cdot \cos\left(\Theta - \frac{\pi}{2}\right) \text{[cm]}$$

Ecuación 31

$$M3y = Avance \cdot \text{sen}(\Theta - 90º) \text{[cm]}$$

Ecuación 32

$$M3y = Avance \cdot \text{sen}\left(\Theta - \frac{\pi}{2}\right) \text{[cm]}$$

En cambio, si el movimiento es en el sentido negativo del eje x:

Ecuación 33

$$M3x = -Avance \cdot \cos(\Theta - 90º) \text{[cm]}$$

Ecuación 34

$$M3x = -Avance \cdot \cos\left(\Theta - \frac{\pi}{2}\right) \text{[cm]}$$

Ecuación 35

$$M3y = -Avance \cdot \text{sen}(\Theta - 90º) \text{[cm]}$$

Ecuación 36

$$M3y = -Avance \cdot \text{sen}\left(\Theta - \frac{\pi}{2}\right) \text{[cm]}$$

Para estas últimas expresiones, si se sustituye un ángulo de 90º se anulan las contribuciones en y, tal y como se refleja en la última imagen.

Como ya se ha mencionado, lo que se ha considerado como positivo o negativo en las anteriores expresiones podría cambiar con la rotación del robot, pero este efecto se encarga de corregirlo la propia expresión.

Con el motor 3 moviéndose hacia el sentido positivo del eje x, en la posición establecida como referencia, a 90 grados, se obtendría lo siguiente:

Ecuación 37

$$M3x = Avance \cdot \cos(90 - 90º) \cong 0.0638\, cm$$

Si debido a la navegación del robot, el robot ha girado 180º, y por tanto el valor del ángulo de la pose es de 270º:

Ecuación 38

$$M3x = Avance \cdot \cos(270º - 90º) \cong -0.0638\, cm$$

En este caso la contribución sería negativa, reflejando lo que ocurre en la realidad.

Retomando parte de las explicaciones que se presentaron para el caso de la rotación pura, hay que verificar que estas expresiones dejan inalterados los valores de x e y en la pose cuando se produce una rotación.

Si el robot gira en sentido antihorario, la contribución total a los ejes x e y serían las siguientes:

Ecuación 39

$$X = -Avance \cdot \cos(\Theta - 30º) - Avance \cdot \text{sen}(\Theta - 60º) + Avance \cdot \cos(\Theta - 90º) = \text{= Avance} \cdot (\text{-cos}(\Theta - 30º) - sen(\Theta - 60º) + \cos(\Theta - 60º)) = Avance \cdot (-\cos(\Theta) \cdot \cos(30º) - sen(\Theta) \cdot sen(30º) - sen(\Theta) \cdot \cos(60º) + \cos(\Theta) \cdot sen(60º) + \cos(\Theta) \cdot \cos(90º) + \text{sen}(\Theta) \cdot sen(90) = Avance \cdot \left(-\frac{\sqrt{3} \cdot \cos(\Theta)}{2} - \frac{\cdot \text{sen}(\Theta)}{2} - \frac{\cdot \text{sen}(\Theta)}{2} + \frac{\sqrt{3} \cdot \cos(\Theta)}{2} + sen(\Theta)\right) = 0$$

De nuevo se comprueba que el modelo es el correcto, ya que en un movimiento de rotación pura no se altera el valor de la posición en X de la pose. Ocurre esto mismo para el valor de la posición en y. En ambos casos, se obtiene el mismo resultado si el sentido de giro es el opuesto ya que la expresión sería la misma pero cambiada de signo.

Para la última deducción que se ha expuesto se han empleado las siguientes razones trigonométricas:

Ecuación 40

$$\text{sen}(\alpha - \beta) = sen(\alpha) \cdot \cos(\beta) - \cos(\alpha) \cdot sen(\beta)$$

Ecuación 41

$$\cos(\alpha - \beta) = cos(\alpha) \cdot \cos(\beta) - \text{sne}(\alpha) \cdot sen(\beta)$$

Se puede dar por concluido el método teórico que se propone en este documento para el estudio de la odometría del robot omnidireccional. No obstante, conviene mencionar que, a la hora de realizar una posterior implementación práctica, se observarán desviaciones entre el valor estimado y la posición real del robot. Esto se debe a las imprecisiones en las medidas, el ruido del sistema y el que las ruedas puedan deslizarse, junto a otros efectos que no se han modelado en este trabajo. Se tendrá en cuenta esto último a la hora de realizar la implementación del método expuesto, ajustando las expresiones para lograr una mejor estimación de la pose.

Capítulo III. Realización práctica

Sobre la base de las ideas presentadas en el capítulo anterior, se puede comenzar a describir el modelo experimental que se ha desarrollado en este TFG. En este trabajo se ha diseñado un robot para posteriormente fabricarlo a partir de piezas impresas en 3D. También se ha diseñado su sistema de control, el cual, con motivo de la existencia del propósito de que el robot fuese inalámbrico, requirió de realizar múltiples adaptaciones, tanto del software como del hardware.

Otros aspectos del proyecto, que se darán a conocer en este tercer capítulo, son la implementación del sistema para el cálculo de la odometría del robot, y el software que se ha desarrollado para controlar el robot con un mando, empleando la plataforma que ofrece ROS.

3.1. Ruedas del robot

Dentro del conjunto de piezas que se han fabricado para el robot, son las ruedas las que han requerido un diseño más riguroso, ya que se ello dependía el correcto funcionamiento del robot en cuanto a su navegación.

En este sentido, y tal y como se detalló en el apartado destinado a la explicación de las ruedas omnidireccionales, este diseño permite una gran maniobrabilidad. Un punto que se ha tenido en consideración ha sido que existen distintos modelos de ruedas omnidireccionales, por lo que fue necesario un proceso de selección en el que se tuviese en cuenta las características particulares del robot que se pretendía implementar.

Una vez se contaba con la clara idea de que la mejor opción para el prototipo era una rueda omnidireccional con dos filas de rodillos, ya que de ese modo se llegaba a un equilibrio entre la complejidad del sólido y su funcionalidad, se comenzó el proceso de obtención de un modelo.

Resulta conveniente mencionar que existen repositorios públicos, de diseños funcionales, de piezas modeladas en 3D. Una de las páginas más conocidas es Thingiverse. En ella, y volviendo al caso particular de este proyecto, se pueden encontrar diseños ya preparados de ruedas omnidireccionales. Las ruedas que están instaladas en el robot, así como el resto de las piezas que conforman el prototipo, se han diseñado desde 0 con la versión de estudiantes del software de modelado 3D Inventor Professional, ya que se consideraba que una de las oportunidades que brindaba este proyecto era el aprender a diseñar piezas, para que estas se ajustasen a los requerimientos del robot que se quería fabricar. Un ejemplo que ilustra esto último es el que el eje de la rueda debe encajar de forma precisa en el eje del motor que se escogiese, en este caso el modelo 28BYJ-48. Sin embargo, sí que resulta útil consultar las propuestas de otras personas de cara a buscar soluciones al problema de diseño que se había planteado.

3.1.1. Diseño de las ruedas

Una vez se había decidido cómo debía ser la rueda que se instalase en el robot, se comenzó el proceso de modelado con ayuda del Autodesk Inventor Professional.

La rueda omnidireccional se basaría en una estructura que serviría de unión al eje del motor empleado, y en la que se distribuirían 2 filas en las que se colocarían un total de 8 rodillos. De este modo, y dado que ambas filas están desfasadas 45 grados, se podía garantizar que en todo momento, una de estas 8 pequeñas ruedas que se colocan en la estructura, estaría en contacto con el suelo, permitiendo el desplazamiento singular que ofrece este tipo de piezas.

Con todos los modelos preparados, se generaron los archivos en formato .stl. Este procedimiento era absolutamente necesario para poder cargarlos en el software con el que se prepararon todas las impresiones realizadas en este Trabajo de Fin de Grado, el Ultimaker Cura.

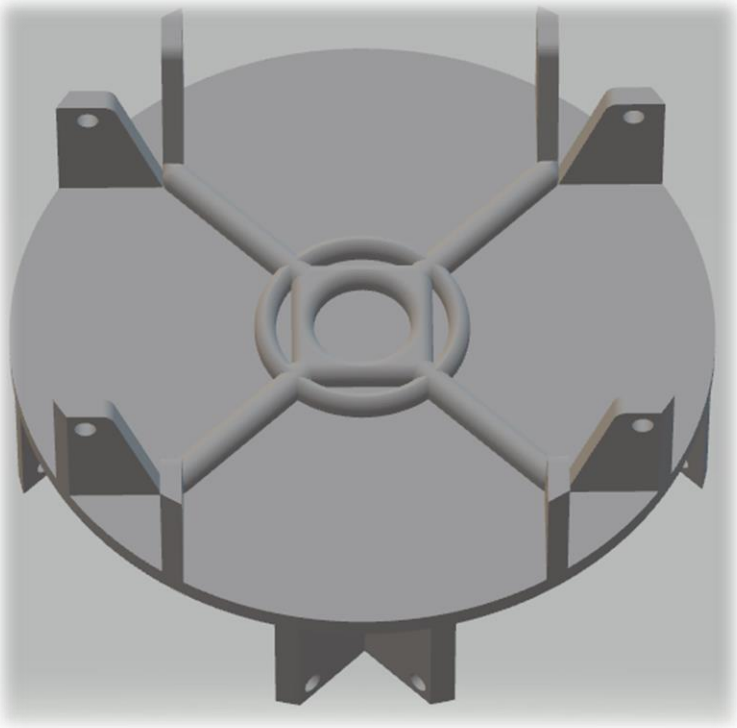


Figura 52. Render de la cara externa del soporte para los rodillos. Primer diseño.

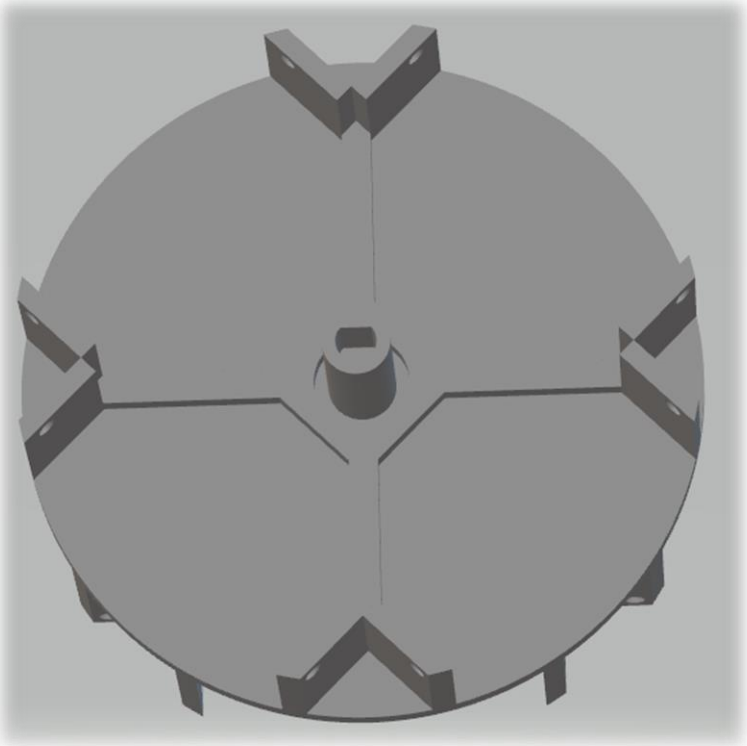


Figura 53. Render de la cara interna del soporte para los rodillos. Primer diseño.

Este modelo fue la primera pieza que se imprimió en el trabajo, y presentó una serie de problemas que hubo que corregir.

En el segundo diseño, el hueco donde se encaja el eje del motor se alargó hasta el otro extremo de la pieza, con el objetivo de que la impresora no crease ninguna estructura de plástico en su interior. Esto se debió a que era muy complicado retirar el plástico que se generaba en el orificio. Por otro lado, y dado que por las características de la pieza era necesario retirar una gran estructura de plástico, creada por la impresora para poder producir las partes horizontales en altura, se unieron los soportes de los rodillos de la cara interna de la rueda, dotándolas de este modo de una mayor resistencia, evitando que se pudiesen romper en la etapa de post procesado.

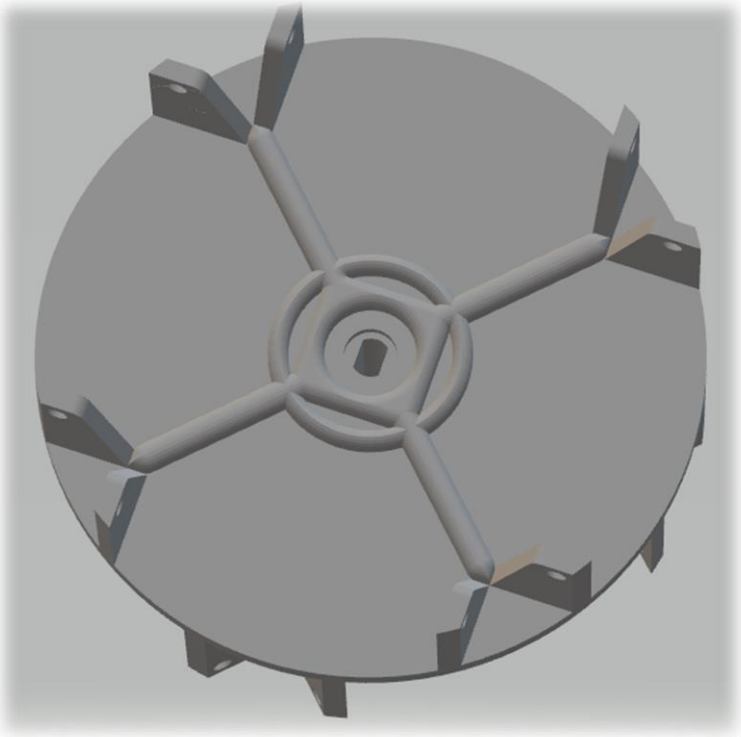


Figura 54. Render de la cara externa del soporte para los rodillos. Segundo diseño.

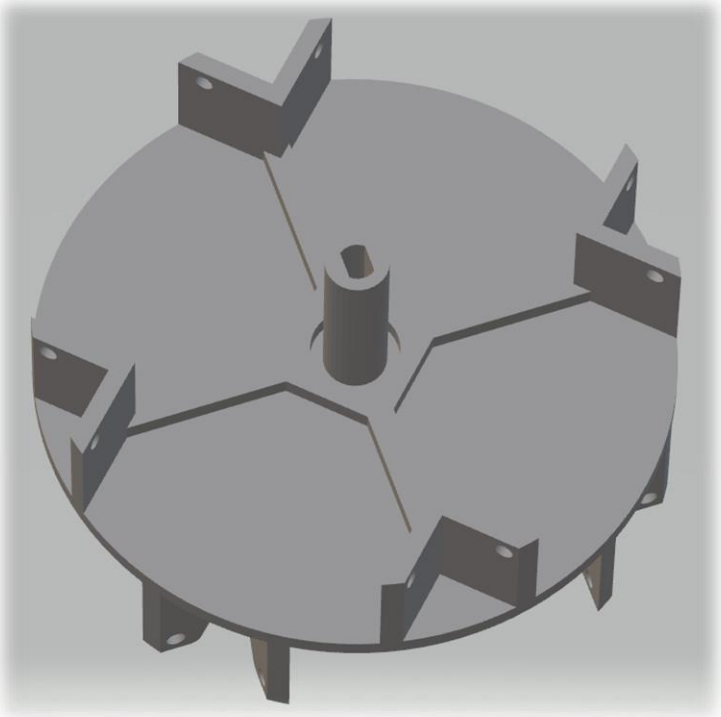


Figura 55. Render de la cara interna del soporte para los rodillos. Segundo diseño.

El nuevo modelo se imprimió de forma completa para poder ir realizando las primeras pruebas de funcionamiento, colocándola en el eje del motor que se iba a emplear. No obstante, y dado que aún faltaba por imprimir otras dos ruedas omnidireccionales completas, se mejoró el diseño. En este caso se modificó el eje de unión entre la pieza

impresa y el eje del motor, para que estos se ajustasen de un modo mucho más preciso. En este sentido, y dado que la impresora tiene ciertas limitaciones, los modelos que esta genera son ligeramente distintos a los diseñados en cuanto a las dimensiones. Esto hacía intuir que sería complicado encajar el eje de la rueda en el motor, por lo que se decidió aumentar la resistencia del eje, incrementando la cantidad de material en torno a este.

Por otro lado, los orificios donde se iban a colocar los ejes de los rodillos se abrieron por la parte superior. Con esto último se consiguió simplificar el proceso de montaje, ya que de este modo solo había que colocar la rueda, con su eje apoyada en los nuevos orificios, y cerrarlos colocando plástico fundido por encima. Por último, se eliminó una hendidura que se había colocado en el modelo, alrededor del eje, que entorpecía el proceso de retirada del plástico sobrante tras la impresión. Adicionalmente se redondearon los bordes de la cara externa.

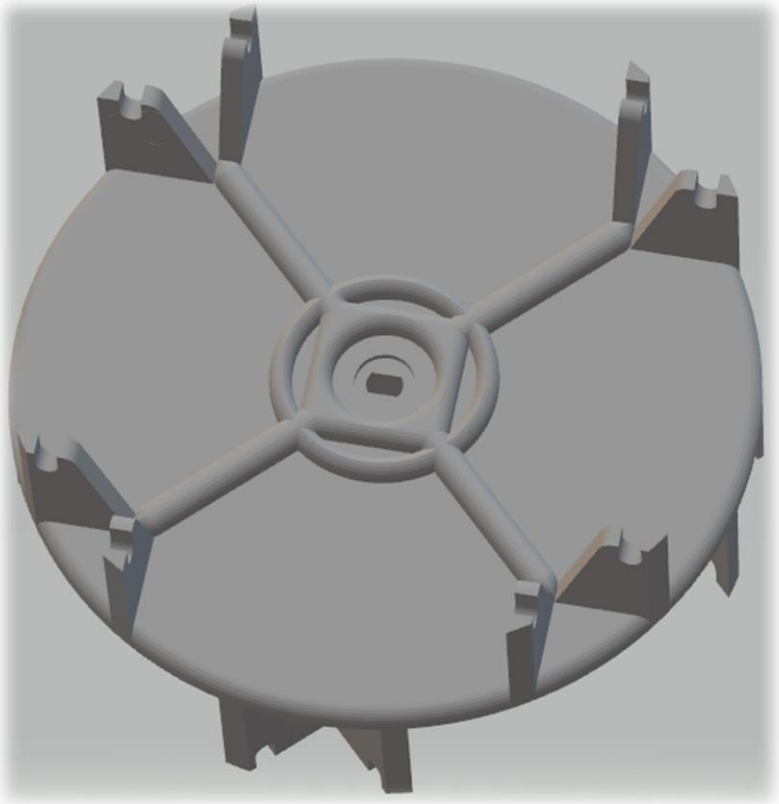


Figura 56. Render de la cara externa del soporte para los rodillos. Tercer diseño.

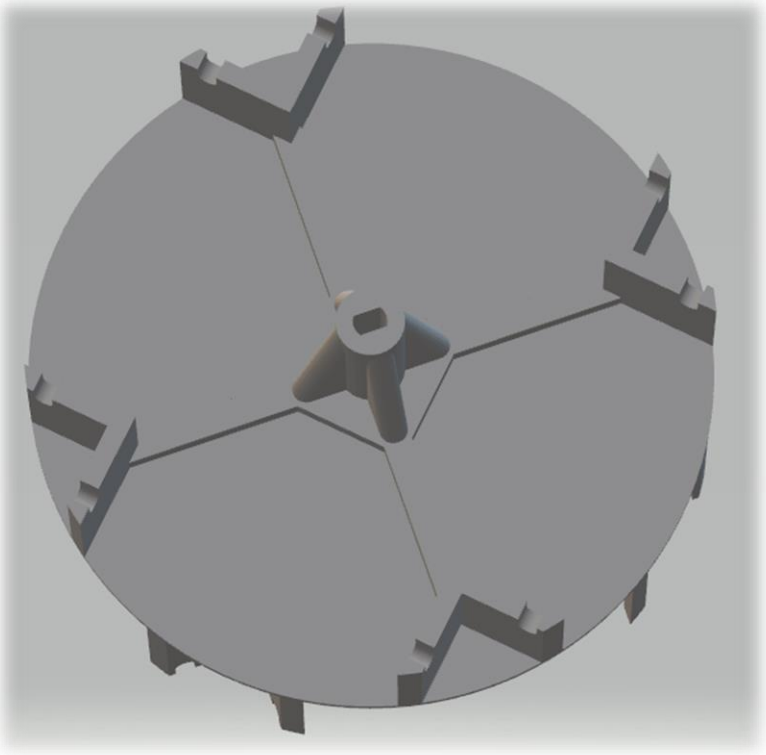


Figura 57. Render de la cara interna del soporte para los rodillos. Tercer diseño.

Con este último diseño se podía dar por concluido el proceso de creación de la estructura de la rueda, pero para conformar una rueda omnidireccional completa hacen falta los rodillos que entran en contacto con el suelo, cuyo diseño resultó más sencillo.

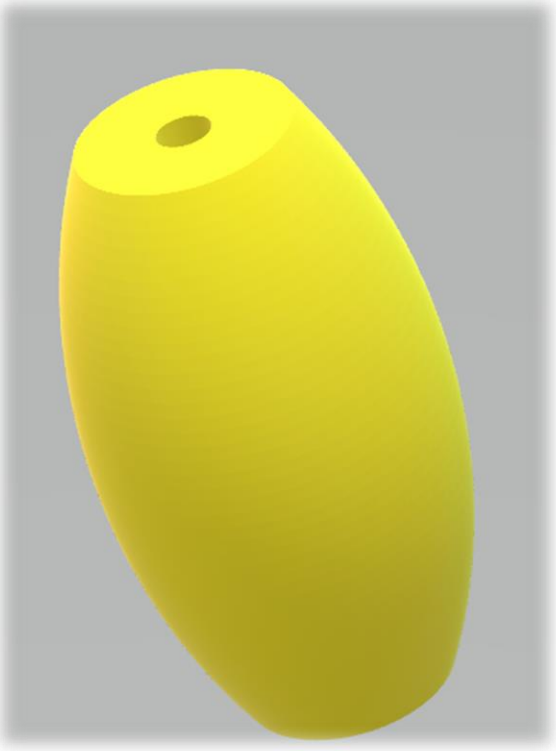


Figura 58. Render de un rodillo

3.1.2. Producción de las ruedas

Existen dos partes fundamentales que constituyen una rueda omnidireccional. Los rodillos y el soporte de estos, en el que se encuentra el eje. Para ambas piezas se utilizó una impresora 3D. Una vez se contaba con el archivo preparado del modelo, este se cargaba en el cura.

Para el soporte se utilizó PLA negro, y los ajustes que se utilizaron son los que se expusieron en el apartado 2.2.3 para este material. Esto se decidió de este modo ya que se pretendía lograr un resultado con una alta resistencia.

En general, la orientación en la que se imprime la pieza es un factor muy determinante, tanto para lograr un buen resultado, como para reducir la cantidad de material que se necesita. Esto último era de especial relevancia en esta pieza ya que, independientemente de la posición en la que esta se colocase en la superficie de impresión, resultaba necesaria una considerable estructura que posteriormente se tendía que retirar. Continuando con este razonamiento, dado que no existían grandes diferencias entre situar la cara interna o la externa de la pieza hacia arriba, se aplicó otro criterio para decidir la orientación. La situación descrita era conocida desde que se comenzó el diseño en Autodesk Inventor. Por este motivo, se decidió que la cara interna estuviese compuesta por superficies más simples, que formasen en la mayoría de los casos ángulos rectos. Junto a esto, se atendió a la resistencia de los soportes de esta cara interna ya que la estructura de plástico se ha de retirar tirando de ella, lo que podría provocar que se rompiese algo. De estos últimos planteamientos se deduce que la mejor opción es situar la cara externa, que cuenta con formas más redondeadas y menos resistentes hacia arriba.

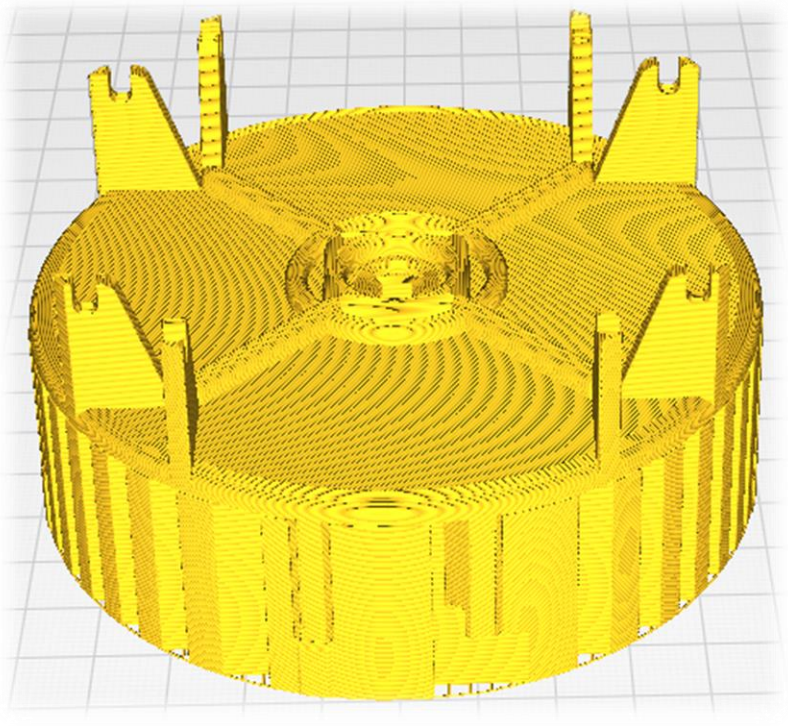


Figura 59. Colocación del soporte para los rodillos en Ultimaker Cura

Para esta pieza el software indica que, de manera aproximada, se consumen 87 gramos de PLA y que el proceso de impresión tiene una duración de 4 horas. Solo cabía una de estas piezas dentro de la superficie de impresión.

Figura 60. Resultado logrado para el soporte para los rodillos

Por otro lado, hubo que imprimir 8 rodillos por cada una de las piezas anteriores. Inicialmente se imprimió esta pieza con PLA, pero el resultando en la práctica era que estas piezas patinaban, por lo que el comportamiento del robot no era el adecuado. Como solución a este problema, se buscó un material distinto, que presentase una mayor adherencia a las superficies. Un material que reúne las características que solicita la pieza a diseñar es el filaflex, que es un filamento flexible, y en este caso se utilizó una bobina de color amarillo.

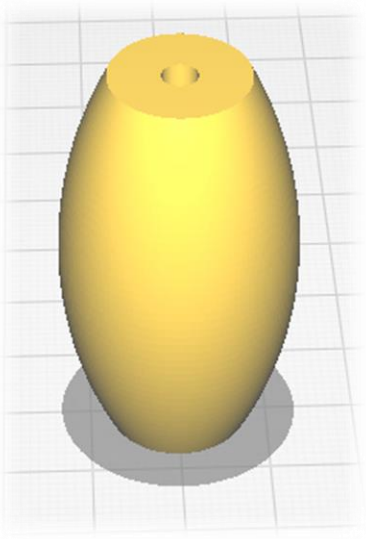


Figura 61. Colocación de rodillo en el Ultimaker Cura

La colocación de la pieza en la superficie de impresión resulta más sencilla en este caso, ya que colocándola del modo en el que se muestra en la anterior ilustración, no es necesario que se genere ningún soporte de plástico que retirar posteriormente. En este caso, el Ultimaker Cura indicaba un consumo y tiempo aproximados de 13 gramos y 3 horas.

Figura 62. Resultado obtenido para el rodillo

En este caso si caben varias de estas piezas de forma simultánea en el área de impresión. Sin embargo, al imprimir con filaflex hay que desactivar la opción de retracción en el extrusor, ya que en caso contrario este podría atascarse. Por esta razón no se puede imprimir más de un sólido al mismo tiempo, debido a que, con motivo de tener esta opción desactivada, el movimiento del extrusor de una pieza a la siguiente dejaría un rastro de material que afectaría al acabado de ambas piezas.

3.2. **Estructura del robot**

Junto a un diseño singular de ruedas, el robot necesitaba que la disposición de estas fuese la adecuada para lograr la alta maniobrabilidad de la que se quiere dotar al sistema. A partir del tipo de rueda omnidireccional escogido, la estructura que mejor se ajustaba era la de un triángulo equilátero, colocando una rueda en cada uno de los lados.

Una vez se había decidido la forma, también era importante que se dejase suficiente espacio en el interior para poder alojar toda la electrónica., dejando un cierto margen, por si se decidía añadir algún hardware al sistema posteriormente.

3.2.1. Diseño de la estructura

Sobre la base de las premisas que se han presentado, en relación a las características que debía tener la estructura, se comenzó a trabajar con el Autodesk Inventor.

Inicialmente se diseñó e imprimió un diseño muy simple, con el objetivo de que este sirviese como base para elaborar posteriormente un modelo mucho más trabajado, una vez se conociesen los distintos componentes que se debían situar en el robot. La impresión se realizó en tres piezas distintas, dado que, por las dimensiones de la superficie de impresión de la impresora, la pieza no cabía entera. Adicionalmente se reservó un espacio para colocar la base de los motores paso a paso, en los que además se dispusieron unos elementos para la unión de las tres partes.

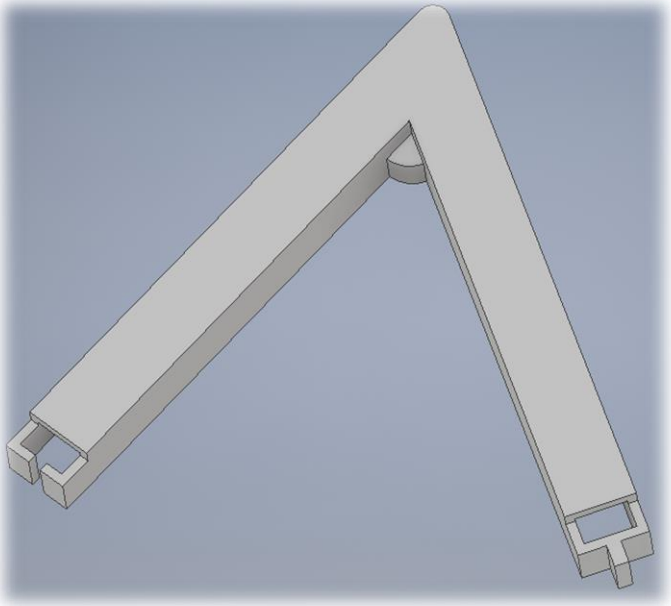


Figura 63. Render de la parte superior de un vértice de la estructura triangular. Primer diseño

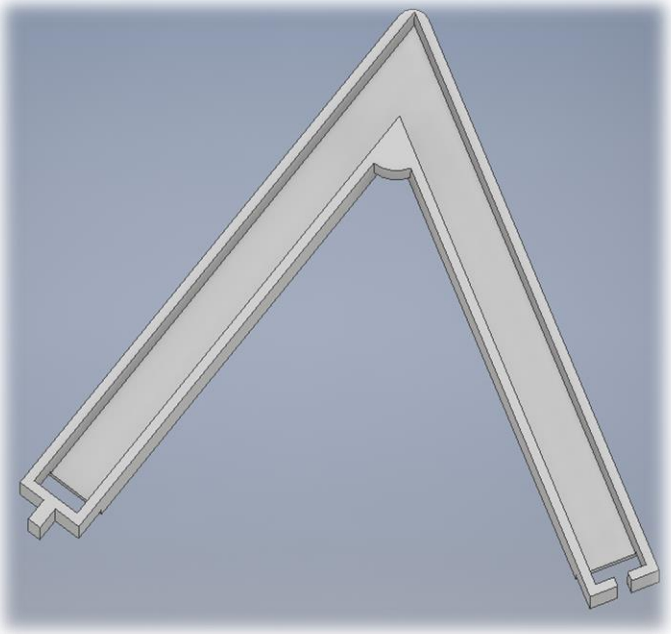


Figura 64. Render de la parte inferior de un vértice de la estructura triangular. Primer diseño

Como se ha mencionado, fue necesario separar en tres piezas la estructura porque no cabía entera en la superficie habilitada para la impresión de la BQ Witbox 2. Con la ayuda del modo de ensamblaje del software de modelado que se ha utilizado en el proyecto, se puede comprobar que el diseño es correcto.

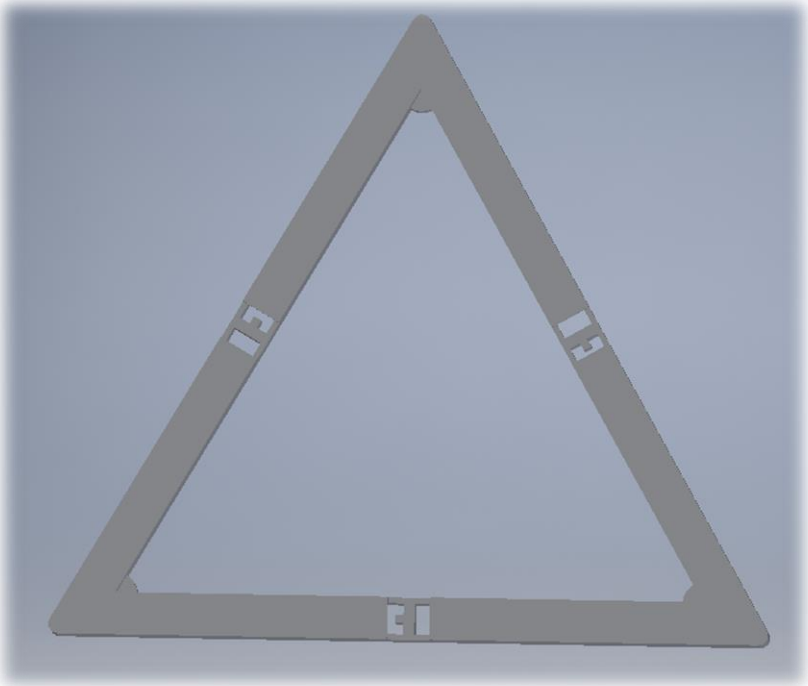


Figura 65. Ensamblaje de la primera estructura diseñada. Parte superior

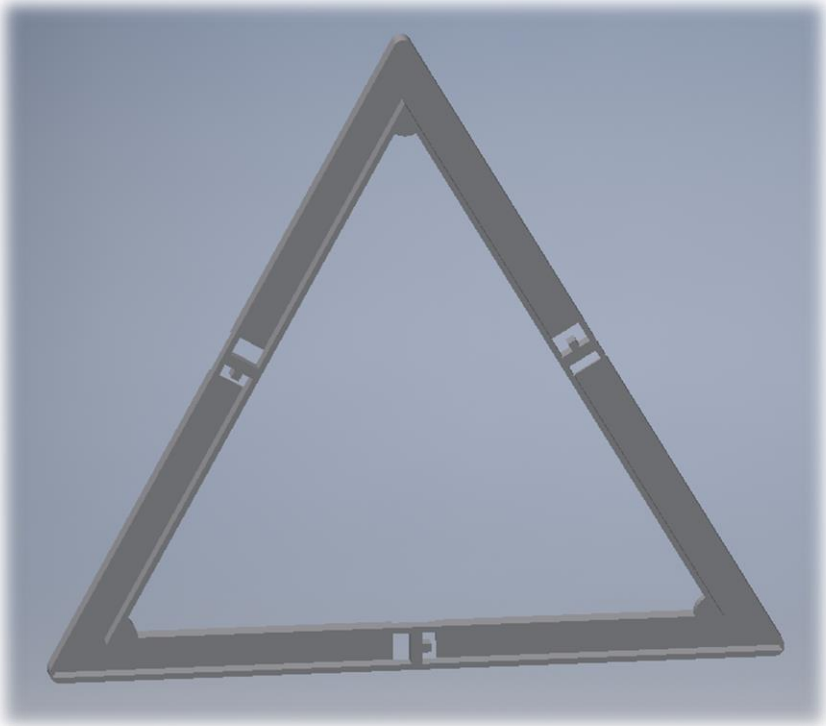


Figura 66. Ensamblaje de la primera estructura diseñada. Parte inferior

Como se mencionó, esta estructura solo era provisional hasta la concreción de todos los componentes que se debían colocar en el robot, además de la forma y tamaño de estos. En los marcos de las observaciones que se realizaron, se apreció principalmente que era necesario diseñar una estructura mucho más robusta y que también incorporase una mayor superficie interior, la cual pudiese albergar los tres controladores de los motores, la placa Arduino que se escogiese, un regulador de tensión, una pequeña Protoboard, la batería, el módulo de comunicación WIFI, un interruptor y que contase con un espacio adicional para poder añadir componentes posteriormente, según se requiriese por el desarrollo del prototipo. Este espacio se utilizó finalmente para colocar una pequeña pantalla LCD. Se ha de mencionar que todos estos componentes se fueron sustituyendo en correspondencia con las distintas opciones que fueron surgiendo, quedando finalmente aquellas piezas que resultaban más ventajosas para la consecución del objetivo del proyecto.

Ante la situación descrita en el párrafo anterior, en primer lugar, la pieza prevista para la estructura triangular exterior se alargó y se rellenó para que quedase maciza (posteriormente, en la etapa de impresión, se seleccionaría un porcentaje de relleno suficiente para que, sin ser maciza, presentase una alta resistencia). Además, se prepararon los extremos para poder añadir en el montaje una pieza que incorporase el motor, y que se uniera mediante tornillos. De este modo se consiguió aumentar el tamaño y la robustez del robot.

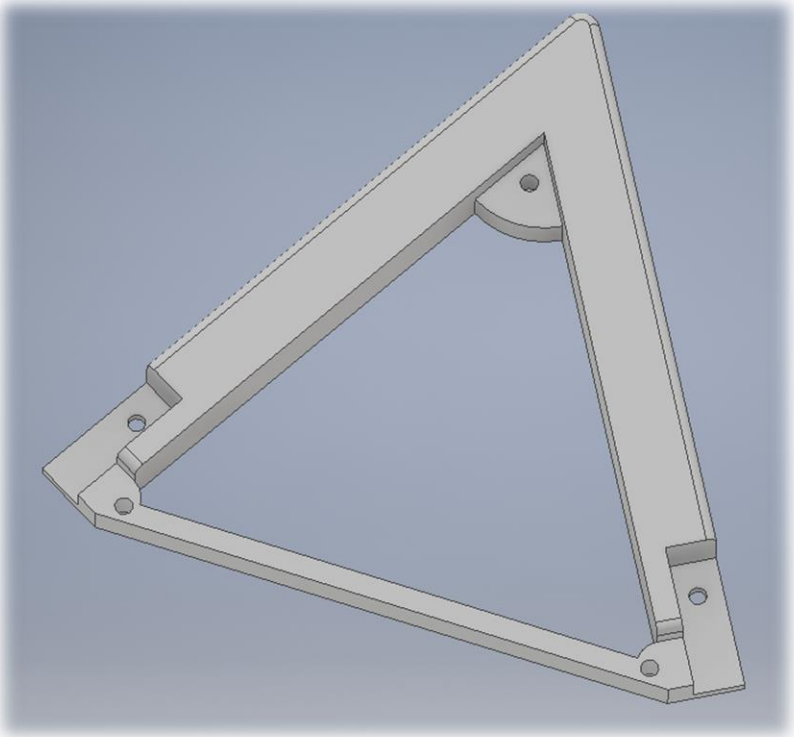


Figura 67. Render de la parte superior de un vértice de la estructura triangular. Segundo diseño

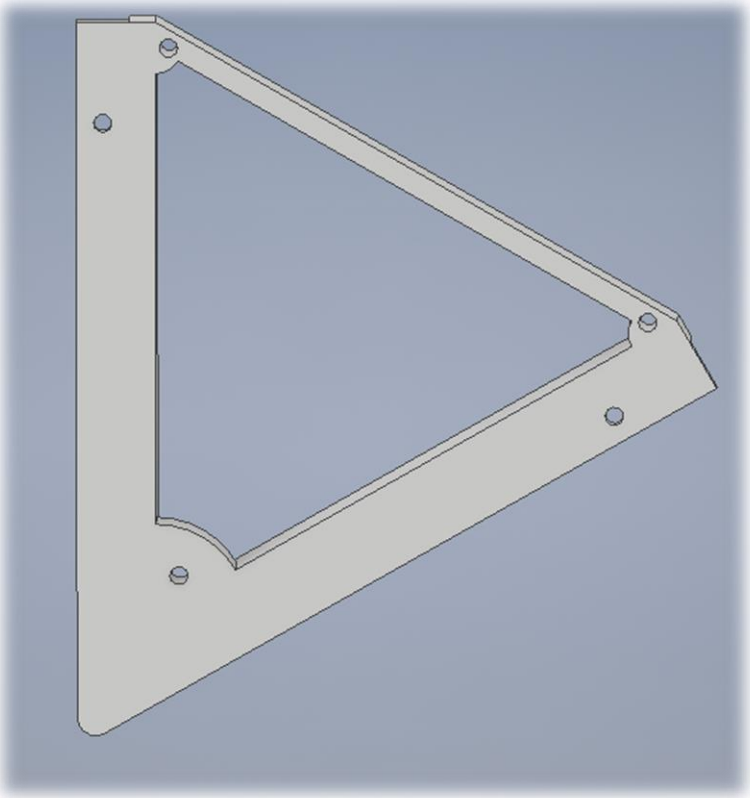


Figura 68. Render de la parte inferior de un vértice de la estructura triangular. Segundo diseño

Por otro lado, aún no se había modelado una pieza con la que se pudiese añadir los motores a la estructura con firmeza. Se imprimió un sólido que quedase bien ajustado por presión al motor, y que en sus extremos contaba con los orificios con los que se uniría a la estructura principal mediante tornillos. Se dotó la pieza de un hueco adicional en la parte superior, que evitaría que la rueda basculase al colocar la rueda en el eje del motor. El modelo se diseñó específicamente para el motor paso a paso 28BYJ-48.

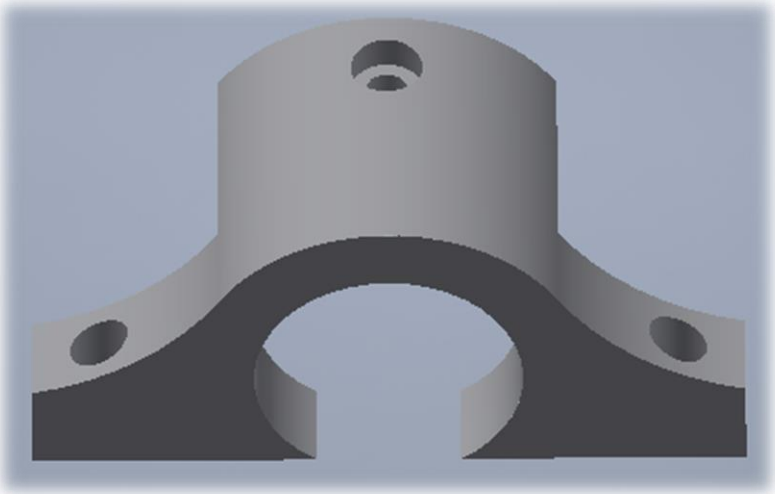


Figura 69. Render del soporte para motor paso a paso. Parte superior

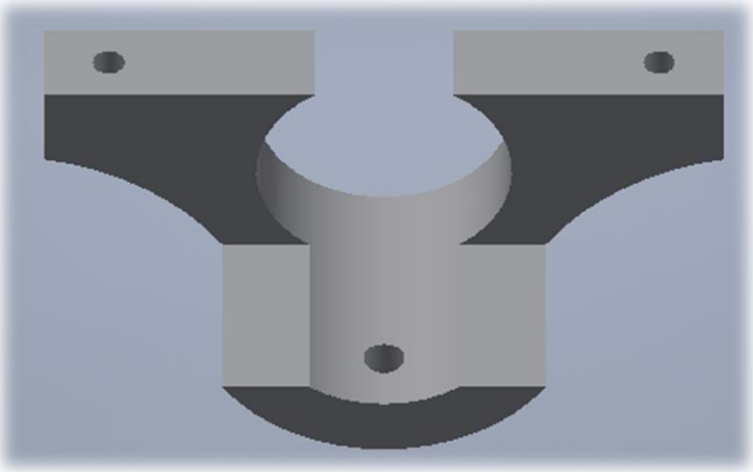


Figura 70. Render del soporte para motor paso a paso. Parte inferior

Con el montaje de las partes mencionadas concluido, solo había que añadir una superficie en la que disponer los distintos componentes del robot, la cual se manufacturó a partir de una plancha de poliestireno. Esta superficie se fabricó con las dimensiones del triángulo que se forma en el interior de la estructura, atendiendo a dejar el espacio necesario para poder colocar los motores paso a paso en su posición, lo cual se puede visualizar en la siguiente imagen.

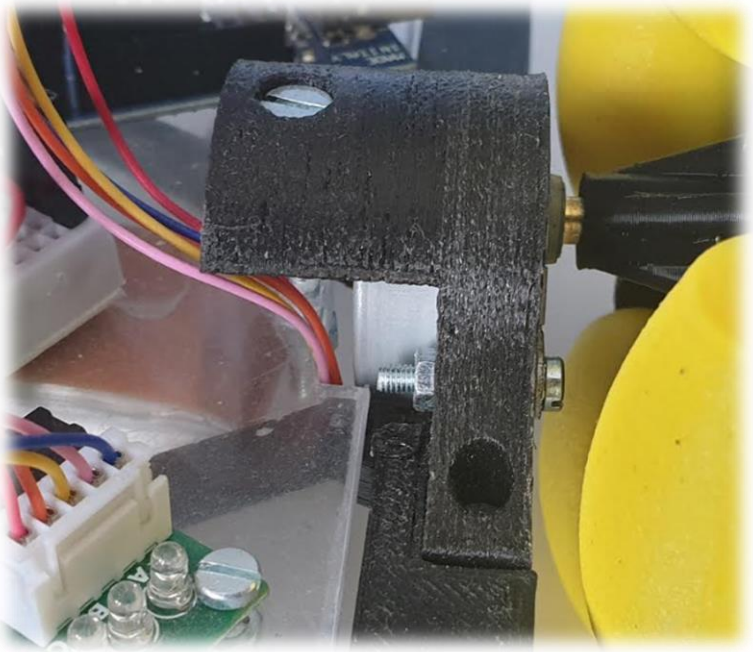


Figura 71. Muesca en la superficie de poliestireno para el paso de cables y colocación del motor

De nuevo, se utilizó el modo de ensamblaje del Inventor para corroborar que las piezas encajaban correctamente.

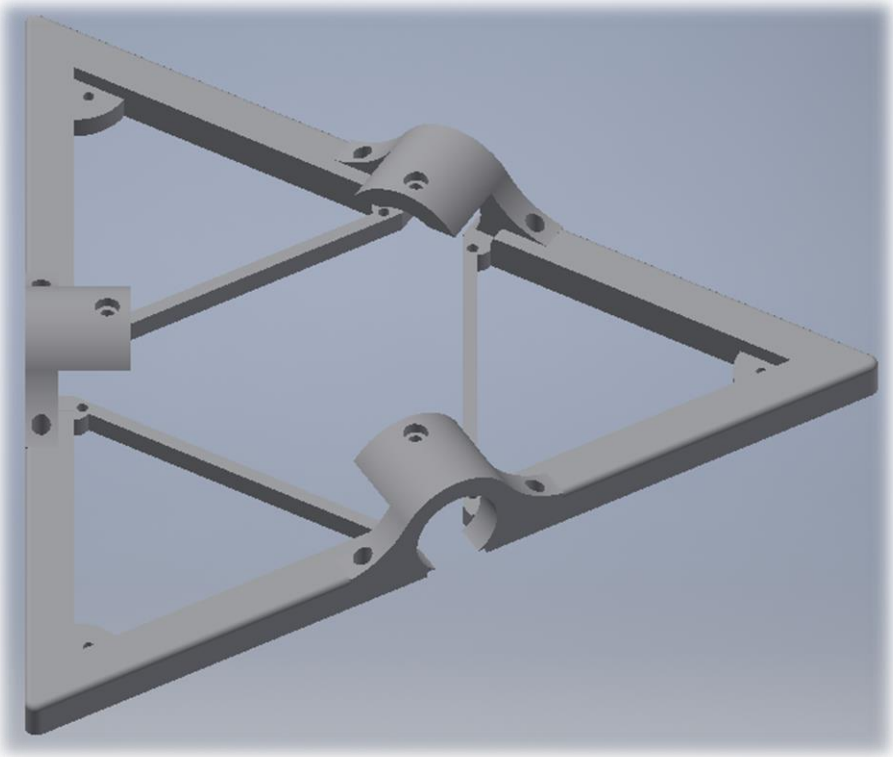


Figura 72. Ensamblaje de la segunda estructura diseñada. Parte superior

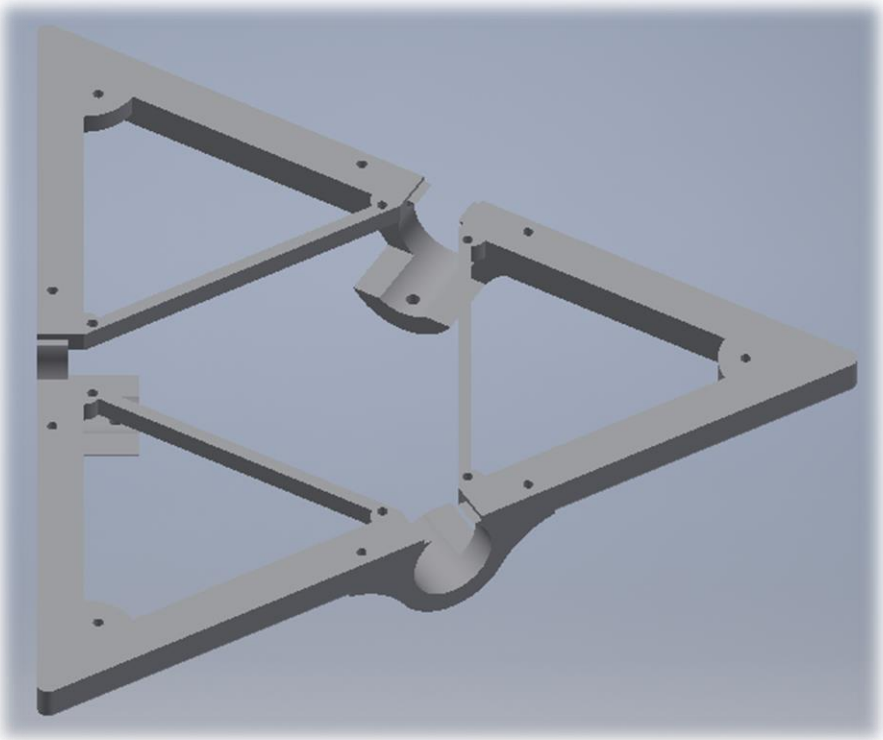


Figura 73. Ensamblaje de la segunda estructura diseñada. Parte inferior

Por otro lado, fue necesario diseñar otras piezas que, sin ser parte de la propia estructura, se colocarían en ella.

En primer lugar, se modeló una pieza cuyo propósito era sostener la batería en la parte inferior del robot, para la cual se deben instalar dos sólidos como el que se muestra en la siguiente imagen.

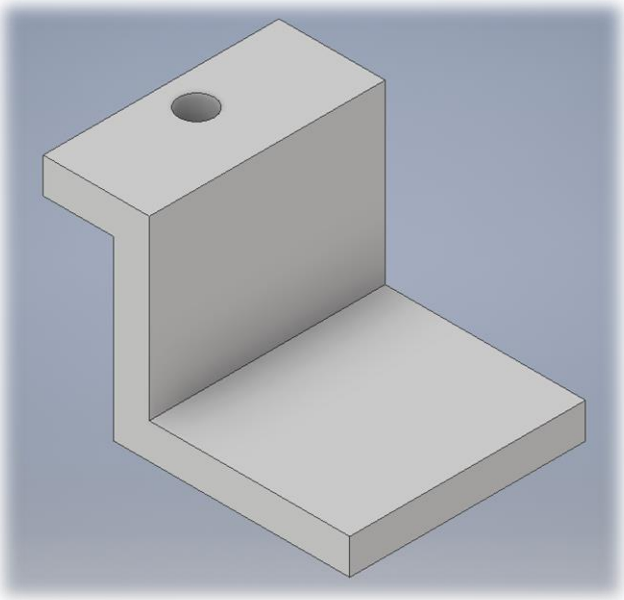


Figura 74. Render del soporte para la batería

Y en segundo lugar, se diseñó una pieza en la que colocar el interruptor de encendido del robot y el módulo ESP-01.

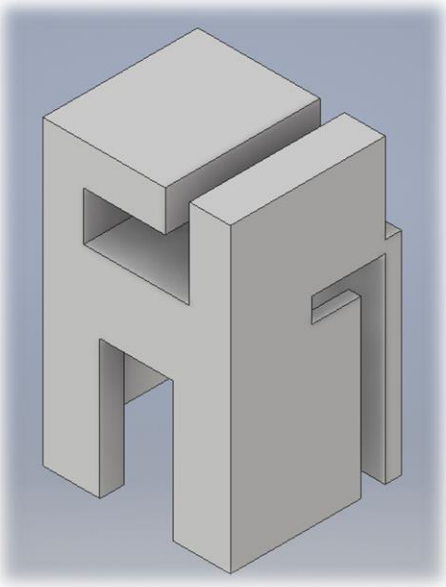


Figura 75. Render del soporte para el módulo ESP-01 y el interruptor de encendido

En la anterior imagen se puede observar que se ha diseñado esta última pieza de modo que se pudiesen colocar ambos componentes con las conexiones ya establecidas previamente, ya que se han habilitado los espacios que eran necesarios para el paso de los cables.

3.2.2. Producción de la estructura

La estructura del robot debe ser resistente, manteniendo los componentes del robot en su interior bien fijados. Se escogió como filamento de impresión PLA de color negro. Los ajustes que se establecieron en el Ultimaker Cura fueron los mismos que para el resto de las piezas de PLA del proyecto, los cuales se describen en el apartado 2.2.3.

La estructura se basa principalmente en 3 piezas que, al unirse a los soportes de los motores colocados entre ellas, forman un triángulo equilátero. Tras un diseño inicial, se realizaron mejoras que dieron lugar a un montaje mejorado respecto al anterior. En primer lugar, hubo que imprimir cada uno de los vértices (con un ángulo de 60 º) que conforman la estructura triangular.

En este caso, existía una orientación que se podía establecer en la impresora 3D que no generaba ninguna estructura por lo que era la óptima.

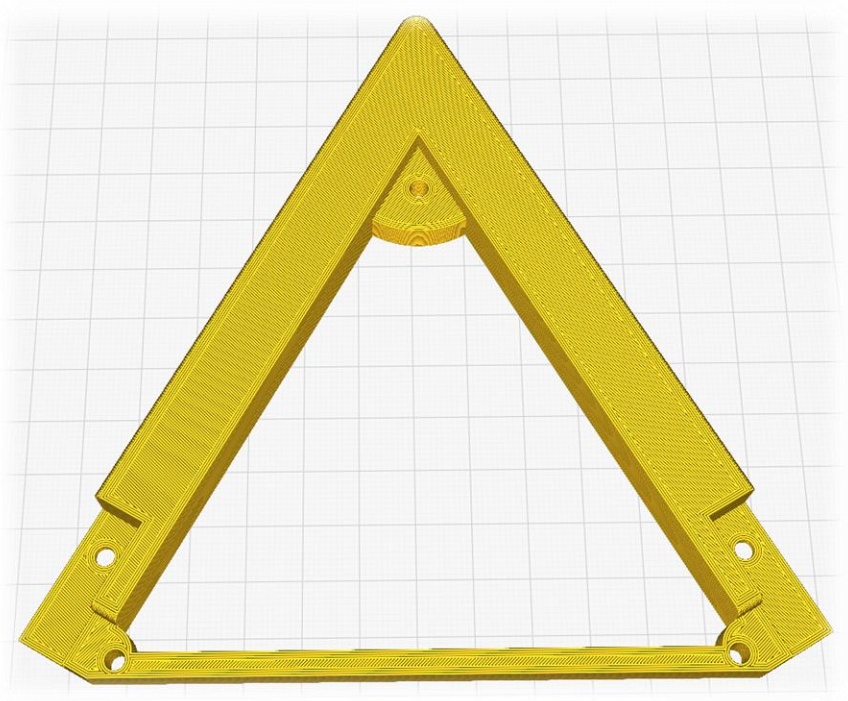


Figura 76. Colocación de un vértice de la estructura en Ultimaker Cura

Para esta pieza, el programa indicaba una duración de la impresión y consumo de PLA estimados de 1 hora y 54 minutos, y 27 gramos respectivamente. Existía también otra posibilidad, que consistía en colocar simultáneamente dos piezas en la superficie de impresión, ya que había suficiente espacio para ello. En este caso, la duración de la impresión aumenta hasta las 3 horas y 51 minutos, con un consumo de 54 gramos.

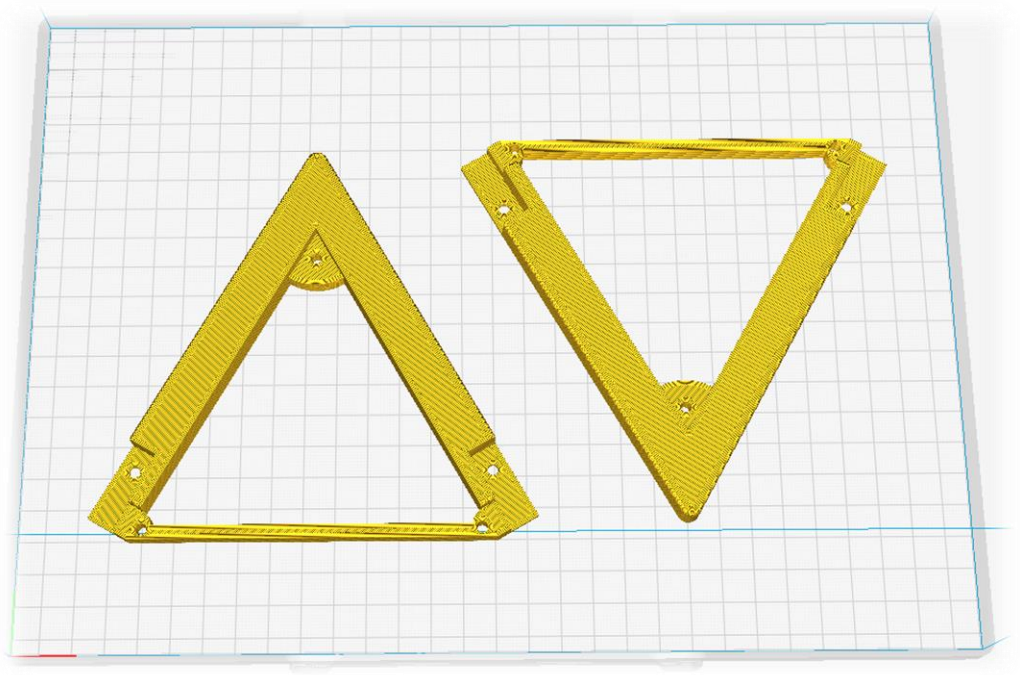


Figura 77. Colocación de dos vértices de la estructura en Ultimaker Cura

El resultado obtenido es el siguiente.

Figura 78. Resultado logrado para un vértice de la estructura triangular

Por otro lado, había que continuar el proceso con la impresión de los soportes de los motores. En este caso, se podían imprimir las 3 piezas simultáneamente, ya que había suficiente área para ello, y gracias a que, al estar empleando PLA, se puede activar la opción de habilitar la retracción en el extrusor, disminuyendo de este modo las imperfecciones que se producen. De nuevo, existía una colocación en la que se evitaba que la impresora crease soportes en la pieza, por lo que se esa fue la que se escogió.

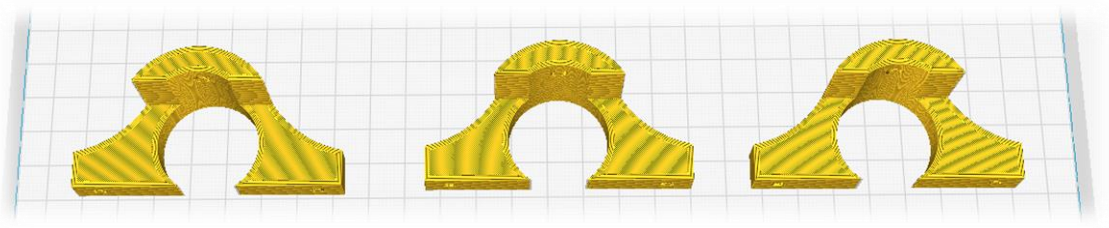


Figura 79. Colocación de tres soportes para motores paso a paso en Ultimaker Cura

En la siguiente imagen se muestra el resultado de la impresión.

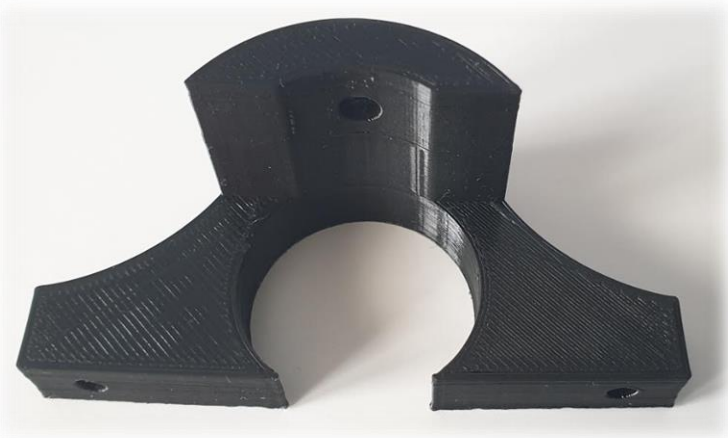


Figura 80. Resultado obtenido para el soporte de los motores

En este caso la duración de la impresión prevista era de 2 horas y 24 minutos, estimando un consumo de PLA de 31 gramos.

Como ya se mencionó en el apartado anterior, los motores se colocan en sus soportes a presión, quedando fijados, mientras que para unir dicha pieza con los vértices de la estructura había que utilizar tornillos, con sus correspondientes tuercas.

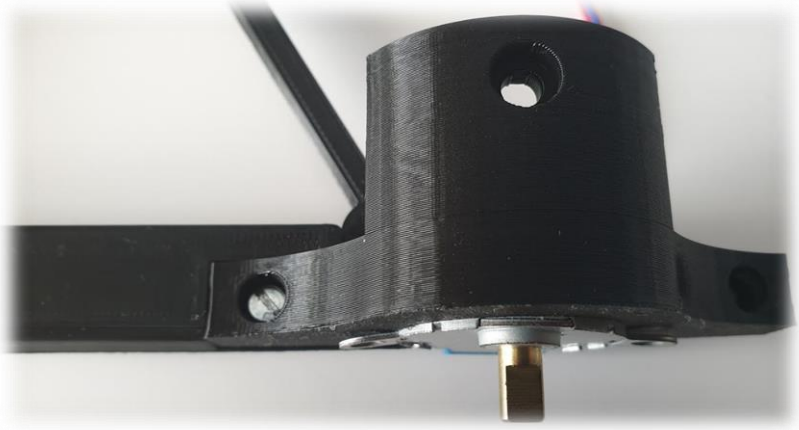


Figura 81. Colocación de tornillos en el modelo

En el espacio interior de la estructura, se colocó una plancha de poliestireno de 4 mm de grosor. Esta serviría de superficie para colocar la electrónica, la cual se fijaría a la superficie con tornillos, empleado los agujeros de sujeción de los componentes para ello.

En relación con la fabricación del soporte de la batería, la mejor colocación es situarla de lado sobre el área de impresión. Además, se podían producir las dos piezas que eran necesarias es una única impresión.

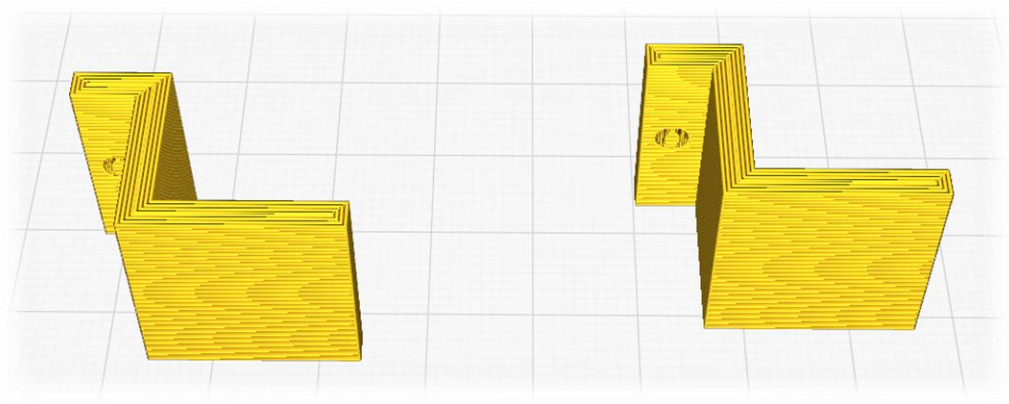


Figura 82. Colocación de los dos soportes para la batería en Ultimaker Cura

La impresión tuvo una duración de aproximadamente 20 minutos, en los que se emplearon unos 5 gramos de PLA negro, obteniéndose el siguiente resultado.

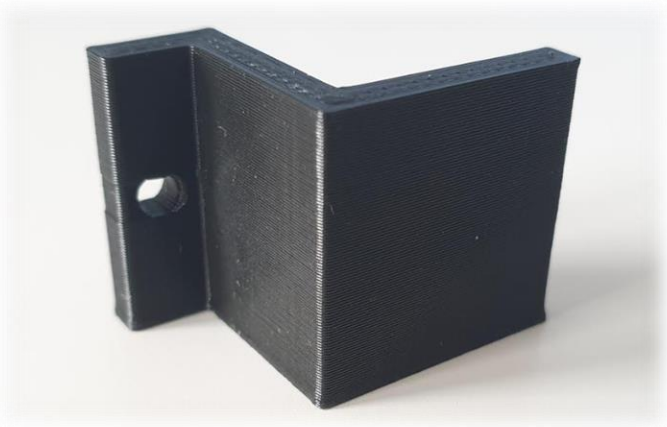


Figura 83. Resultado logrado para el soporte de la batería.

Se concluye este apartado con el soporte para el módulo ESP-01 y el interruptor. En este caso si fue necesario que la impresora crease un soporte de impresión que hubo que retirar, ya que debía disponer un hueco en el interior de la pieza en la que se alojarían las conexiones entre los cables de alimentación y tierra del robot a los terminales del interruptor.

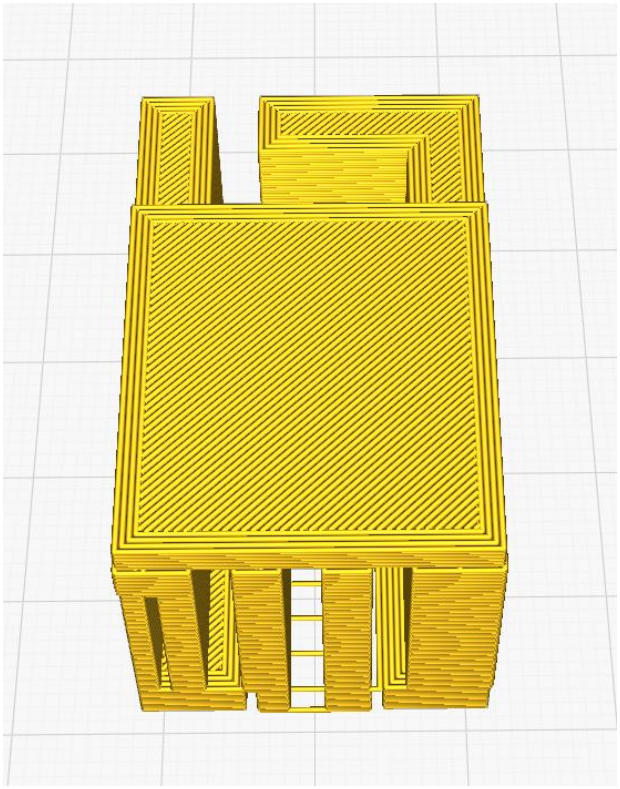


Figura 84. Colocación de los dos soportes para el módulo ESP-01 y el interruptor en Ultimaker Cura

Se decidió que, en este caso, una buena colocación de la pieza era con la parte trasera hacia arriba. El Ultimaker cura mostraba que el consumo iba a ser de 9 gramos, y que la impresión se demoraría unos 40 minutos. Transcurrido este tiempo se obtuvo la última pieza que se necesitaba para el montaje del robot.

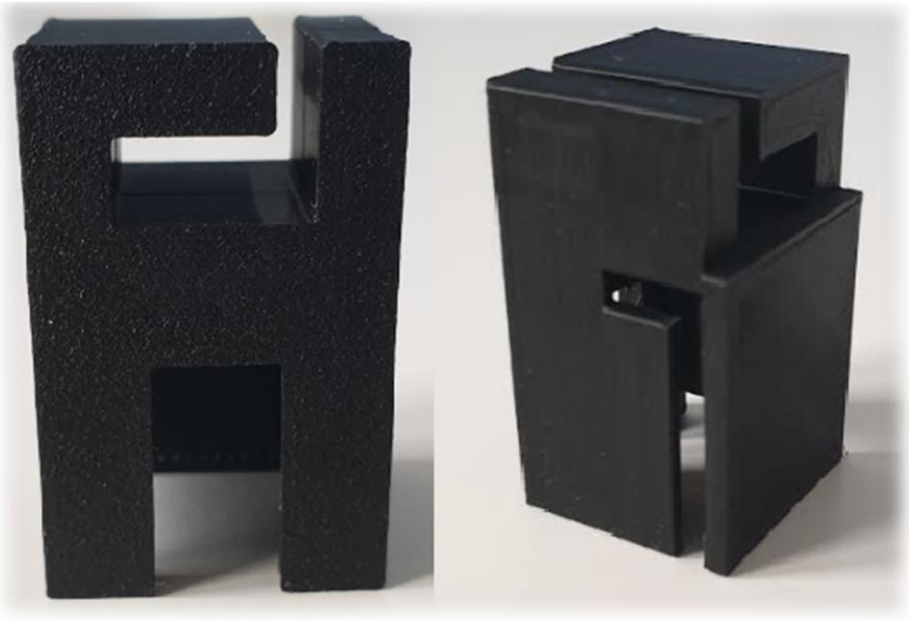


Figura 85. Resultado tras la impresión del soporte para el módulo ESP-01 y el interruptor

3.3. Selección de la placa Arduino

En el transcurso del proyecto se contempló la utilización de distintos modelos de Arduino con los que se iban solventando los distintos requerimientos que se presentaban durante el desarrollo de este.

En primer lugar, y con el objetivo de ir realizando pruebas de funcionamiento para los distintos componentes del robot, se utilizó un Arduino UNO. Este es el modelo más común de la marca, sin embargo, respondía perfectamente a las demandas de aquel momento del desarrollo. De este modo fue posible conocer el funcionamiento de los motores paso a paso y de sus controladores, tanto de forma individual, como con los 3 motores funcionando simultáneamente. En una fase más avanzada del desarrollo, sirvió para aprender a controlar el módulo de pantalla LCD. Adicionalmente, cabe mencionar que fue en esta plataforma en la que se realizó la primera verificación de operatividad del software con el que se realiza la comunicación entre el sistema ROS y el robot, el Ros_Arduino_Bridge.

En el momento, en el que ya se contaba con un robot cuyo movimiento podía ser controlado a través de una comunicación serial, mediante cable USB, se advirtió la necesidad de dotar al robot de algún hardware que le permitiese realizar una conexión mediante Wifi. Existían varias posibilidades en aquel momento. Finalmente se optó por continuar trabajando con un Arduino WeMos D1 R2 con el que se contaba, y el cual alberga un chip de comunicación Wifi, en concreto el ESP8266. Este modelo de Arduino se presentaba como una gran opción ya que es muy similar al Arduino UNO, pero con la ventaja de incorporar el módulo mencionado.

Antes de comenzar a trabajar con el nuevo módulo de comunicación Wifi, se intentó replicar el funcionamiento que se había logrado con la tarjeta anterior, en el que se controlaba el movimiento de las ruedas por serial. Se presentó un primer problema, y es que el pin analógico con el que cuenta el Arduino WeMos D1 no funcionaba al intentar utilizarlo como salida digital. Esto suponía que faltaba un pin para poder controlar todos los motores. Junto a esto, se presentó otro inconveniente al hallar que no se podían utilizar los pines RX y TX ya que estos eran utilizados por el chip ESP8266. De todo lo mencionado en este párrafo se deduce que la tarjeta Arduino WeMos no era la opción óptima con la que seguir trabajando. No obstante, estos inconvenientes se podían solucionar mediante puertas lógicas, lo cual se describió en el capítulo anterior, en el apartado en el que se trató el tema de las puertas lógicas.

A pesar de que con el método descrito se podía continuar trabajando con el Arduino WeMos, existían mejores alternativas, con las que además se contaría con pines adicionales para futuros añadidos para el robot. De este modo se decidió que se debía continuar con una tarjeta Arduino MEGA, a la que se le añadiría un módulo WI-FI ESP-01, ya que la misma no cuenta con capacidad de establecer una conexión Wifi por defecto, como si se podía en el caso anterior.

Como ya se mostró en líneas anteriores, la placa Arduino MEGA ofrece numerosos pines, con lo que esto soluciona los problemas que se habían planteado para el caso del WeMos D1. De este modo, fue sencillo conseguir replicar el funcionamiento del robot en el que el control de los motores paso a paso se realizaba mediante comunicación serial a través de un cable USB. Sin embargo, el Arduino MEGA no cuenta con el chip ESP-12 que incorpora en WeMos D1. Por este motivo fue necesario incorporar a la placa un módulo ESP-01, el cual resultaba más complejo de utilizar.

Como no se pretendía programar el propio ESP-01 sino el Arduino MEGA, y que este se comunicase con el módulo WI-FI, no se contaba con la posibilidad de utilizar las librerías disponibles para el mencionado módulo. Por el contrario, se debía trabajar con los denominados comandos AT, los cuales se describen en el apartado 2.5.4 de la memoria.

Atendiendo a lo expresado en el párrafo anterior, se preparó un código con el que el Arduino pudiese recibir las órdenes de movimiento a través de un dispositivo conectado a su misma red Wifi, para de este modo probar el nuevo módulo que se había instalado.

En aquel momento ya se contaba con un robot que permitía realizar una navegación que aprovechase parte del potencial de las ruedas omnidireccionales. No obstante, se debía avanzar en el desarrollo del sistema de control del robot para que este permitiese realizar movimientos más complejos, y que la comunicación necesaria se realizase a través del módulo Wifi instalado.

Por todo esto, se pensó en la posibilidad de implementar el control del robot a través de un sistema de nodos en ROS. Inicialmente, y debido a que esta herramienta era novedosa en este proyecto, se decidió empezar a trabajar con el Rosserial. En cualquier caso, tanto estas decisiones, como las que acontecieron en etapas posteriores del desarrollo no exigieron un cambio de placa Arduino. De este modo, el modelo final del robot cuenta con un Arduino MEGA.

3.4. Montaje, incorporación de componentes y conexionado

Habiendo descrito el procedimiento para la obtención de las distintas piezas que conforman el robot omnidireccional, se puede comenzar a describir el proceso de montaje.

En la siguiente tabla se detallan todos los elementos que son necesarios para constituir el robot.

Pieza	Cantidad
Rodillos de filaflex	24
Soporte de rodillos de PLA	3
Vértices para la estructura de PLA	3
Superficie interna de poliestireno	1
Soporte para los motores de PLA	3
Soporte para el módulo ESP-01 e interruptor	1
Soporte para batería LiPo de PLA.	1
Arduino MEGA	1
Motor paso a paso 28BYJ-48	3
Controlador para motor paso a paso ULN2003APG	3
Módulo ESP-01	1
Interruptor	1
Protoboard de 170 agujeros	1
Regulador de tensión MP1584EN	1
Batería LiPo 7.4 V, 1500 mAh	1
Tornillos de 4 x 40 mm	3
Tuercas para tornillo de 4 x 40 mm	3
Tornillos de 3 x 16 mm	25
Tuercas para tornillo de 3 x 16 mm	25
Barillas metálicas (o clavos) de 2.2 x 55 mm	24

Tabla 17. Listado de piezas que contiene el robot

En primer lugar, hay que montar cada una de las ruedas omnidireccionales. Para ello, se introduce una barilla metálica por el eje de un rodillo de filaflex y se coloca en los soportes de la estructura de la rueda.

Figura 86. Colocación de los rodillos en sus soportes

El siguiente paso consiste en fijar esta unión fundiendo el plástico que rodea a los extremos de la barilla metálica, empleando por ejemplo, un soldador.

Figura 87. Fijación de la unión entre los rodillos y la estructura de la rueda omnidireccional

Este procedimiento hay que repetirlo para cada uno de los 24 rodillos. De este modo ya se habrán obtenido las 3 ruedas omnidireccionales completas.

Tras ello, se puede pasar al montaje de la estructura. Se colocan los soportes de los motores como elementos conectores de los vértices de PLA, y se fija esta unión empleando tornillos de 3 x 16 mm junto a sus respectivas tuercas, del modo en el que se muestra en la siguiente imagen. Debido a que se han diseñado los soportes de los motores, para que queden ajustados a la carcasa de los motores paso a paso, conviene colocar estos motores en este momento del proceso, ya que posteriormente sería más complicado.

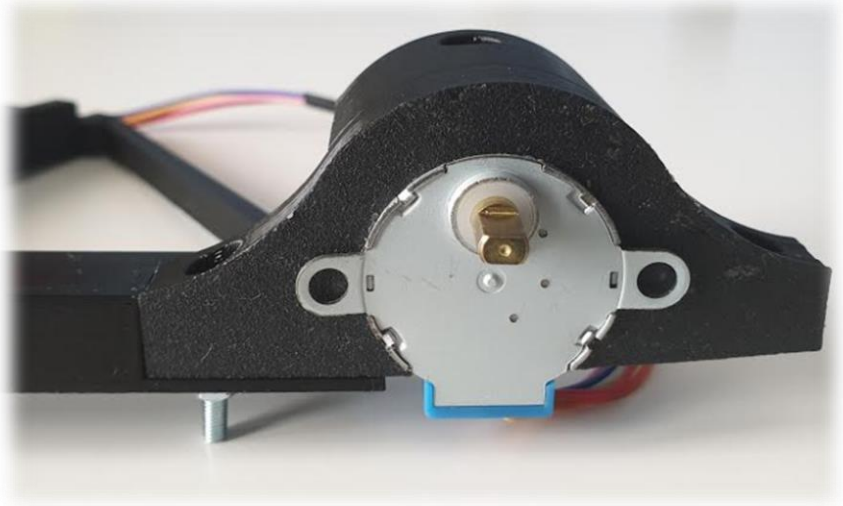


Figura 88. Colocación de los motores paso a paso en la estructura

Los motores quedan ajustados en su posición a presión, pero también es posible emplear tornillos para unir los motores a su correspondiente soporte, si así se desea, necesitando en este caso realizar previamente los agujeros necesarios.

Figura 89. Incorporación de tornillos para sujetar los motores paso a paso

En ese momento ya será posible colocar la superficie de poliestireno donde posteriormente se dispondrán los elementos que contiene el robot. Existen 3 agujeros destinados a colocar en ellos tornillos de 3 x 16 mm que aseguren esta unión.

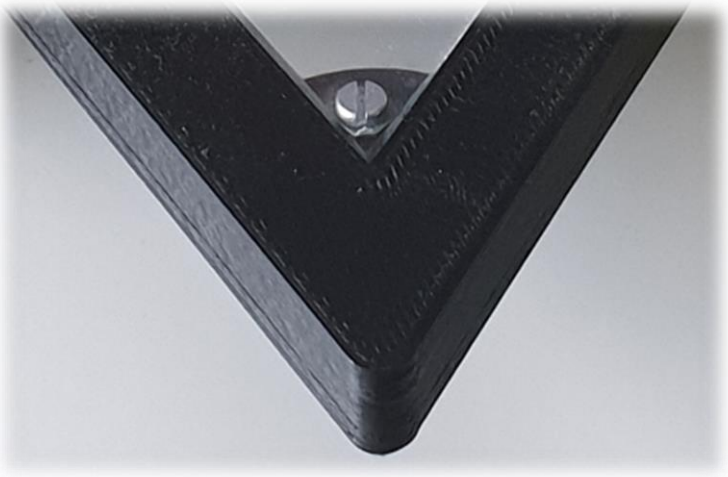


Figura 90. Fijación de la superficie de poliestireno a la estructura del robot

Se continúa con la colocación de la placa Arduino MEGA y de los controladores de motores, taladrando y colocando tornillos de 3 x 16 mm en aquellos agujeros de sujeción de las placas en los que así sea necesario. En concreto se utilizaron 2 tornillos para el Arduino MEGA y otros 2 para cada uno de los controladores de la placa. De este modo quedaban bien adheridos a la superficie del robot. Se coloca también la pequeña Protoboard de 170 puntos en el centro de la estructura, utilizando el adhesivo que incorpora en la cara inferior. Se sitúa el regulador de tensión en esta placa del modo que se muestra en la imagen, fijándolo con la ayuda de una pistola termofusible.

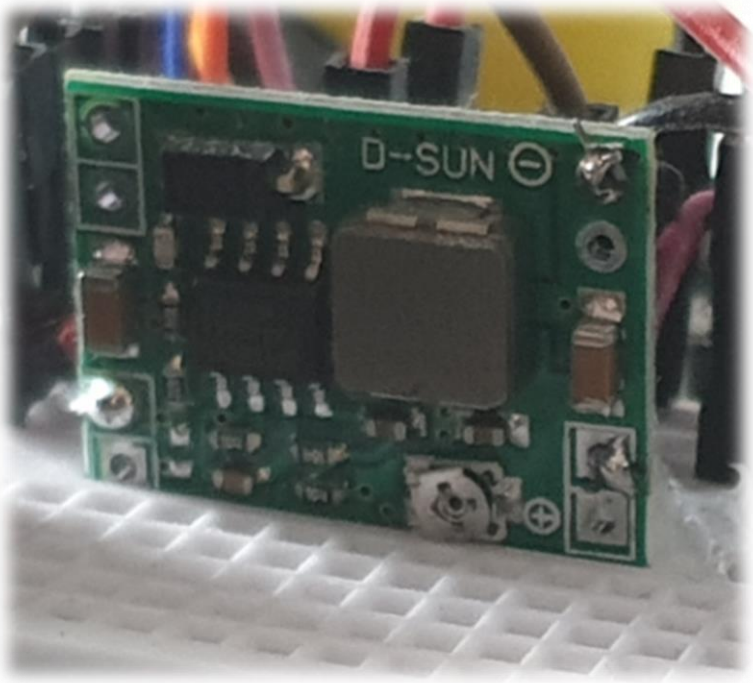


Figura 91. Ubicación del regulador de tensión en el robot.

Se emplea el mismo método de unión para la pantalla LCD.

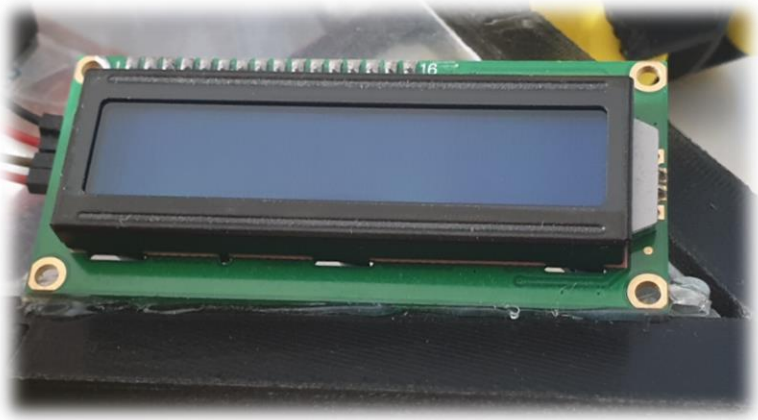


Figura 92. Colocación de la pantalla LCD

Seguidamente, se procede a realizar el conexionado eléctrico de todos los elementos. Con el objetivo de mostrar la forma de hacerlo se exponen las imágenes que se presentan a continuación. Para elaborar estas representaciones se ha utilizado el software Fritzing. En relación con el módulo ESP-01 y al interruptor, hay que incluirlos en este momento en el montaje, aunque aún no se haya colocado el soporte en el que se colocarán.

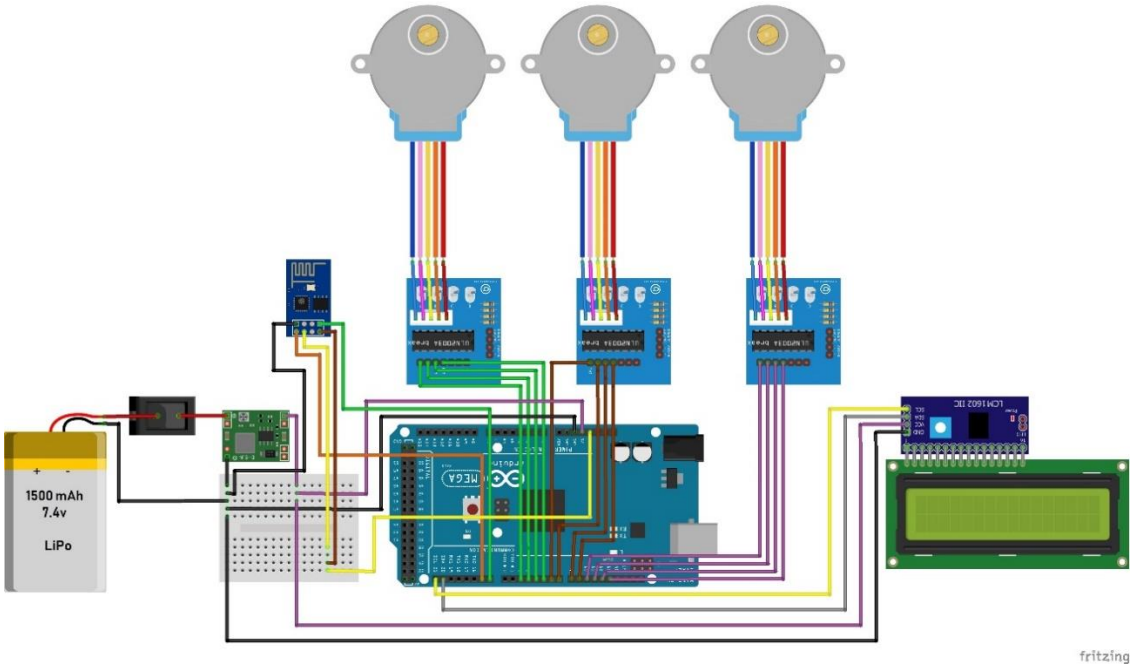


Figura 93. Conexionado completo de la electrónica del robot omnidireccional

Como se mencionó anteriormente, tras realizar el conexionado se coloca el soporte del módulo ESP-01 y del interruptor, uniéndolo a la superficie en este caso, colocando un poco de plástico con una pistola termofusible en la base. Esto se hizo de este modo para evitar ocupar espacio adicional teniendo que fijar esta pieza a la placa con tornillos.

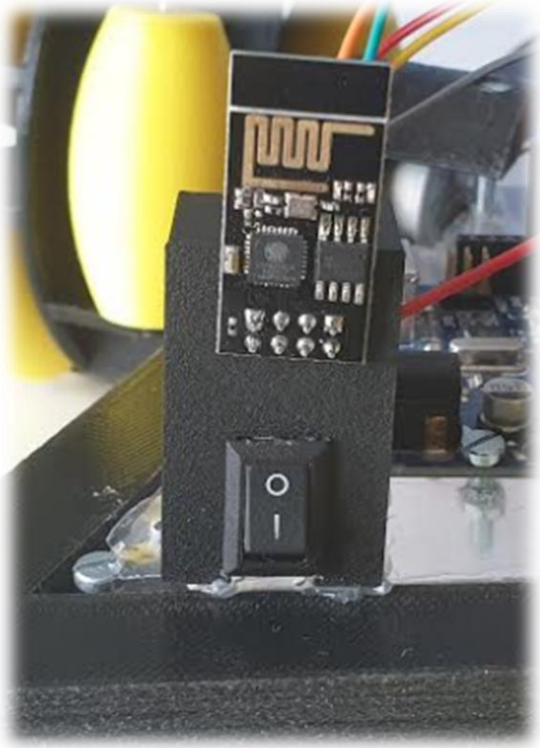


Figura 94. Colocación del módulo ESP-01 y del interruptor en su soporte

El resultado final es el que se muestra en la siguiente imagen.

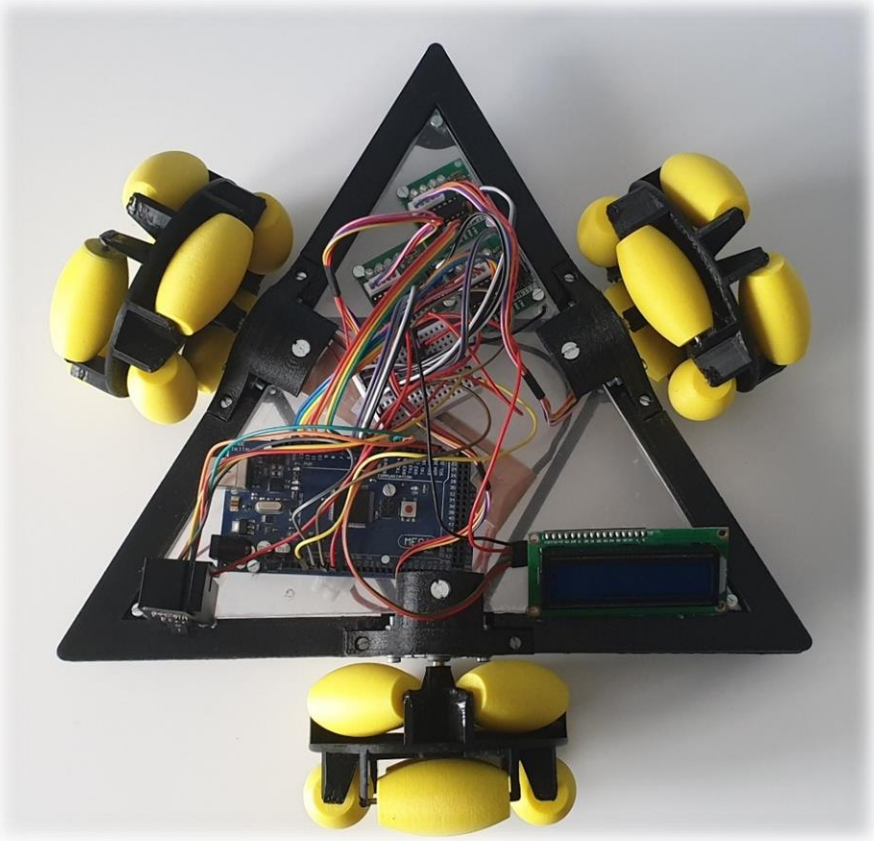


Figura 95. Robot omnidireccional fabricado

Por su parte, la batería se instala en la parte inferior del robot, y se sostiene empleando la pieza que se ha diseñado para ello.

Figura 96. Colocación de la batería en la parte inferior del robot

3.5. Control del movimiento del robot

Ante un reto como el que se ha presentado en este proyecto, en relación a la necesidad de aprovechar las virtudes que ofrece un diseño de robot que incorpora ruedas omnidireccionales, en una distribución como la que se ha descrito, se requería un sistema de control que permitiese dotar al robot de una gran maniobrabilidad.

Junto a lo anterior, este control se debía hacer mediante una comunicación Wifi, de modo que el robot fuese completamente inalámbrico.

Para lograr este objetivo, ha sido necesario pasar por distintos modelos hasta llegar a uno que cumpliese con todos los requisitos, cada uno de ellos con sus ventajas e inconvenientes.

En los siguientes apartados se expondrán las etapas por las que ha pasado el desarrollo del proyecto, hasta llegar al sistema de control que finalmente se ha implementado en el sistema.

3.5.1. Control mediante comunicación serial

El proceso de búsqueda del sistema de control más adecuado para el robot se inició con un diseño que fuese simple, pero que permitiese realizar unos primeros ensayos, en los que verificar que los modelos 3D que se habían impreso, respondían satisfactoriamente en la práctica.

De este modo, y por el momento dejando de lado la comunicación Wifi, se elaboró un código en el Arduino IDE que permitiese elegir entre una serie de movimientos predefinidos, cada uno de ellos asociado a un identificador.

En los ensayos, el Arduino estaba conectado al ordenador a través de un cable USB tipo B, el cual permitía enviar estas peticiones desde el monitor serial del software en el ordenador.

El objetivo era verificar que las ruedas omnidireccionales diseñadas, y su disposición, eran las correctas para poder reproducir los movimientos habituales de este tipo de robots. Para ello, se implementaron 9 tipos distintos de movimientos, los cuales permitían realizar movimientos de rotación pura, y de traslación pura en las direcciones perpendiculares a los ejes de las ruedas, en ambos casos en los dos sentidos de giro. También se podía detener el robot.

Identificador	Sentido de giro motor 1	Sentido de giro motor 2	Sentido de giro motor 3
0	-	-	-
1	Derecha	Derecha	Derecha
2	Izquierda	Izquierda	Izquierda
3	Derecha	Izquierda	-
4	Derecha	-	Izquierda
5	-	Derecha	Izquierda
6	Izquierda	Derecha	-
7	Izquierda	-	Derecha
8	-	Izquierda	Derecha

Tabla 18. Sentido de giro de los motores según el valor del identificador

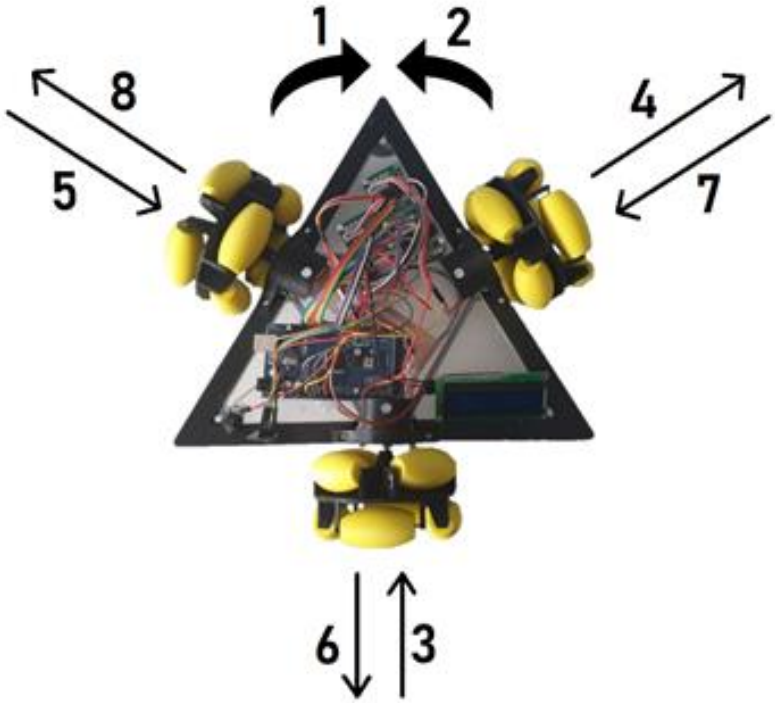


Figura 97. Discretización de los movimientos del robot

En la anterior tabla se muestra el sentido de giro que realiza cada rueda para cada identificador, de modo que se puedan describir todos los movimientos antes mencionados, y que se reflejan en la última imagen. En este modelo no se podía controlar la velocidad de giro a no ser que se realizasen cambios en el código. En concreto existe una variable llamada "valorDelay" que controla el ritmo al que se dan los pasos. Hay que recordar que, como se ha mencionado anteriormente, se trabajaría con una secuencia de 4 etapas (pasos completos). El "valorDelay" controla el tiempo que transcurre entre una etapa y la siguiente, determinando de este modo la velocidad de giro de los motores.

En el apartado que se dedicó a la explicación teórica de los motores se expuso una tabla con la secuencia de activaciones de los pines para mover un motor. En este caso, el

objetivo es que avancen 2 o 3 motores a la vez, en el mismo sentido o en sentidos opuestos. Por este motivo es necesario entrelazar los pasos de cada motor.

Por ejemplo, para realizar el movimiento asociado al identificador 3, el primer motor debe girar en un sentido, el segundo en sentido contrario, y el tercero debe permanecer parado.

Secuencia	Motor	IN1	IN2	IN3	IN4
1	M1	High	High	Low	Low
1	M2	High	Low	Low	High
1	M3	Low	Low	Low	Low
valorDelay					
2	M1	Low	High	High	Low
2	M2	Low	Low	High	High
2	M3	Low	Low	Low	Low
valorDelay					
3	M1	Low	Low	High	High
3	M2	Low	High	High	Low
3	M3	Low	Low	Low	Low
valorDelay					
4	M1	High	Low	Low	High
4	M2	High	High	Low	Low
4	M3	Low	Low	Low	Low
valorDelay					

Tabla 19. Secuencia de activaciones de los pines de control de los motores para el identificador 3

Entrando en el detalle de la implementación, se intentó elaborar un modelo simple que permitiese reproducir estos movimientos predefinidos, que estaban asociados a los identificadores. Si, como en el ejemplo planteado, se introdujese por el monitor serial del Arduino IDE un 3, este valor se almacenaría a partir de las siguientes instrucciones.

```
if (Serial.available() > 0){
      String str = Serial.readStringUntil('\n');
      identificador = str.toInt();
      Serial.println(identificador);
   }
```

Cuando el puerto serial está disponible, es decir, que se ha introducido un número por teclado, se almacena el valor en un string, que posteriormente se convierte en un valor entero, para finalmente almacenarlo en la variable “identificador”.

Este valor no se modifica hasta que se envíe un nuevo dato.

A continuación, existen varias posibilidades para indicar a los motores paso a paso el movimiento que deben realizar. Se podría crear una estructura Switch-case, en la que se saltase a diferentes funciones en función del identificador. En esa función se escribiría la secuencia completa que se presentó en la anterior tabla. Esta solución supondría que el código se extendiese considerablemente, ya que había que repetir en numerosas ocasiones las mismas instrucciones. Por este motivo, se implementó un bucle for que llamase a una misma función 4 veces. Cada una de estas llamadas se activarían los pines necesarios, para cada motor, atendiendo tanto a la etapa de la secuencia en la que se encuentra el programa, como al identificador.

El bucle for es el siguiente:

```
for(int i = 0; i < 4; i++){
    moverMotores(identificador);
    delay(valorDelay);
```

Como se puede apreciar, entre cada llamada a la función “moverMotores”, se detiene la ejecución del programa un tiempo establecido por “valorDelay”.

Por su parte, el contenido de la función es el siguiente:

```
// Función para el movimiento de los motores
void moverMotores(int identificador){


  // Motor 1 (M1)


  // Los identificadores que indican movimiento positivo de M1
  if(identificador == 1 || identificador == 3|| identificador == 4){
    Secuencia_Actual_A +=1;
  }
  // Los identificadores que indican movimiento negativo de M1
  if(identificador == 2 || identificador == 6 || identificador == 7){
    Secuencia_Actual_A -=1;
  }


  if (Secuencia_Actual_A == 4){
     Secuencia_Actual_A = 0;
  }
```

```
if (Secuencia_Actual_A == -1){
     Secuencia_Actual_A = 3;
  }
  switch(Secuencia_Actual_A) {
  case 0:
      digitalWrite(M11, LOW);
      digitalWrite(M12, LOW);
      digitalWrite(M13, HIGH);
      digitalWrite(M14, HIGH);
      break;
  case 1:
      digitalWrite(M11, LOW);
      digitalWrite(M12, HIGH);
      digitalWrite(M13, HIGH);
      digitalWrite(M14, LOW);
      break;
  case 2:
      digitalWrite(M11, HIGH);
      digitalWrite(M12, HIGH);
      digitalWrite(M13, LOW);
      digitalWrite(M14, LOW);
      break;
  case 3:
      digitalWrite(M11, HIGH);
      digitalWrite(M12, LOW);
      digitalWrite(M13, LOW);
      digitalWrite(M14, HIGH);
      break;
  default:
      digitalWrite(M11, LOW);
      digitalWrite(M12, LOW);
      digitalWrite(M13, LOW);
      digitalWrite(M14, LOW);
      break;
}
```

```
// Motor 2 (M2)


  // Los identificadores que indican movimiento positivo de M2
  if(identificador == 1 || identificador == 5|| identificador ==
6){
    Secuencia_Actual_B +=1;
  }
  // Los identificadores que indican movimiento negativo de M2
  if(identificador == 2 || identificador == 3 || identificador ==
8){
    Secuencia_Actual_B -=1;
  }


  if (Secuencia_Actual_B == 4){
     Secuencia_Actual_B = 0;
  }
  if (Secuencia_Actual_B == -1){
     Secuencia_Actual_B = 3;
  }


  switch(Secuencia_Actual_B) {
  case 0:
      digitalWrite(M21, LOW);
      digitalWrite(M22, LOW);
      digitalWrite(M23, HIGH);
      digitalWrite(M24, HIGH);
      break;
  case 1:
      digitalWrite(M21, LOW);
      digitalWrite(M22, HIGH);
      digitalWrite(M23, HIGH);
      digitalWrite(M24, LOW);
      break;
  case 2:
      digitalWrite(M21, HIGH);
      digitalWrite(M22, HIGH);
```

```
digitalWrite(M23, LOW);
      digitalWrite(M24, LOW);
      break;
  case 3:
      digitalWrite(M21, HIGH);
      digitalWrite(M22, LOW);
      digitalWrite(M23, LOW);
      digitalWrite(M24, HIGH);
      break;
  default:
      digitalWrite(M21, LOW);
      digitalWrite(M22, LOW);
      digitalWrite(M23, LOW);
      digitalWrite(M24, LOW);
      break;

  }
  // Motor 3 (M3)

  // Los identificadores que indican movimiento positivo de M3
  if(identificador == 1 || identificador == 7|| identificador == 8){
    Secuencia_Actual_C +=1;
  }
  // Los identificadores que indican movimiento negativo de M3
  if(identificador == 2 || identificador == 4 || identificador == 5){
    Secuencia_Actual_C -=1;
  }

  if (Secuencia_Actual_C == 4){
     Secuencia_Actual_C = 0;
  }
  if (Secuencia_Actual_C == -1){
     Secuencia_Actual_C = 3;
  }
```

```
  switch(Secuencia_Actual_C) {
  case 0:
      digitalWrite(M31, LOW);
      digitalWrite(M32, LOW);
      digitalWrite(M33, HIGH);
      digitalWrite(M34, HIGH);
      break;
  case 1:
      digitalWrite(M31, LOW);
      digitalWrite(M32, HIGH);
      digitalWrite(M33, HIGH);
      digitalWrite(M34, LOW);
      break;
  case 2:
      digitalWrite(M31, HIGH);
      digitalWrite(M32, HIGH);
      digitalWrite(M33, LOW);
      digitalWrite(M34, LOW);
      break;
  case 3:
      digitalWrite(M31, HIGH);
      digitalWrite(M32, LOW);
      digitalWrite(M33, LOW);
      digitalWrite(M34, HIGH);
      break;
  default:
      digitalWrite(M31, LOW);
      digitalWrite(M32, LOW);
      digitalWrite(M33, LOW);
      digitalWrite(M34, LOW);
      break;

  }
}
```

En esta función, se incluye la secuencia de activaciones de pines de cada motor, la cual se recorrerá en un sentido o en el contrario según el valor del identificador. Esto se ha conseguido, en el caso del motor A, incrementando, o decrementando, el valor de Secuencia_Actual_A, lo cual afecta al valor que llega a la estructura switch-case. Se reitera este comportamiento para los otros dos motores.

De este modo se habrán activado y desactivado los pines del controlador de motores adecuados a la primera etapa. Se termina la ejecución de la función, y en el bucle for, tras detener la ejecución el tiempo que se hubiese establecido previamente, se realiza una segunda llamada a la función, en la que se activarán los pines de la segunda etapa.

Este procedimiento se repite de forma cíclica, de forma que el identificador no se modifica hasta que así lo requiera el usuario. En la imagen siguiente, se puede ver un ejemplo en el que se ha introducido por el monitor serial los identificadores 3, 1 y finalmente 0, deteniendo el movimiento de los 3 motores en este último caso.

```
// Código para el control de los motores mediante identificadores enviados a través del monitor serial
// Autor: Alberto Díaz Rodríguez
// Fecha: 31/10/18
// Universidad de La Laguna

// El código de números será el siguiente:
// -IDENTIFICADOR --------MOTORES ACTIVADOS -------- SENTIDO DE GIRO-
//                      M1      M2      M3          M1      M2      M3
//        0             NO      NO      NO          -       -       -
//        1             SI      SI      SI          D       D       D
//        2             SI      SI      SI          I       I       I
//        3             SI      SI      NO          D       I       -
//        4             SI      NO      SI          D       -       I
//        5             NO      SI      SI          -       D       I
//        6             SI      SI      NO          I       D       -
//        7             SI      NO      SI          I       -       D
//        8             NO      SI      SI          -       I       D
//-------------------------------------------------------------------

// Nombres de los pines
const int M11 = 2;
const int M12 = 3;
const int M13 = 4;
const int M14 = 5;

const int M21 = 6;
const int M22 = 7;
const int M23 = 8;
const int M24 = 9;
```

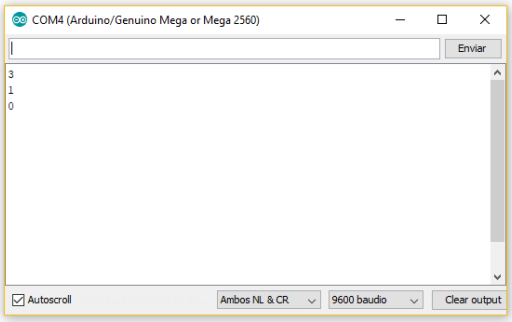


Figura 98. Envío de órdenes en el sistema de control mediante comunicación serial

Gracias a este código, se pudo ver el comportamiento de las ruedas omnidireccionales en la práctica. En este caso se pudo percibir que las ruedas patinaban un poco, lo cual afectaba de forma considerable a los movimientos que realizaba el robot. Por este motivo se tuvo que tomar la decisión de cambiar el material en el que se habían fabricado los rodillos de las ruedas. En un primer momento se habían impreso con PLA, pero tras este ensayo, se decidió utilizar en su lugar filaflex. Con este nuevo material el comportamiento de robot mejoró notoriamente. Se verificó de este modo que, el diseño de las ruedas omnidireccionales y la colocación de las mismas en la estructura del robot era el adecuado para los objetivos que se perseguían en el proyecto.

La versión completa del código que se ha introducido en este apartado se puede localizar en el apartado 6.2.1. Además, cabe mencionar que este programa se puede utilizar tanto para el Arduino UNO como para el Arduino MEGA.

3.5.2. Control mediante servidor web

Tras haber verificado que el robot diseñado y fabricado, respondía correctamente ante un control básico, se continuó el desarrollo del proyecto tratando de replicar el funcionamiento que se había logrado, pero en este caso realizando el envío de los identificadores a través de una conexión Wifi.

A nivel de hardware, se sustituyó el Arduino UNO por un Arduino WeMos D1, para contar de este modo con un chip ESP-8266, que permitía a la placa conectarse a un punto de acceso. Se buscó información acerca de las librerías que se podían utilizar al estar trabajando con esta tarjeta. De este modo se encontró la librería ESP8266WiFi.h [28], la cual permite conectar la placa a una red Wifi indicando la ssid y la contraseña de dicha red.

Con todo lo anterior listo, se escribió un código que pudiese recibir los identificadores que se mencionaron en el modelo anterior, pero en esta ocasión utilizando la conexión que se había establecido. Atendiendo a este objetivo, se implementó un servidor web, el cual consistió en crear un servidor alojado en Arduino, al cual se pudiese conectar un cliente, a través de un navegador, ya sea en un móvil o en un ordenador, el cual debía estar conectado a la misma red.

En este diseño, el Arduino es el que se encarga de iniciar la comunicación, mostrando al cliente, a través de una interfaz simple, las posibilidades que éste tenía para interactuar con el sistema. En concreto, se permitía realizar el envío de los identificadores que definían cuál de los movimientos predefinidos debía describir el robot. Sin embargo, no se llegaba a realizar el movimiento de los motores en este código ya que, como se ha mencionado anteriormente en el documento, el Arduino WeMos D1 no cuenta con el número de pines que son necesarios para este robot.

Resulta relevante mencionar que, aunque no se pudo realizar un movimiento real de los motores, si se implementó un método en el que se simulaba el número de pasos que avanzaba cada motor durante la ejecución del programa, almacenando esta información en variables, para posteriormente calcular el número de grados que había girado cada motor, y enviar esta información finalmente al cliente.

Como información adicional, hay que recordar que la placa Arduino WeMos no está incluida por defecto en el Arduino IDE, y tras instalarla, antes de cargar el programa hay que seleccionar algunos ajustes que no son necesarios en el caso de otras tarjetas. La placa que hay que seleccionar es la LOLIN (WEMOS) D1 R2 & mini, y la velocidad de subida se puede ajustar a 115200 baudios.

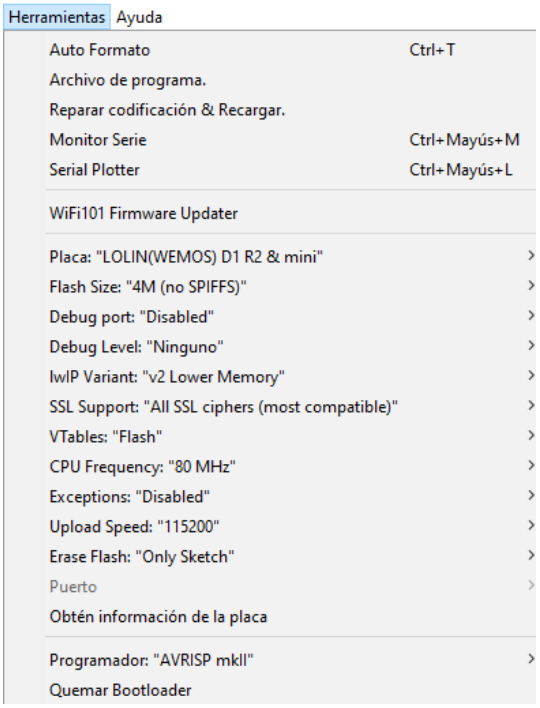


Figura 99. Configuración para cargar un programa en el Arduino WeMos desde el Arduino IDE

En cuanto a la implementación realizada, en primer lugar, se realiza la conexión a la red Wifi.

```
WiFi.mode(WIFI_STA);  // ESP8266 en modo estación
  WiFi.begin(ssid, password); // ssid y contraseña de la red
  while (WiFi.status() != WL_CONNECTED) { //Se espera hasta que se haya establecido la conexión
    delay(500);
    Serial.print(".");
  }
```

Una vez se ha conectado el Arduino a la red, se inicia el servidor.

```
server.begin();
```

Posteriormente se muestra por el monitor serial la dirección IP a la que hay que conectarse en la red.

```
// Se muestra la dirección IP a la que hay que conectarse para interactuar con el software
  Serial.print("URL : http://");
  Serial.println(WiFi.localIP());
```

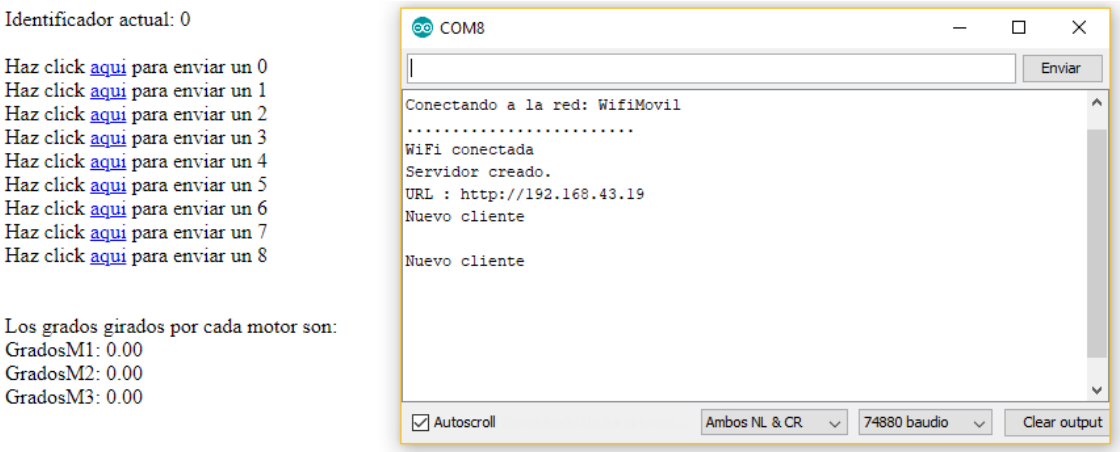


Figura 100. Interfaz del sistema de control basado en un servidor web en Arduino WeMos

Como se muestra en la anterior imagen, tras realizar la conexión, se muestra en el navegador el identificador actual, que al cargar el programa comienza con un valor de 0, y se ofrece la posibilidad de clicar en distintos lugares de la pantalla, marcados en azul. Cada uno de ellos se encarga de enviar un identificador distinto a la placa.

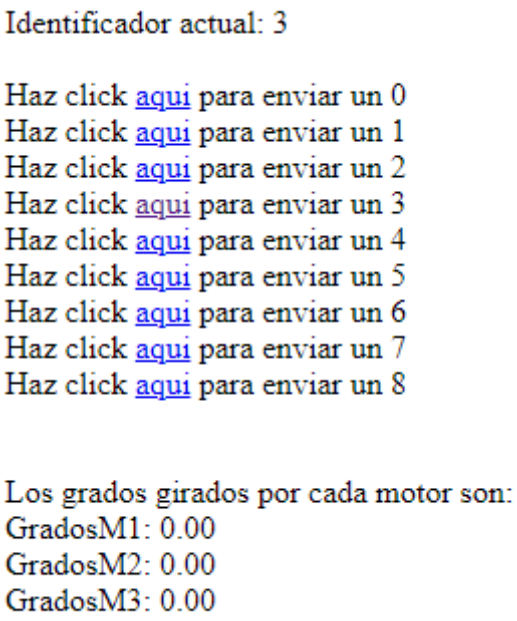


Figura 101. Envío de órdenes al robot desde un navegador

En la anterior imagen se muestra lo que ocurre cuando se hace clic para enviar un 3. Se muestra por pantalla este nuevo valor, y el programa sigue funcionando. Como se mencionó, este código simula que se están moviendo los motores, almacenando en las variables pasosM1, pasosM2 y pasosM3 el número de pasos que hubiese dado el robot en una situación real. En realidad, se están incrementando estos parámetros por cada etapa de activación de pines del motor, en vez de por cada paso real. Debido a esto, en el momento en el que se calculan los grados girados, se incluye un 4 en el denominador que corrige esto, calculando correctamente el ángulo de cada motor.

Tras dejar pasar un tiempo, si se envía desde el navegador una nueva orden, de modifica el identificador, y la interfaz actualiza los valores de giro de los motores.

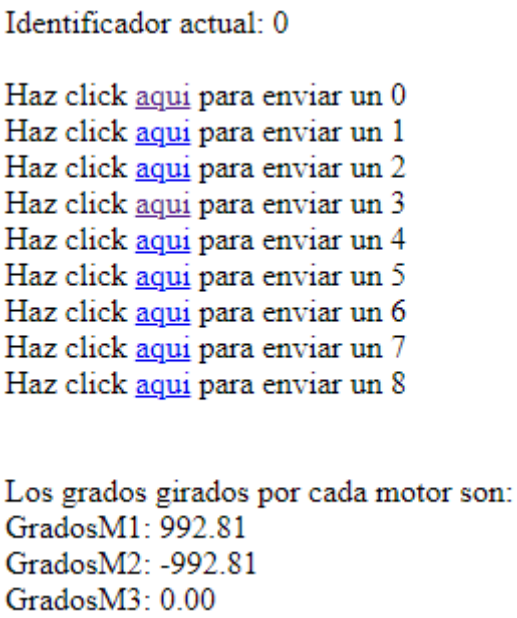


Figura 102. Visualización de los grados que han girado los motores desde un navegador

En el ejemplo mostrado en la imagen anterior, se ha vuelto a seleccionar el identificador 0. Como ha estado activado el movimiento asociado al identificador 3 durante un tiempo, el motor 1 ha estado girando en sentido positivo, y el motor 2 en sentido negativo. Por su parte, el tercer motor, ha permanecido este tiempo en reposo.

El bucle principal del código comienza con la búsqueda de un cliente activo.

```
WiFiClient client = server.available();
```

Seguidamente, y utilizando el mismo método que en el programa expuesto en el apartado anterior, se llama a la función que se encarga del movimiento de los motores. Se emplea un bucle for para que se hagan 4 llamadas seguidas, deteniendo la ejecución entre cada llamada un tiempo determinado al cargar el programa (valorDelay), atendiendo de este modo a las 4 etapas que definen una secuencia completa para girar el eje de cada motor 1 paso.

En este caso, la función a la que se ha hecho referencia, solo se encarga de incrementar o decrementar el valor de las variables pasosM1, pasosM2 y pasosM3, simulando un avance real de los motores.

```
void moverMotores(int identificador){
```

```
// Motor 1 (M1)

// Los identificadores que indican movimiento positivo de M1

if(identificador == 1 || identificador == 3|| identificador ==
4){

  pasosM1 ++;

}

// Los identificadores que indican movimiento negativo de M1

if(identificador == 2 || identificador == 6 || identificador ==
7){

  pasosM1 --;

}
```

```
// Motor 2 (M2)

// Los identificadores que indican movimiento positivo de M2

if(identificador == 1 || identificador == 5|| identificador ==
6){

  pasosM2 ++;

}

// Los identificadores que indican movimiento negativo de M2

if(identificador == 2 || identificador == 3 || identificador ==
8){

  pasosM2 --;

}
```

```
// Motor 3 (M3)

// Los identificadores que indican movimiento positivo de M3

if(identificador == 1 || identificador == 7|| identificador ==
8){

  pasosM3 ++;

}

// Los identificadores que indican movimiento negativo de M3

if(identificador == 2 || identificador == 4 || identificador ==
5){

  pasosM3 --;

}
```

En cuanto a la información que se muestra por pantalla, es necesario configurar el servidor HTTP, y enviar el texto en HTML, para que aparezca correctamente en el navegador.

```
// Configuración servidor HTTP
  client.println("HTTP/1.1 200 OK");
  client.println("Content-Type: text/html");
  client.println("");
  client.println("<!DOCTYPE HTML>");
  client.println("<html>");
```

```
// Se muestra al cliente el identificador actual
  switch(identificador) {
    case 0: client.print("Identificador actual: 0");break;
    case 1: client.print("Identificador actual: 1");break;
    case 2: client.print("Identificador actual: 2");break;
    case 3: client.print("Identificador actual: 3");break;
    case 4: client.print("Identificador actual: 4");break;
    case 5: client.print("Identificador actual: 5");break;
    case 6: client.print("Identificador actual: 6");break;
    case 7: client.print("Identificador actual: 7");break;
    case 8: client.print("Identificador actual: 8");break;
  }
// Se muestran las opciones que puede escoger el cliente para enviar órdenes
  client.println("<br><br>");
  client.println("Haz click <a href=\"/ORDEN=0\">aqui</a> para enviar un 0<br>");
  client.println("Haz click <a href=\"/ORDEN=1\">aqui</a> para enviar un 1<br>");
  client.println("Haz click <a href=\"/ORDEN=2\">aqui</a> para enviar un 2<br>");
  client.println("Haz click <a href=\"/ORDEN=3\">aqui</a> para enviar un 3<br>");
  client.println("Haz click <a href=\"/ORDEN=4\">aqui</a> para enviar un 4<br>");
  client.println("Haz click <a href=\"/ORDEN=5\">aqui</a> para enviar un 5<br>");
```

```
  client.println("Haz  click  <a  href=\"/ORDEN=6\">aqui</a>  para
enviar un 6<br>");

  client.println("Haz  click  <a  href=\"/ORDEN=7\">aqui</a>  para
enviar un 7<br>");

  client.println("Haz  click  <a  href=\"/ORDEN=8\">aqui</a>  para
enviar un 8<br>");
```

Como se debe mostrar la información referente a los grados girados por cada motor, cada vez que se actualiza la información que se muestra por pantalla, es en este tramo del código en el que se realizan los cálculos necesarios. Posteriormente se termina de enviar la información al cliente.

```
// Se muestran también los grados girados

  // La relación entre los pasos y los grados girados es: 512
pasos son 360 grados. Por tanto:

  gradosM1 = float(pasosM1 * 360 / 4 * 512);

  gradosM2 = float(pasosM2 * 360 / 4 * 512);

  gradosM3 = float(pasosM3 * 360 / 4 * 512);
```

```
client.println("<br>");

  client.println("<br>");

  client.println("Los grados girados por cada motor son:<br>");

  client.print("GradosM1: ");

  client.println(gradosM1);

  client.println("<br>");

  client.print("GradosM2: ");

  client.println(gradosM2);

  client.println("<br>");

  client.print("GradosM3: ");

  client.println(gradosM3);

  client.println("</html>");

  Serial.println("");
```

Una vez que se ha enviado la información al cliente, este tiene la posibilidad de responder, interactuando con aquellas zonas de la pantalla en la que se puede hacer clic. Con relación a esto, se debe prever esta posibilidad en el código, identificando en qué momento dicho cliente ha realizado una petición de cambio de identificador.

```
if (!client) {
    return;
  }
  // Se espera la respuesta del cliente
  while(!client.available()){
    delay(1);
  }
  // Se lee la orden enviada por el cliente
  orden = client.readStringUntil('\r');
  client.flush();
```

Por último, falta presentar la forma en la que se identifica la petición que ha solicitado el cliente. Para que esto sea posible, hay que conocer previamente el mensaje que envía el cliente para cada petición. Si por ejemplo, se ha hecho clic en el botón para cara cambiar al identificador 8, el string enviado contendrá la cadena de caracteres "/ORDEN=8". De este modo, para averiguar si fue un 8 la petición del cliente, se busca esa cadena en el texto que se ha recogido como respuesta del cliente.

```
if (orden.indexOf("/ORDEN=8") != -1) {
    identificador = 8;
  }
```

Estas últimas líneas de código se repiten para cada uno de los identificadores posibles.

El código completo de este método de control se expone en el apartado 6.2.2 de la memoria. De este modo se concluye la explicación de este primer modelo de servidor web, implementado en el Arduino WeMos D1. Sin embargo, el desarrollo de esta idea no concluyó en este punto. Tras decidir que, a pesar de que existían algunas posibilidades de seguir trabajando con esta placa, era más viable continuar el proyecto con un Arduino MEGA. A este nuevo dispositivo hubo que añadirle un módulo ESP-01 que, aunque pertenece a la familia de los chips ESP8266, como el ESP-12 del Arduino WeMos D1, para esta versión no era posible utilizar la librería con la que se ha trabajado hasta este punto del apartado. Esto se debía a que, para utilizar la librería ESP8266WiFi.h, había que programar directamente el ESP-01. Sin embargo, la implementación que se quería realizar consistía en cargar el programa principal en el Arduino MEGA y que este estableciese una comunicación serial con el módulo de comunicaciones Wifi.

A partir de los planteamientos expuestos en el párrafo anterior, se puede deducir que sería necesario realizar modificaciones al código antes expuesto, para adaptar el mismo al nuevo hardware con el que contaba el robot. Adicionalmente, el nuevo modelo

de control no pretendía ser solo una réplica del anterior, sino que se pretendió mejorar, atendiendo a distintos aspectos.

En primer lugar, y gracias a que con la nueva placa ya se contaba con pines suficientes para el proyecto, se implementó el sistema de movimiento de los motores, y se añadió el módulo LCD IIC / I2C 1602, en el que se podría visualizar la dirección IP que se debía introducir en el navegador. Otro avance importante en el desarrollo fue que, junto a la incorporación de la pantalla LCD, se instaló en el robot una batería. De este modo, el prototipo ya estaba preparado para ser inalámbrico. El resto de las modificaciones que se realizaron en este momento del proyecto serán descritas en las próximas líneas.

En este nuevo modelo se pudo comprobar experimentalmente que los cálculos de los ángulos de rotación de los motores, a los que se hizo referencia en los párrafos anteriores, eran correctos. No obstante, uno de los principales objetivos era que el robot fuese completamente inalámbrico tras aplicar todos los cambios que estaban previstos para este nuevo modelo. Esto supuso que en el código final no aparezca el cálculo de estos ángulos, ya que no se podían mostrar posteriormente por el monitor serial.

Entrando ya en el detalle de los cambios realizados, a nivel de hardware se cambió la placa, instalando un Arduino MEGA, al cual se conectaron todos los pines que eran necesarios para el manejo de los 3 controladores de motores, el nuevo módulo ESP-01 y la pantalla LCD. Previamente se había estudiado el modo de funcionamiento de estos nuevos módulos, por lo que tras conectarlos, no resultó muy difícil conseguir que funcionasen correctamente, a excepción del ESP-01, el cual, con motivo de tener que trabajar con los comandos AT, en vez de con la librería ESP8266WiFi.h, exigió que se realizasen importantes cambios en el código. Especialmente en el conjunto de instrucciones necesarias para configurar la conexión, que son las que se presentan a continuación.

```
// Se envian todos los comandos AT necesarios para configurar el
módulo ESP-01.

sendData("AT+CWMODE=3\r\n",1000); // ESP-01 en modo (3): enviar y
recibir datos.

sendData("AT+CIPMUX=1\r\n",1000); // Se configura para múltiples
conexiones, aunque en principio solo se producirá una.

sendData("AT+CIPSERVER=1,80\r\n",1000); // Se activa el servidor
en el puerto 80.

delay(1000);

sendData("AT+CWJAP=\"WifiMovil\",\"12345678\"\r\n",1000);//Se
indica el SSID y la contraseña de la red WIFI para establecer la
conexión.

delay(8000);

sendDataURL("AT+CIFSR\r\n",1000); // Se obtiene la dirección IP a
la que conectarse.
```

Como se puede apreciar, para el envío de estos comandos AT, que son necesarios para el funcionamiento del módulo, se están utilizando una serie de funciones generales de acceso a los datos del módulo Wifi, inspiradas en código de libre disposición, las cuales se pasan a describir.

```
// Función para enviar datos a la URL
String sendData(String command, const int timeout){
    String response = "";
    Serial3.print(command);
    long int time = millis();
    while( (time+timeout) > millis()){
      while(Serial3.available()){
        char c = Serial3.read();
        response+=c;
      }
    }
    return response;
}
```

En esta función se envía el comando AT requerido en cada momento. El parámetro timeout coincide con el máximo tiempo que se le asigna a la función para que realice dicho envío de información. Se almacena la respuesta, ya que esta es una información de gran utilidad para realizar una depuración del código así fuese necesario.

```
// Función para obtener la URL
String sendDataURL(String command, const int timeout){
    String response = "";
    Serial3.print(command);
    long int time = millis();
    while( (time+timeout) > millis()){
      while(Serial3.available()){
        char c = Serial3.read();
        response+=c;
      }
    }
```

```
    for(int j = 121; j < 135; j++){
      URL += response[j];
    }
    if (response[135] != 34){
      URL += response[135];
    }
    return response;
}
```

Por su parte, esta función trabaja del mismo modo que la anterior, con la única singularidad de que se utiliza para el envío del comando AT que devuelve la dirección IP en la que se ha creado el servidor web, la cual es importante conocer para posteriormente mostrarla por la pantalla LCD. Por este motivo, se almacena esta dirección IP en un string de nombre URL.

```
// Función para recibir datos de la URL
void getData(String command, const int timeout){
    String response = "";
    Serial3.print(command);
    long int time = millis();
    while( (time+timeout) > millis()){
      while(Serial3.available()){
        char c = Serial3.read();
        response+=c;
      }
    }
if (response.indexOf("/ORDEN=0") != -1) {
    identificador = 0;
  }
  if (response.indexOf("/ORDEN=1") != -1) {
   identificador = 1;
  }
  if (response.indexOf("/ORDEN=2") != -1) {
    identificador = 2;
  }
```

```
  if (response.indexOf("/ORDEN=3") != -1) {
    identificador = 3;
  }
  if (response.indexOf("/ORDEN=4") != -1) {
    identificador =  4;
  }
  if (response.indexOf("/ORDEN=5") != -1) {
    identificador = 5;
  }
  if (response.indexOf("/ORDEN=6") != -1) {
    identificador = 6;
  }
  if (response.indexOf("/ORDEN=7") != -1) {
    identificador = 7;
  }
  if (response.indexOf("/ORDEN=8") != -1) {
    identificador = 8;
  }
}
```

En este caso, la función getData está basada en el mismo funcionamiento que las anteriores, pero esta se utilizar cuando se quiere recibir información que ha enviado el cliente. En caso de que eso ocurra, se ha de obtener el identificador que ha seleccionado el usuario, lo cual, como se puede apreciar en el código expuesto, coincide con el método que se empleó en el Arduino WeMos.

En cuanto a la función que activa el movimiento de los motores, es la misma que se presentó en el método de control mediante comunicación serial. Se mantiene también el bucle for de 4 iteraciones que se encarga de llamar a dicha función.

En otro sentido de las ideas, al añadir la pantalla LCD, hubo que incorporar al código la librería [29] que permite trabajar con este componente, la cual se llama LiquidCriystal_I2C.h. Continuando con este módulo, ya en la función setup() se incluyen instrucciones relacionadas con este. En un primer momento se realiza una configuración inicial.

```
LiquidCrystal_I2C lcd(0x27,16,2);

lcd.init();

lcd.backlight();

lcd.clear();
```

Después de conocer la dirección IP del servidor web se muestra por la pantalla.

```
// Se muestra la dirección IP por la pantalla LCD para que el
usuario se conecte.

lcd.setCursor(0,0);

lcd.print("URL:");

lcd.setCursor (0,1);

lcd.print(URL);

lcd.display();
```

Por último, en la primera ejecución del bucle principal, cuando ya se ha conectado un cliente al servidor, se procede a apagar la pantalla, reduciendo el consumo de batería del robot de este modo.

```
LiquidCrystal_I2C lcd(0x27,16,2);

lcd.init();

lcd.noDisplay(); // Se apaga la pantalla una vez se ha conectado
un cliente.
```

Figura 103. Dirección IP de la conexión mostrada en la pantalla LCD

En la anterior imagen se puede ver que efectivamente, al iniciar el programa, se muestra la URL a través de la pantalla.

En cuanto al funcionamiento del programa, es bastante similar al modelo que se implementó en el Arduino WeMos, con la diferencia de que ahora las ruedas si se mueven.

Se mejoró la presencia de la web creada en el servidor, creando una interfaz basada en botones. Cada uno de ellos fue necesario ubicarlo de forma lógica en la pantalla, de modo que, partiendo de que la instrucción 0, que detiene los motores, se situó en el centro, el resto de los botones se colocó en la dirección en la que avanza el robot. Esto último se puede entender un poco mejor observando la representación que se ha expuesto en la figura 97.

Figura 104. Vista en teléfono móvil del control del robot mediante servidor web en Arduino MEGA

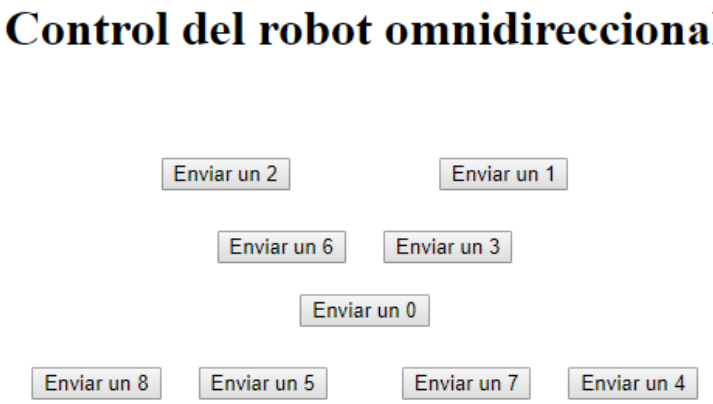


Figura 105. Vista en ordenador del control del robot mediante servidor web en Arduino MEGA

Para lograr que los botones estén colocados en el lugar deseado hay que definir los espacios en el código HTML. Por ejemplo, para los botones de envío de los identificadores 6 y 3 se incluyó el siguiente código.

```
webpage+="<h2 align=\"center\"><button onclick=location=\"/ORDEN=6\">Enviar
un 6</button>    ";

webpage+="<button onclick=location=\"/ORDEN=3\">Enviar un 3</button></h2>";
```

De este modo se concluye con los sistemas de control del movimiento basados en servidores web que se han implementado en este proyecto. Con este último modelo se pudo aprender mucho acerca del funcionamiento del módulo ESP-01, lo cual sería de gran utilidad más adelante. Por otro lado, ya se había conseguido sacar algo de provecho al sistema de ruedas omnidireccionales con el que cuenta el robot. No obstante, el objetivo desde un primer momento fue conseguir una maniobrabilidad mucho mayor. En relación a esta idea, y con el propósito de utilizar un dispositivo con el que controlar el robot de un modo mucho más preciso, se comenzó a trabajar con ROS. En concreto se pretendía desarrollar un sistema que permitiese controlar el robot mediante un mando, por supuesto, sin renunciar a que la comunicación fuese inalámbrica.

3.5.3. Control mediante Rosserial

Cuando se decidió que se debía incluir un mando para controlar los movimientos del robot, de modo que se pudiesen aprovechar, en relación a las posibilidades de movimientos, las virtudes del robot omnidireccional, se consideró que ROS era un sistema adecuado para ello.

En primer lugar, y con el objetivo de determinar si este entorno era viable para el proyecto, se debía encontrar un método para comunicar la placa Arduino con el sistema ROS. Un método muy extendido dentro de este entorno es el Rosserial, del cual ya se habló en el capítulo anterior de la memoria. Este sistema incluya las herramientas necesarias para que esta comunicación, entre los equipos, sea lo más sencilla posible, pudiendo utilizar todas las características de ROS, tales como los mensajes y servicios propios de la plataforma.

El lado negativo de este sistema, en base a las necesidades del proyecto que se ha desarrollado en este Trabajo de Fin de Grado, es que la comunicación es serial, a través de un cable. Además, introduce una gran sobrecarga en el sistema, ya que requiere del envío de muchos datos en la comunicación, y realiza un uso intensivo de la memoria del microcontrolador, lo cual no soporta una placa inalámbrica de las características mencionadas, basada en los comandos AT.

En la página web de ROS se ofrecen algunos códigos de ejemplo [30] con los que se puede utilizar el sistema Rosserial. Se probaron algunos de ellos, pero fue necesario establecer una conexión entre el ordenador y el Arduino MEGA mediante un cable USB tipo B. Debido a esta circunstancia, no se podía implementar una solución basada en el modelo por defecto de Rosserial.

La solución pasaba por utilizar una variación del sistema, en la que los mensajes se pudiesen enviar de forma inalámbrica. En relación con esto, ya en el apartado 2.8.2, se

mencionó la posibilidad de utilizar el método que se ofrece en la web de ROS para dar soporte a nuevo hardware. De este modo, se comenzó a trabajar para tratar de adaptar el módulo ESP-01 a este sistema.

La circunstancia de que, en el caso de este proyecto, en el que el módulo se debía comunicar con el Ardunino MEGA, en lugar de ser el microcontrolador principal, supuso un obstáculo para poder cumplir con el propósito que se ha mencionado.

Se buscaron soluciones expuestas en la red, pero ninguna se ajustaba al hardware con el que contaba el robot. Se trató de trabajar con un sistema denominado espros [31], que permitía realizar una comunicación Wifi entre ROS y un microcontrolador basado en el ESP8266. En concreto, en el ejemplo se utilizaba un microcontrolador NodeMCU, el cual está basado en un ESP-12E, pero que no cuenta con el número de pines que se requiere en este proyecto. Sin embargo, no se consiguió adaptar este sistema al ESP-01. La obligación de tener que utilizar los comandos AT en este módulo, como consecuencia de que no se estaba programando directamente en él, sino que se estaba concibiendo como un módulo del Arduino MEGA, estaba impidiendo que se trabajase con cualquier método relacionado con Rosserial.

Como consecuencia de todo lo que se ha expuesto en este apartado, se decidió abandonar la idea de utilizar Rosserial, pero no se descartó seguir trabajando con ROS. Se debía encontrar un método que permitiese comunicar un nodo de ROS con un microcontrolador Arduino, ya fuese a través del ESP-01, o empleando otro hardware similar que se pudiese conseguir.

3.5.4. Control mediante Ros_Arduino_Bridge. Protocolo TCP/IP

Ros_Arduino_Bridge es un paquete, que se ofrece en la página web oficial de Ros, y que contiene los medios necesarios para utilizar las herramientas propias de ROS junto a un microcontrolador Arduino. Se emplean mensajes significativamente más sencillos que en el Rosserial, logrando de este modo que la sobrecarga que se produce en el microcontrolador sea mínima a la hora de realizar comunicaciones, procesamiento de datos y uso de memoria. Definiendo de este modo una interfaz intermedia, que consiste en un nodo en ROS, que traduce los comandos propios de este sistema, a los mensajes de ROS estándar.

En cuanto a la parte que corresponde al código que se carga en el Arduino, Ros_Arduino_Bridge incluye herramientas que no son necesarias para este proyecto, tales como el control de motores, un controlador PID para determinar la velocidad de giro de estos, o un método de cálculo de la odometría del robot. Todo esto último está pensado para robots de control diferencial, por lo que no se podían utilizar junto al robot omnidireccional diseñado. De todo esto se deduce que fue necesario eliminar partes del código ofrecido, conservando aquello que guarda relación con el objetivo de control que se trataba de conseguir.

Por otro lado, el Ros_Arduino_Bridge emplea una comunicación serial cableada, de modo que se debía adaptar el mismo al uso del módulo ESP-01. Sin embargo, y al contrario que con el Rosserial, en este nuevo sistema si resultaba factible modificar el código, ya que se trabaja con comandos sencillos y cortos, de modo que, para el envío y recepción de los mensajes, se utilizasen los comandos AT con los que ya se había trabajado en la implementación del servidor web en el Arduino MEGA. En este sentido, fue necesario realizar modificaciones tanto en el código del Arduino, como en el sistema implementado en ROS.

Comenzando por el código que se iba a cargar en el Arduino, tras eliminar las instrucciones que no eran necesarias, se incorporaron las funciones de envío y recepción de mensajes a través del módulo de comunicaciones Wifi con las que se había estado trabajando en los modelos anteriores. En este caso concreto, la única diferencia entre la función de envío y la de recepción, es que esta última modifica el string que se emplea en el bucle principal para almacenar la información que ha llegado a la placa, y que se ha de interpretar.

```
// Función para enviar datos
String sendData(String command, const int timeout){


    String response = "";
    Serial3.print(command);
    long int time = millis();
    while( (time+timeout) > millis()){
      while(Serial3.available()){
        char c = Serial3.read();
        response+=c;
      }
    }
    return response;
}
```

```
// Función para recibir datos
void getData( const int timeout){


    Stringchr = "";
    long int time = millis();
    while( (time+timeout) > millis()){
      while(Serial3.available()){
        char c = Serial3.read();
        Stringchr +=c;
      }
    }
}
```

Antes de continuar, conviene mencionar que, dado que se decidió emplear inicialmente el protocolo TCP/IP para las comunicaciones, el código que hace referencia a la configuración y establecimiento de la conexión al punto de acceso es muy similar al que se presentó para el servidor web. De momento solo era necesario poder recibir datos ya que aún no se había implementado el sistema de cálculo de la odometría.

```
Serial3.begin(115200);
sendData("AT+CWMODE=3\r\n",1000,DEBUG); // Modo de
funcionamiento del módulo.
sendData("AT+CIPMUX=1\r\n",1000,DEBUG); // Múltiples conexiones
sendData("AT+CIPSERVER=1,4445\r\n",1000,DEBUG); // Se activa el
servidor en puerto 4445
delay(500);
sendData("AT+CWJAP=\"WifiMovil\",\"12345678\"\r\n",1000,DEBUG);
//Red wifi, usuario y clave
delay(5000);
sendData("AT+CIFSR\r\n",1000,DEBUG); // Se obtiene la dirección
IP
```

El siguiente paso consistió en sustituir la recepción de mensajes serial por un sistema en el que se comprobase si había llegado un mensaje al ESP-01, adaptando este sistema al conjunto de instrucciones con el que contaba el Ros_Arduino_Bridge para interpretar la información recibida.

```
if (Serial3.available()) {
    Serial.println("Recibiendo mensaje");
    Serial3.print("AT+CIPRECVMODE=1\r\n");
    getData(500);

    for(int j = 0; j < Stringchr.length(); j++){
    chr = Stringchr[j];
```

Como se puede interpretar en este último fragmento de código, tras recibir el mensaje, este se guarda completo en un string de nombre Stringchr. Sin embargo, para adaptar estos cambios al sistema inicial, se debe ir reconociendo cada uno de los caractéres que se ha recibido de manera individual.

Con el objetivo de corroborar que, efectivamente la recepción de los mensajes era correcta, se utilizó una aplicación externa, PuTTY. Este programa permite abrir una ventana de comandos con la que realizar una conexión a un dispositivo, en este caso el Arduino. Se ofrecen distintos tipos de conexión, tales como Raw, Rlogin, SSH, Serial o Telnet, siendo este último el que se empleó. Para configurar el software, era necesario indicar la dirección IP y el puerto en el que se realizaría la conexión, junto al tipo de conexión deseado, pudiendo escoger entre los antes mencionados. Resulta relevante mencionar que el ordenador en el que se utilice la aplicación debe estar conectado a la misma red que el microcontrolador que va a recibir el mensaje.

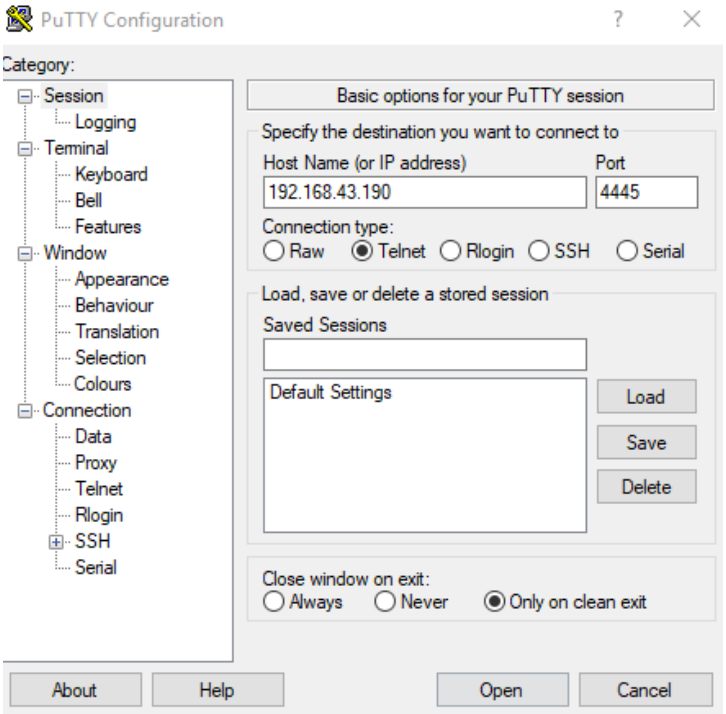


Figura 106. Ventana de configuración de PuTTY

Al presionar en el botón Open se abre la ventana en la que se puede enviar la petición. Por ejemplo, para realizar una lectura analógica del pin 3 del Arduino MEGA, se introduce “a 3”.

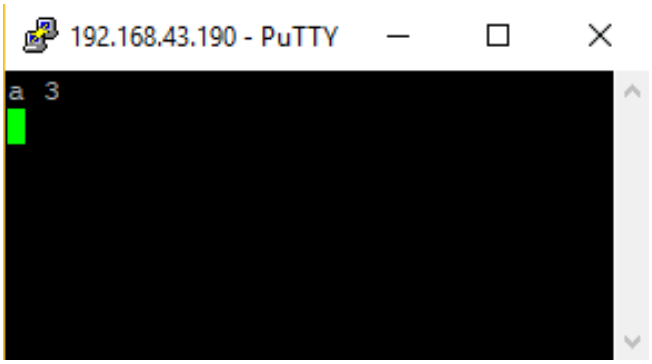


Figura 107. Terminal para el envío de comandos en PuTTY

El programa interpretaba el mensaje correctamente, de modo que se ejecutaban las instrucciones correspondientes a una lectura analógica, que son las que se muestran a continuación.

```
case ANALOG_READ:

    Serial.println(analogRead(arg1));

    respuesta = String(analogRead(arg1));

    sendData("AT+CIPSEND=" + String(0) + "," + respuesta.length()
+ "\r\n", 500, true);

    sendData(respuesta, 1000, true);

    break;
```

Se mostraba correctamente el valor por el monitor Serial del Arduino IDE. Este método se podía utilizar, de forma análoga, para el resto de los comandos que integraba el Ros_Arduino_Bridge en aquel momento. Como se puede observar, también se envía el resultado de la lectura a través de la conexión Wifi, recibiéndose en la ventana de comandos de PuTTY.

Por otro lado, y en este caso atendiendo a la parte implementada en ROS, las principales modificaciones consistieron, del mismo modo que en lo referente al Arduino, en adaptar los códigos a una comunicación que debía ser inalámbrica, y en la que se pudiese enviar un mensaje con la velocidad deseada de los 3 motores paso a paso.

En los siguientes párrafos se presentarán los cambios que se han realizado al modelo desde el que se partió del Ros_Arduino_Bridge para adaptarlo al modelo que se ha preparado en el Arduino MEGA.

En relación con el archivo de configuración de parámetros, es decir, el archivo my_arduino_params.yaml, hubo que añadir la IP y el puerto en el que se iba a establecer la comunicación Wifi, sustituyendo de este modo al puerto USB y la velocidad de transmisión de la información en la comunicación serial.

```
IP: 192.168.43.190

socket: 4445
```

Por otro lado, y dado que no se van a emplear los parámetros que prevé el software para el control de un robot de movimiento diferencial, se comentan las líneas referentes a los parámetros del PID, y hay que prestar especial atención a comentar la línea en la que se llama a un bucle de comprobación de sensores en el que se activa el pin 13, correspondiente al indicador LED, de la placa Arduino. Se comenta por tanto la siguiente línea.

```
#arduino_led: {pin: 13, type: Digital, rate: 5, direction: output}
```

El que se encienda y apague el LED de la placa de forma cíclica es útil al empezar a trabajar con este sistema, ya que de ese modo se puede comprobar que la conexión entre ROS y el microcontrolador se ha establecido correctamente, sin embargo, para la implementación final de robot, implicaría tener que estar recibiendo un mensaje para el encendido y el apagado de dicho LED, de forma continuada, lo cual afectaría a los ciclos de actualización de la velocidad de motores que se han implementado.

En cuanto a los cambios respecto al Ros_Arduino_Bridge ofrecido en la página web oficial de ROS, principalmente hubo que sustituir los envíos y establecimientos de comunicaciones seriales, por un sistema en el que la información se enviase estableciendo una conexión con un punto de acceso Wifi.

En relación con el nodo que se inicia al utilizar Ros_Arduino_Bridge, cuyo archivo de código tiene el nombre de Arduino_node.py, se deben realizar los mismos cambios que en el archivo de parámetros de configuración. De este modo se añaden las líneas siguientes, en las que se indica la dirección IP y el puerto.

```
self.IP = rospy.get_param("~IP", "192.168.43.190")

self.socket = int(rospy.get_param("~socket", 4445))
```

En cuanto al controlador de la placa Arduino, en concreto el archivo Arduino_driver.py, los cambios son similares a los casos anteriores.

En la definición del método __init__, que se encarga de inicializar los atributos del objeto Arduino en este caso, se deben incorporar los nuevos parámetros.

```
def __init__(self, IP="192.168.43.190", socket=4445, timeout=0.5,
motors_reversed=False):
```

Junto a esto, dentro del propio método se deben inicializar estos parámetros.

```
self.socket = socket
self.IP = IP
```

Además, y en este caso atendiendo al método que establece la conexión, hubo que adaptarlo al modo de creación de un socket TCP.

```
print "Connecting to Arduino on port", self.IP, " ", self.socket,
"..."
self.port = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
self.port.connect((self.IP, self.socket))
```

Por otro lado, y continuando con el mismo fichero, se definen una serie de métodos con los que se determina qué acciones realizar según la orden que se desea ejecutar. Por ejemplo, en el caso de querer realizar una lectura analógica de un pin de la placa, se debe emplear el siguiente método.

```
def analog_read(self, pin):
    return self.execute('a %d' %pin)
```

Como se puede apreciar, en este fragmento se realiza una llamada a otro método, "execute", en el cual hubo que adaptar las instrucciones a la sintaxis propia de los sockets TCP, lo cual se muestra en el siguiente fragmento de código al emplear los términos "send" y "recv".

```
def execute(self, cmd):
        ''' Thread safe execution of "cmd" on the Arduino returning
a single integer value.
        '''
        self.mutex.acquire()

        try:
          self.port.flushInput()
        except:
          pass

        ntries = 1
        attempts = 0
```

```
        try:
            self.port.send(cmd + '\r')
            value = self.port.recv(15)
            print "\nRecibido: " + value + "\n"
            while attempts < ntries and (value == '' or value ==
'Invalid Command' or value == None):
                try:

                    self.port.flushInput()
                    self.port.send(cmd + '\r')
                    value = self.recv(self.timeout)
                    print "Recibido 2: " + value
                except:
                    print "Exception executing command: " + cmd
                attempts += 1
        except:
            self.mutex.release()
            print "Exception executing command: " + cmd
            value = None
        self.mutex.release()
        return int(value)
```

Este tipo de adaptaciones hubo que realizarlas para cada uno de los métodos que se deseaban utilizar con el robot omnidireccional. Otro aspecto al que es importante atender en esta parte del código es que, al estarse utilizando el protocolo TCP/IP, es necesario recibir una confirmación de que el mensaje se ha recibido, lo cual generaría una sobrecarga que supondría un problema más adelante. Al final del archivo arduino_driver.py, se encuentra un código con el que se puede comprobar si el funcionamiento de la comunicación entre Ros_Arduino_Bridge y el microcontrolador es el correcto.

```
""" Test basico de funcionamiento """
if __name__ == "__main__":
    if os.name == "posix":
        IP_name = "192.168.43.190"
myArduino = Arduino(IP=IP_name, socket=4445, timeout=0.5)
myArduino.connect()
```

```
    print "Ejecucion detenida 1 segundo"
    time.sleep(1)

    print "Leyendo el puerto analogico 0", myArduino.analog_read(0)
    time.sleep(0.6)
    print "Leyendo el puerto analogico 1", myArduino.analog_read(1)

    print "La connexion ha sido correcta",

    myArduino.close()

    print "Comunicación cerrada"
```

Tras adaptar este fragmento de código a la conexión inalámbrica, se diseñó una prueba que consistía en realizar una lectura analógica en el pin 0 de la placa Arduino MEGA, y tras esperar un tiempo, enviar una petición para recibir el valor de otro pin, en este caso el 1. De la forma en la que se muestra el código, este funcionaba perfectamente. Sin embargo, si el tiempo de espera entre los dos envíos de las peticiones de lectura se reducía por debajo de 0.6 segundos, la segunda orden no se atendía correctamente.

Esto se debía a que el proceso completo, en el que se envía el mensaje, se recibe en el Arduino, se procesa la información y se envía la confirmación de la recepción, requería de un tiempo mayor al que se estaba dejando entre las dos peticiones que se realizaban en el código de prueba. Esto suponía un importante obstáculo de cara a realizar el envío de las velocidades de los motores a una frecuencia adecuada.

De este modo, se tomaron una serie de decisiones con las que se consiguió solventar este problema. Una de ellas fue la de sustituir el protocolo TCP/IP por el UDP, ya que de este modo no era necesario que el Arduino confirmase la recepción del mensaje, reduciendo de este modo el número de operaciones que se debían llevar a cabo para enviar un mensaje desde ROS hasta el microcontrolador.

3.5.5. Control mediante Ros_Arduino_Bridge. Protocolo UDP

En este apartado, atendiendo a los últimos planteamientos que se han presentado en este documento, se describirá la adaptación al protocolo UDP que se implementó en el Ros_Arduino_Bridge, la cual, junto a otras medidas que se presentarán en los próximos párrafos, ayudaron a reducir el tiempo que se debía conceder entre cada envío de mensajes desde ROS.

Las adaptaciones que se habían realizado para el caso anterior, en relación con el establecimiento de la conexión con un punto de acceso Wifi, seguían siendo válidas para

este nuevo protocolo en su mayoría. Por este motivo, se mencionarán en este apartado las correcciones que fueron necesarias para este cambio de protocolo.

Atendiendo al código de Arduino, se realizaron las modificaciones oportunas en las instrucciones de configuración de la conexión inalámbrica, basadas en el envío de comandos AT. En este caso, y de cara a un posterior envío de la pose del robot, se incorporó la posibilidad de realizar envíos a un socket UDP implementado en el ordenador.

```
Serial3.begin(115200);

sendData("AT+CWMODE=3\r\n",1000); // Configuración punto de acceso

sendData("AT+CWJAP=\"WifiMovil\",\"12345678\"\r\n",1000);//red
wifi, usuario y clave

delay(5000);

sendData("AT+CIPMUX=1\r\n",1000); // Múltiples conexiones

sendData("AT+CIFSR\r\n",1000); // Se obtienen las direcciones IP


sendData("AT+CIPSTART=0,\"UDP\",\"0.0.0.0\",4445,4445,2\r\n",1000
); // Para recibir la información procedente de ROS en el puerto
4445


sendData("AT+CIPSTART=1,\"UDP\",\"192.168.43.189\",4446,4446,2\r\
n",1000); // IP del ordenador destino. Se envía al puerto 4446

sendData("AT+CIPDINFO=0\r\n",1000);

delay(500);
```

De forma paralela a lo anterior, y con el objetivo de solo fuese necesario el envío de un único mensaje para indicar las velocidades de los tres motores, se debía poder recibir en el Arduino un mensaje con estos tres parámetros. Esta circunstancia dio lugar a la necesidad de incorporar en el código, un comando para la recepción de este tipo de mensajes. Junto a esto, el modelo que se había tomado como base para el desarrollo de este sistema de control, solo permitía recibir 2 parámetros por cada mensaje recibido. El que las velocidades de los motores llegasen juntas, en un único mensaje, era un requisito con el que debía contar el código, ya que de este modo se reduce la frecuencia de envío de mensajes que es necesaria para logar una navegación fluida del robot. De este modo se reduce la magnitud del problema presentado al final del apartado anterior, por el que se determinó que era necesario un tiempo de espera entre el envío de los mensajes desde ROS.

En primer lugar, se definió el nuevo comando en el fichero commands.h.

```
#define MOTORS_SPEED   'x'
```

Se asignaría de este modo el carácter x al envío de la velocidad de cada uno de los tres motores paso a paso. La siguiente modificación que se realizó fue crear una nueva variable, arg3, que almacenaría la velocidad del tercer motor cuando se recibiese un mensaje del tipo que se acababa de implementar. Esto supuso que hubiese que realizar cambios en aquellas funciones en las que intervendría este parámetro. Destaca entre esto último, las modificaciones que se llevaron a cabo en el conjunto de instrucciones que, partiendo del string que se recibe en el Arduino, se determina el valor de cada uno de los argumentos necesarios para la función a la que se ha llamado. Al haber incorporado este nuevo comando, para la velocidad de los motores, se debía almacenar correctamente el valor correspondiente al tercer motor, sin que esto afectase a los otros dos argumentos, los cuales intervienen en las llamadas a otras funciones distintas. Por ejemplo, en la petición de la lectura analógica en un pin solo interviene la variable arg1. El código resultante es el que se muestra en las siguientes líneas.

```
if (chr == 13) {
      if (arg == 1) argv1[index] = NULL;
      else if (arg == 2) argv2[index] = NULL;
        else if (arg == 3) argv3[index] = NULL;
        Serial.print("Command ");
        Serial.println(cmd);
        runCommand();
        resetCommand();
    }
// Se utilizan espacios para delimitar las partes del mensaje
    else if (chr == ' ') {
      // Se cambia al segundo argumento
      if (arg == 0) arg = 1;
      else if (arg == 1)  {
        argv1[index] = NULL;
        arg = 2;
        index = 0;
      } Se cambia al tercer argumento
      else if (arg == 2)  {
        argv2[index] = NULL;
        arg = 3;
        index = 0;
```

```
          }
      continue;


    }
    else {
      if (arg == 0) {
        // El primer argumento es un comando de una letra
        cmd = chr;
      }
      else if (arg == 1) {
        argv1[index] = chr;
        index++;
      }
      else if (arg == 2) {
        argv2[index] = chr;
        index++;
      }
      else if (arg == 3) {
        argv3[index] = chr;
        index++;
      }
    }
```

Con este nuevo protocolo ya no era necesario que el Arduino enviase un mensaje con el resultado de la petición proveniente de ROS. De este modo, el código que se ejecuta cuando la orden requerida es la de realizar una lectura analógica de un pin del microcontrolador, es el siguiente.

```
case ANALOG_READ:
      Serial.println(analogRead(arg1));
      Serial.println("El pin que se esta leyendo es el: ");
      Serial.print(arg1);
    break;
```

Para probar el nuevo sistema era necesario adaptar la parte implementada en ROS al nuevo protocolo. En este caso, y como ya se ha mencionado, gran parte del trabajo que se expuso en el apartado anterior era válido para el protocolo UDP. Los cambios consistieron en sustituir las instrucciones propias de los sockets TCP por las de los sockets UDP. Este es el caso de la instrucción “send”, la cual hubo que sustituir por ”sendto”. En el caso de las líneas destinadas a la recepción de datos, estas ya no eran necesarias ya que el Arduino ya no iba a responder a los mensajes que recibiese. Particularizando esta idea en los distintos ficheros que componen el paquete Ros_Arduino_Bridge, en los que influye el medio de comunicación que se emplee, el archivo de configuraciones no es necesario modificarlo respecto a lo que se planteó para el protocolo TCP/IP. Ocurre lo mismo para el arduino_node.py. En el caso del fichero arduino_driver.py, además de sustituir las instrucciones que antes se mencionaron ante el cambio de protocolo, también se ve modificado el método en el que se establece la conexión.

```
print "Connecting to Arduino on port", self.IP, " ", self.socket,
"..."

self.port = socket.socket(socket.AF_INET,  socket.SOCK_DGRAM)
```

En este mismo archivo, y en este caso atendiendo a la necesidad de enviar en un único mensaje la velocidad de cada uno de los motores, se definió un nuevo método.

```
def motors_speed(self, vel1, vel2, vel3):

        return self.execute_ack('x %d %d %d' %(vel1, vel2,
vel3))
```

En el último fragmento de código se pueden apreciar dos detalles importantes. La cadena de texto que se está creando comienza por el carácter x, que es la letra que se asignó en el código del Arduino al comando de recepción de las velocidades de los motores. El otro aspecto importante es que se utiliza el método execute_ack, que es el siguiente.

```
def execute_ack(self, cmd):

        ''' Thread safe execution of "cmd" on the Arduino returning
True if response is ACK.

        '''

        self.mutex.acquire()

        try:

            self.port.flushInput()

        except:

            pass

        self.port.sendto(cmd + '\r',(self.IP,self.socket))

        self.mutex.release()

        return 1
```

Como se puede comprobar, este método es el que realiza el envío del mensaje al incluir la instrucción sendto, cuyos parámetros son el mensaje y la dirección y puerto al que se debe enviar la información.

Junto a esto, es necesario definir en el arduino_node.py, el código del tópico que se se va a utilizar para el envío de la información de la velocidad de los motores.

Hay que definir también en este código el tópico que se va a utilizar para el envío de la información de las velocidades de los motores.

```
# Topico de la velocidad de los 3 motores
    def callback(self, data):
      self.controller.motors_speed(data.x,data.y,data.z)
        rospy.loginfo("El mensaje recibido es: [%f,%f,%f]",
data.x,data.y,data.z)
        return
```

Como se puede apreciar, se muestra por la ventana de comandos, en tiempo de ejecución, los valores que se están transmitiendo en cada momento. De este modo se puede comprobar durante el funcionamiento del robot que los valores provenientes de los nodos previos son los correctos.

Conseguir que todos los cambios que se han presentado en este apartado funcionasen en conjunto requirió de la dedicación de una cantidad importante de tiempo, en el que se fueron realizando las pruebas necesarias hasta lograr un correcto funcionamiento del sistema de comunicación.

En otro sentido de las ideas, se implementó un nuevo sistema de control del movimiento de los motores paso a paso, que respondía a la necesidad de atender a las velocidades de los motores a las que se ha hecho referencia, dejando atrás el método de control basado en movimientos predefinidos. Este nuevo sistema se basaría en interrupciones software, que detendrían la ejecución del ciclo principal, atendiendo de este modo al movimiento de los motores paso a paso.

En primer lugar, se deben incluir aquellas funciones destinadas a la configuración de estas interrupciones, las cuales se basan en código de libre disposición.

```
// Función para deshabilitar el Timer3
void disableTimer3(){
  TCCR3A = 0;
  TCCR3B = 0;
}
```

```
// Configuración del Timer3
void setupTimer3_25mS(){
  // Se deshabilitan las interrupciones
  cli();

  disableTimer3();

  // Se ajusta el parámetro para el tiempo de cuenta deseado
  OCR3A = 575;

  // Se activa el modo CTC. WGM32 establece los valores de los
bits CS10 y CS12 para un preescalado del Timer3 de 1024
  TCCR3B = bit (WGM32) | bit (WGM32) | bit (CS30);

  // Se activa el temporizador
  TIMSK3 |= (1 << OCIE3A);

  // Se habilitan las interrupciones
  sei();
}
```

Un aspecto importante que considerar de esta configuración es el parámetro OCR3A, que es un registro de comparación de salida, propio de los microcontroladores ATmega2560, el cual incluye el Arduino MEGA, y que permite establecer el valor deseado para el contador del temporizador Timer3. De este modo, permite establecer cada cuanto tiempo se produce una interrupción software.

A este registro se le ha asignado un valor de 575, que se ha determinado, de forma experimental, que es el más adecuado para el comportamiento que se quiere lograr, el cual se describirá tras presentar el código en las siguientes líneas.

Primeramente, la función a la que accede el código cada vez que se produce una interrupción es la siguiente.

```
ISR(TIMER3_COMPA_vect)
```

En ella se atiende a cada una de las secuencias que se deben producir en cada uno de los motores, para que estos se muevan acorde a la velocidad que se les ha asignado.

```
// Motor A
Interrupciones_A = Interrupciones_A -1;


if (Interrupciones_A <= abs(numero_cambio_secuencia_A)) {
  Interrupciones_A = 255;


 // Toca la siguiente secuencia
  if(numero_cambio_secuencia_A > 0){
    Secuencia_Actual_A +=1;
    avanceMotores[0] +=1;
  }
  if(numero_cambio_secuencia_A < 0){
    Secuencia_Actual_A -=1;
    avanceMotores[1] +=1;
  }


  if (Secuencia_Actual_A == 4){
     Secuencia_Actual_A = 0;
  }
  if (Secuencia_Actual_A == -1){
     Secuencia_Actual_A = 3;
  }
```

En este primer fragmento, se decide si se debe producir un cambio de secuencia en el motor A. Para ello, se compara el número de interrupciones que se ha producido , con el valor de la velocidad que se ha recibido para el motor A. De un modo más concreto, la variable interrupciones_A comienza con un valor de 255, decrementándose este valor en una unidad por cada interrupción que se produce. Por otro lado, en numero_cambio_secuencia_A se almacena el valor de la velocidad que se ha asignado al motor A en la última recepción de un mensaje proveniente de ROS. Esta velocidad, en un principio, podía tomar cualquier valor entero entre -255 y 255, pero para lograr un mejor comportamiento del robot se redujo este rango. Precisando un poco más en esta idea, se estableció un nuevo rango que comprendía los enteros desde el -210 hasta el 210.

En los siguientes párrafos se tratará el valor de la velocidad de los motores en valor absoluto, de modo que las explicaciones sean más claras. La razón de haber tomado esta decisión de reducir el rango de valores de velocidades es que se quiso aumentar la frecuencia de las interrupciones software, para que la diferencia entre las velocidades que

suponen cada uno de los valores del rango, no fuesen tan distintas. Sin embargo, esto provocaba que para velocidades próximas a 255 los motores perdiesen paso y por ello no giraban. A pesar de esto, el comportamiento experimental mostraba un mejor resultado.

Si por ejemplo la velocidad del motor, enviada desde ROS, fuese de 252, se tendrían que producir 4 interrupciones para que la rueda girase, lo cual sería un movimiento 4 veces más lento que el que se produciría con una asignación de 255. Extrapolando esta idea al resto de los valores del rango, sólo se conseguía una velocidad suficiente en los valores más próximos a 255, siendo el ritmo de giro demasiado lento para valores alejados a este.

Esta es la razón por la que en el modelo que se ha implementado, solo se pueden seleccionar velocidades dentro del rango de enteros entre -210 y 210. El punto clave de esta idea radica en que, al haber aumentado la frecuencia de las interrupciones, la diferencia que se puede apreciar entre una velocidad de 209 y de 210, es mucho menor que la que se producía en un primer momento entre 254 y 255. Por tanto, con este cambio, se ha visto reducido el rango de valores que se podían utilizar, pero se ha logrado que el comportamiento del robot en la práctica sea mejor en cuanto a poder utilizar todos los valores del rango. El valor del parámetro OCR3A que se estableció, es el adecuado para conseguir el rango de valores expresado.

Retomando la explicación del código, y atendiendo al último fragmento de código que se ha expuesto, cuanto menor fuese el valor de la velocidad, más interrupciones se tienen que producir para proceder a un cambio de secuencia del motor, con lo que se consigue que la velocidad de giro del motor sea menor. Otro aspecto importante es que, según si el valor de velocidad es positivo o negativo, se recorre la secuencia en un sentido u otro, la cual se ha implementado a partir de una estructura switch-case, como se muestra a continuación.

```
switch(Secuencia_Actual_A) {
  case 3:
      digitalWrite(M11, LOW);
      digitalWrite(M12, LOW);
      digitalWrite(M13, HIGH);
      digitalWrite(M14, HIGH);
      break;
  case 2:
      digitalWrite(M11, LOW);
      digitalWrite(M12, HIGH);
      digitalWrite(M13, HIGH);
      digitalWrite(M14, LOW);
      break;
```

```
case 1:
      digitalWrite(M11, HIGH);
      digitalWrite(M12, HIGH);
      digitalWrite(M13, LOW);
      digitalWrite(M14, LOW);
      break;
  case 0:
      digitalWrite(M11, HIGH);
      digitalWrite(M12, LOW);
      digitalWrite(M13, LOW);
      digitalWrite(M14, HIGH);
      break;
  }
```

Este código que se ha presentado para el primer motor es análogo al que se utiliza para los otros dos. Gracias a este método se ha conseguido lograr que el robot pueda describir todos los tipos de movimientos que eran objetivo de este proyecto, con un número bastante reducido de líneas de código.

En una de las secciones de código que se han presentado, se pudo ver que se incluían instrucciones en las que se empleaba una vector de nombre avanceMotores. Este se utiliza para los cálculos de la odometría del robot, pero esto es una idea que se explicará más adelante en la memoria.

Antes de concluir con la parte referente al Arduino, queda explicar la forma en la que se envía la pose de vuelta al ordenador, en concreto a un nodo en ROS en el que se ha implementado un socket UDP. Tan solo es necesario generar un string con el formato adecuado para que lo interprete el receptor y utilizar la función que ya se ha descrito previamente para el servidor web.

```
if(ciclos > 6000){ // Solo se realiza el envío si han pasado un
número de ciclos para reducir la sobrecarga
    respuesta = "(" + String(Pose[0]) + "," + String(Pose[1]) +
"," + String(Pose[2]) + ")";
    sendData("AT+CIPSEND=" + String(1) + "," + respuesta.length()
+ "\r\n", 10); // Modo de envío de datos
    sendData(respuesta, 10);
    ciclos = 0;
  }
```

Solo se envía la pose cuando se han producido un número determinado de ciclos de programa, y se haya recibido un mensaje en el último ciclo, de modo que de tiempo de enviar los datos, antes de que sea necesario recibir unos nuevos valores de velocidad de los motores, logrando de este modo que el funcionamiento del robot no se vea afectado por la necesidad de tener que estar utilizando el módulo de comunicaciones Wifi de forma constante.

El programa completo con la implementación realizada en el Arduino MEGA se presenta en el apartado 6.2.4, en el capítulo de los anexos.

Tras lograr el objetivo perseguido ya se contaba con un sistema que permitía enviar mensajes desde ROS hasta el Arduino, junto a un sistema de control del movimiento de los motores con el que ya se podían aprovechar las ventajas que supone el haber diseñado un robot con ruedas omnidireccionales. De este modo, se podía continuar con el desarrollo del proyecto, elaborando un sistema implementado en ROS en el que se obtuviesen los valores adecuados de velocidad en los motores, y que de esta forma el robot describiese los movimientos solicitados desde un mando.

3.6. Incorporación de un mando en el sistema de control

Tras haber conseguido un sistema con el que se puede establecer una comunicación Wifi entre ROS y la placa Arduino MEGA, junto a un método avanzado para el control del movimiento de los motores paso a paso, el siguiente hito que se debía alcanzar en el proyecto era generar el mensaje con las velocidades de los motores, a partir de la interacción del usuario con un mando. En concreto se quería utilizar un mando común de videoconsola.

De este modo, el primer paso fue incluir en el entorno de trabajo ROS, un nodo que reconociese el mando tras conectarlo un mando al ordenador, en este caso empleando un cable USB para ello. Para ello, se utilizó una herramienta que se ofrece en la página web oficial de ROS [32], que permite iniciar un nodo que recopila la información del estado de los botones y *joysticks* de cualquier mando genérico. De este modo, posteriormente será posible controlar el robot mediante múltiples controladores diferentes.

Para instalar el nodo joy_node del paquete joy, hay que introducir la siguiente instrucción en la ventana de comandos del sistema Linux.

```
$ sudo apt-get install ros-indigo-joy
```

Tras la instalación, hay que configurar el mando, para que el sistema operativo lo pueda reconocer. Escribiendo en la ventana de comandos lo siguiente:

```
$ ls /dev/input/
```

Se mostrará una lista con todos los dispositivos de entrada conectados al ordenador. De este modo se puede identificar el nombre que le ha asignado el sistema al mando. Habitualmente estará identificado como “js0”, siendo este el caso del presente proyecto.

En este momento se puede verificar el mando que se ha conectado lo ha reconocido correctamente el ordenador. Enviando las siguientes palabras a través de la ventana de comandos:

```
$ sudo jstest /dev/input/js0
```

Se puede comprobar las salidas que está generando el estado del mando. Actuando sobre cualquiera de los botones del controlador se pueden visualizar los cambios que se producen en la información que se muestra por pantalla.

Una vez se han realizado esta serie de comprobaciones, se debe hacer accesible el mando para el nuevo nodo que se ha incorporado al modelo. Para ello se utilizan las siguientes instrucciones:

```
$ ls -l /dev/input/js0

$ sudo chmod a+rw /dev/input/js0
```

Antes de iniciar el nuevo nodo, hay que indicarle al mismo cual es el dispositivo de control que se debe utilizar por defecto, en este caso sería el identificado como “js0”.

```
$ roscore

$ rosparam set joy_node/dev "/dev/input/js0"          (En una ventana nueva)
```

De este modo, para iniciar el nodo:

```
$ rosrun joy joy_node
```

Al introducir el texto anterior se iniciará el nodo, pero este no muestra nada por pantalla. En caso de querer visualizar los valores de los botones y *joysticks* del mando hay que enviar la siguiente instrucción en una nueva terminal:

```
$ rostopic echo joy
```

Gracias a todo lo descrito en las anteriores líneas ya se contaba en el modelo con un nodo que reconocía una gran variedad de mandos, y se podía acceder a la información del estado de sus sensores.

El siguiente paso consistía en recopilar esta información en un nuevo nodo que estuviese subscrito al tópico “joy” en el que el primer nodo publicaba la información del controlador. En este sentido, y de nuevo empleando las herramientas ofrecidas en la web de ROS [33], se puede encontrar un código que realiza lo anterior, y que utiliza estos datos para controlar el movimiento de un elemento gráfico al que se hace referencia en el tutorial para usuarios que se estén iniciado en el entorno de ROS. En este caso, el código se quería utilizar para otro propósito, pero sí se podía utilizar como modelo base.

Además, se ofrece una guía de instalación, pero en ella se ubica el nodo, y se le asigna un nombre, que no son los adecuados para este proyecto. De este modo, se creó un nuevo fichero, con extensión .cpp ya que se programaría en el lenguaje C++, y se guardó el mismo en la carpeta src dentro de ros_arduino_python. Debido al lenguaje de programación que se empleó, hubo que modificar el archivo CMakeLists.txt, del paquete

Ros_Arduino_Bridge, para que este nuevo fichero se compilase al ejecutar la instrucción catkin_make.

Habitualmente, la forma de proceder ante este tipo de necesidades de diseño es utilizar un nodo "teleop_twist_joy", que también se puede encontrar en la página web oficial de ROS [34]. Este nodo recibe la información del *joystick*, y genera un cmd_vel, que es un mensaje que transmite las velocidades adecuadas de los motores atendiendo a lo que se indica desde el controlador. No obstante, a pesar de que este es el estándar para el movimiento de los robots, define las velocidades en las direcciones de los ejes x e y para un robot diferencial, por lo que no resulta adecuado para el control del robot omnidireccional diseñado en este proyecto, justificando de este modo el que se haya implementado un sistema en el que el nodo define directamente la velocidad de cada motor.

En el archivo CMakeLists, al que se hizo referencia en los párrafos anteriores, se incluyeron una seria de instrucciones. En primer lugar, se debía permitir el acceso a los paquetes joy, al cual se ha hecho referencia al inicio de este apartado, y a roscpp, que se emplea al trabajar con el lenguaje de programación C++.

```
find_package(catkin REQUIRED COMPONENTS
  joy
  roscpp
)
```

También hay que incluir la instrucción que habilita al sistema ROS para encontrar las rutas donde se encuentran las librerías que se han incluido dentro del código.

```
include_directories(
  ${catkin_INCLUDE_DIRS}
)
```

Por último, se incluyen las siguientes líneas en las que se identifica el nodo, en este caso con el nombre joy_to_speed, que se quiere incluir en el proceso que realiza la operación catkin_make.

```
add_executable(joy_to_speed src/joy_to_speed.cpp)
target_link_libraries(joy_to_speed ${catkin_LIBRARIES})
```

De este modo ya se cuenta con un nodo, perteneciente al paquete Ros_Arduino_Bridge en el que se va a procesar la información proveniente del nodo joy, generando como salida las velocidades que se deben asignar a cada motor para describir el movimiento solicitado desde el mando. Para ello, hay que recordar que era necesario realizar un importante número de modificaciones respecto al modelo inicial del fichero.

De este modo, resulta conveniente explicar el funcionamiento de algunas partes de este código.

En primer lugar, el nodo debe estar subscrito a al tópico “joy” como se mencionó anteriormente.

```
joy_sub_ = n_.subscribe<sensor_msgs::Joy>("joy",
 10, &ControlRemoto::joyCallback, this);
```

Hay que inicializar un parámetro, al que se le ha dado el nombre linear_, el cual permitirá posteriormente, acceder a la posición correcta del vector “axes”, en el que se almacenan los datos del estado de cada uno de los sensores del mando. En el caso de este proyecto, se controlará la rotación del robot mediante el eje x del *joystick* derecho, mientras que para el desplazamiento se emplearán los dos ejes del *joystick* izquierdo. Para almacenar esta información se definen las dos siguientes variables, en las cuales se pueden almacenar dos vectores de 3 posiciones. En concreto se trata de un geometry_msgs::Twist, el cual se emplea habitualmente para los tópicos cmd_vel, en los que se envía la velocidad de traslación, en m/s, y la de rotación, en rad/s, de un robot diferencial. En este caso, debido a que no se está trabajando con un robot diferencial, se ha adaptado el código para almacenar en estas variables la información importante proveniente del nodo previo, con la que definir estas velocidades atendiendo a las características del prototipo fabricado. En la implementación realizada, se utiliza el solo primer vector que contiene el tipo de dato, identificado como “linear”.

```
geometry_msgs::Twist twistJoystickIzq;
geometry_msgs::Twist twistJoystickDer;
```

Atendiendo a los parámetros que se quieren almacenar, se utilizarán las dos primeras posiciones (x e y) para la variable asociada al *joystick* izquierdo, y solo la posición x en el caso de la variable relacionada con el *joystick* derecho.

Se recogen los valores en la función de recepción de la información publicada por el nodo “joy_node”. Esto se puede hacer de este modo ya que las variables antes definidas son de tipo global.

```
void ControlRemoto::joyCallback(const sensor_msgs::Joy::ConstPtr& joy)
{
  // Ejes X e Y del joystick izquierdo
  twistJoystickIzq.linear.x = -100 * joy->axes[linear_ - 1];
  twistJoystickIzq.linear.y = 100 * joy->axes[linear_];

  // Eje X del joystick derecho
  twistJoystickDer.linear.x = -100 * joy->axes[linear_ + 2];
}
```

En el anterior fragmento del código se puede observar que los parámetros se multiplican por un factor 100, para que de este modo el rango de valores con el que se trabaje vaya de – 100 a 100, lo cual es más cómodo que de -1 a 1, que es el rango en el que manda los datos de los actuados el nodo previo. En relación al signo negativo, se ha utilizado para que los valores generados por los movimientos hacia la derecha o hacia arriba de los *joysticks* sean los positivos.

Por otro lado, en el bucle principal se realiza una llamada a una función de tipo void, a la cual se le ha asignado el nombre "getVel". En ella, a partir de la información almacenada, se decide la velocidad que se debe asignar a cada motor para que el robot describa el movimiento que se haya indicado desde el mando.

Para poder comprender esa parte del código, es necesario exponer una serie de ideas de forma previa. En primer lugar, se debe diseñar un modelo que permita obtener la velocidad de cada uno de los motores a partir de tres contribuciones distintas, dadas por los 3 ejes que con los que se está trabajando. De este modo, tras hacer la media de las tres contribuciones se obtiene el valor final, el cual se enviará al microcontrolador Arduino.

En primer lugar, con el joystick derecho se controlan los movimientos de rotación pura.

Figura 108. Movimientos de rotación pura del robot

Los cuales, se lograban activando los motores en los sentidos de giro que se muestran en la siguiente tabla.

Identificador	Sentido de giro motor 1	Sentido de giro motor 2	Sentido de giro motor 3
1	Derecha	Derecha	Derecha
2	Izquierda	Izquierda	Izquierda

Tabla 20. Sentidos de giro de los motores para los identificadores 1 y 2

Inicialmente se asignarán las velocidades de los motores entre -255 y 255. Posteriormente se realizarán las correcciones necesarias para que el Arduino pueda trabajar con los valores que reciba. De este modo, si se asocia el sentido de giro hacia la derecha como positivo, y el sentido de giro hacia la izquierda como negativo, se deben sustituir los siguientes valores en la tabla anterior.

Identificador	Velocidad de giro M1	Velocidad de giro M2	Velocidad de giro M3
1	255	255	255
2	- 255	- 255	- 255

Tabla 21. Velocidades de giro de los motores para los identificadores 1 y 2

En la práctica, el valor del eje x del *joystick* derecho definirá tanto el signo como el valor que se asigne finalmente al motor. Si por ejemplo el *joystick* solo se ha desplazado la mitad del recorrido, en vez de asignar un 255 o un – 255, se asignará la mitad de ese valor. Las expresiones con las que se consigue este comportamiento son las siguientes.

```
// Contribucion del eje X del joystick derecho (Rotacion pura).
m1_vel = 255*twistJoystickDer.linear.x/MODMAX;
m2_vel = 255*twistJoystickDer.linear.x/MODMAX;
m3_vel = 255*twistJoystickDer.linear.x/MODMAX;
```

MODMAX es una Macro de valor 100, que se corresponde con el valor máximo que puede tomar un eje. De este modo queda definida la contribución a la velocidad de los motores debida al eje x del *joystick* derecho.

Por otro lado, y en este caso atendiendo al eje y del joystick izquierdo, este se encargará de controlar el movimiento del robot hacia adelante o hacia atrás.

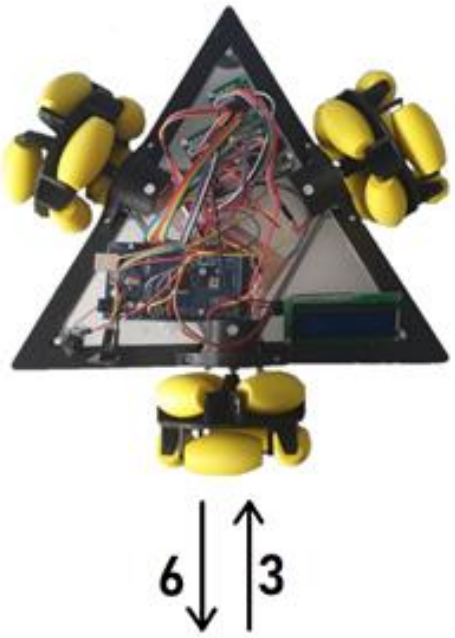


Figura 109. Traslación pura del robot en el eje Y

Estos movimientos, como ya se ha comprobado en los primeros métodos de control del robot omnidireccional se consiguen mediante los siguientes sentidos de giro de los motores.

Identificador	Sentido de giro motor 1	Sentido de giro motor 2	Sentido de giro motor 3
3	Derecha	Izquierda	-
6	Izquierda	Derecha	-

Tabla 22. Sentidos de giro de los motores para los identificadores 3 y 6

En este caso, y asignado los valores de igual modo que para el caso anterior, la tabla resultante es la siguiente.

Identificador	Velocidad de giro M1	Velocidad de giro M2	Velocidad de giro M3
3	255	- 255	-
6	- 255	255	-

Tabla 23. Velocidades de giro de los motores para los identificadores 3 y 6

Esto se traduce en que las instrucciones en el código sean las siguientes.

```
// Contribucion del eje Y del joystick izquierdo
m1_vel += 255*twistJoystickIzq.linear.y/MODMAX;
m2_vel -= 255*twistJoystickIzq.linear.y/MODMAX;
```

En este tipo de movimiento el tercer motor debe permanecer en reposo.

De este modo, se añade a lo anterior la contribución a la velocidad de los motores dada por el movimiento del *joystick* izquierdo en el eje y.

Por último, y en este caso atendiendo al eje x del joystick izquierdo, este debe controlar la navegación del robot hacia la derecha e izquierda del mismo. Sin embargo, no existe un movimiento simple del robot por el que se puede desplazar en dicha dirección, sino que resulta necesario realizar un movimiento combinado entre los 3 motores cuyo resultado sea un desplazamiento, sin rotación, sobre el eje x.

Para obtener las velocidades con las que se conseguiría este tipo de movimiento, se han seleccionado los valores que, junto a la contribución ya definida para el eje y, lograrían realizar uno de los movimientos en diagonal que si puede realizar el robot de forma simple.

Esta idea se puede ver con mayor claridad mediante la siguiente representación.

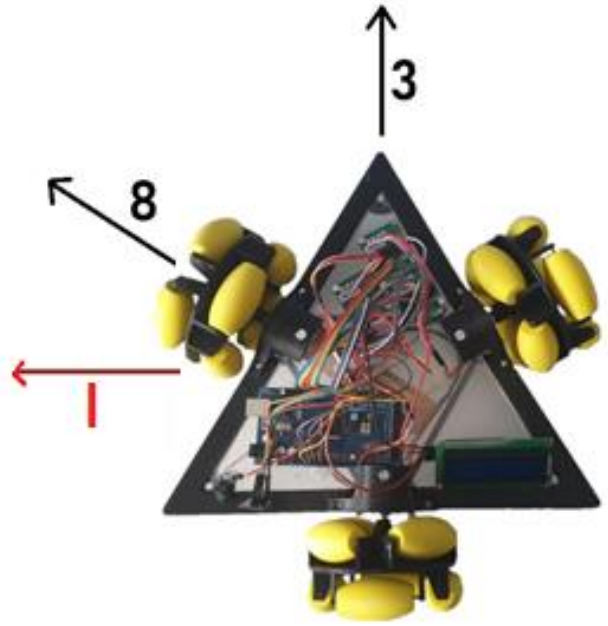


Figura 110. Nueva dirección de movimiento a implementar

Cuando el *joystick* izquierdo esté apuntando hacia la dirección representada con el número 8, la velocidad vendrá definida por la contribución en el eje y, la cual ya se puede calcular gracias a las expresiones antes mostradas, y una contribución en el eje x, cuyas expresiones se quieren obtener. También es conocido el resultado que debe resultar de la suma de ambas contribuciones, que es el que se muestra en la siguiente tabla, y que se corresponde con el tipo de movimiento 8 del robot.

Identificador	Velocidad de giro M1	Velocidad de giro M2	Velocidad de giro M3
8	-	- 255	255

Tabla 24. Velocidades de giro de los motores para el identificador 8

Para calcular la contribución en el eje y, es necesario conocer el ángulo que se forma entre la dirección 8 y la dirección I. Este es de 30º, ya que se corresponde con una de las direcciones perpendiculares de las ruedas. El cálculo por tanto sería el siguiente:

Ecuación 42

$$ContribuciónY = sen(30º) \cdot ValoresMovDir3 = 0.5 \cdot ValoresMovDir3$$

	Velocidad de giro M1	Velocidad de giro M2	Velocidad de giro M3
ValoresMovDir3	255	- 255	-
0.5 · ValoresMovDir3	127.5	- 127.5	-

Tabla 25. Obtención de la contribución en el eje Y para el movimiento con identificador 8

Al sumar los valores obtenidos con la contribución en el eje x, se deben obtener los valores de velocidad que provocan que el robot se desplaza hacia la dirección identificada con el número 8. Junto a esto, la expresión para la contribución en x en la situación planteada es la siguiente:

Ecuación 43

$$ContribuciónX = cos(30º) \cdot ValoresMovDirI \cong 0.866 \cdot ValoresMovDir3$$

Representando ambas ideas en la siguiente tabla se obtienen los valores deseados.

	Velocidad de giro M1	Velocidad de giro M2	Velocidad de giro M3
0.866 · ValoresMovDirI	-127.5	127.5	255
ValoresMovDirI	**-147.23**	**- 147.23**	**294.46**

Tabla 26. Obtención de las velocidades necesarias para describir un movimiento en el eje X

Si se asignan los valores resaltados en la anterior tabla, a la dirección negativa del eje x, el resultado de sumar las dos contribuciones en las que interviene el *joystick* izquierdo, son las velocidades correspondientes a la dirección 8, de modo que se reflejaría el comportamiento correcto. De este modo se pueden definir las direcciones I (Izquierda) y D (Derecha) del modo en el que se muestra en la siguiente imagen.

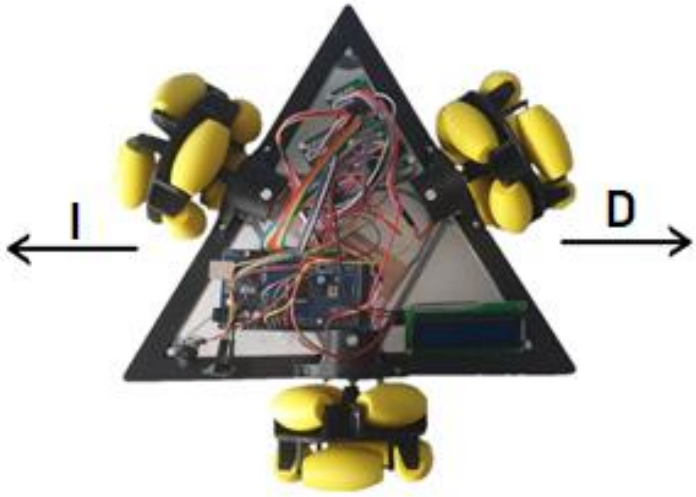


Figura 111. Traslación pura del robot en el eje X

Las velocidades de los motores con las que se conseguirían describir estos movimientos serían las que se muestran en la siguiente tabla. En cuanto a las velocidades para el desplazamiento hacia la derecha, estas serán las mismas que hacia la izquierda, pero con los signos opuestos.

Identificador	Velocidad de giro M1	Velocidad de giro M2	Velocidad de giro M3
I	- 147.23	- 147.23	294.46
D	147.23	147.23	-294.46

Tabla 27. Velocidades de giro de los motores para los identificadores D e I

Con esta información ya se puede concluir el tramo del código que se está analizando.

```
m1_vel += 147.23*twistJoystickIzq.linear.x/MODMAX;
m2_vel += 147.23*twistJoystickIzq.linear.x/MODMAX;
m3_vel -= 294.46*twistJoystickIzq.linear.x/MODMAX;
```

Es importante mencionar que, aunque uno de los valores de velocidad obtenidos es mayor a 255, posteriormente, tras realizar la media de las contribuciones, no será posible que el resultado sea mayor a 255. En realidad, será considerablemente menor a 255. Además, recordando que en la implementación realizada en Arduino, solo se podrán mover los motores si las velocidades de estas están dentro del rango de - 210 a 210.

En relación a esto último, y dado que, si solo se utiliza uno de los *joysticks*, las velocidades tras realizar la media de las 3 contribuciones serían muy bajas, se incluye en el código un reescalado en esos casos. De esta forma, tanto si se realiza una rotación pura, como si se realiza una traslación pura, se aumentan los valores de las velocidades de los motores, multiplicando todos ellos por un mismo factor, el cual logra que el resultado está próximo a 210 (o – 210), de modo que el movimiento del robot no sea excesivamente lento.

```
// Se reescalan los resultados para hacer que el robot no se mueva
excesivamente lento
  // en caso de rotacion pura o traslacion pura


  // Rotacion pura
    if(twistJoystickIzq.linear.x          ==          0          &&
twistJoystickIzq.linear.y == 0){
      m1_vel *= 2.4;
      m2_vel *= 2.4;
      m3_vel *= 2.4;
    }


  // Traslacion pura
  if(twistJoystickDer.linear.x == 0){
      m1_vel *= 1.7;
      m2_vel *= 1.7;
      m3_vel *= 1.7;
    }
```

El siguiente paso ya es calcular la media de las contribuciones y convertir el resultado en un número entero que enviar al Arduino.

```
// Se hace la media de las 3 contribuciones
m1_vel = m1_vel/3;
m2_vel = m2_vel/3;
m3_vel = m3_vel/3;
// Se convierte el resultado en un entero antes de enviarlo
m1_vel = int(m1_vel);
m2_vel = int(m2_vel);
m3_vel = int(m3_vel);
```

Aplicando todo lo que se ha expuesto para el aumento de la velocidad del robot, en el caso de la traslación pura, en los movimientos hacia derecha, izquierda, hacia adelante o hacia atrás del robot, la velocidad que se consigue no es tan alta. Esto se debe a que las velocidades para los movimientos en diagonal son mayores que para los mencionado. Esta es la razón por la que se aplica un factor de 1.7 en caso de traslación pura, y un factor de 2.4 en el caso de rotación pura.

Una alternativa a este sistema es multiplicar en ambos casos por 2.4, y luego comprobar si se ha excedido el límite de velocidad que puede reproducir el robot, en cuyo caso se corrigen las velocidades dividiendo por un factor que logre que la mayor velocidad que se envíe sea como máximo de un valor determinado. Este modelo se implementó en el sistema mediante el siguiente código en el que MAXVEL es el máximo valor de velocidad que se permite.

```
if(abs(m1_vel) > MAXVEL || abs(m2_vel) > MAXVEL || abs(m3_vel) > MAXVEL){
    // Se averigua cual es el mayor
    if(abs(m1_vel) >= abs(m2_vel)){
      if(abs(m1_vel) >= abs(m3_vel)){
        corregirVel = abs(m1_vel)/MAXVEL;
      }
      else{
        corregirVel = abs(m3_vel)/MAXVEL;
      }
    }else{
```

```
if(fabs(m2_vel) >= fabs(m3_vel)){
        corregirVel = abs(m2_vel)/MAXVEL;
      }else{
        corregirVel = abs(m3_vel)/MAXVEL;
      }
    }
    m1_vel = m1_vel/corregirVel;
    m2_vel = m2_vel/corregirVel;
    m3_vel = m3_vel/corregirVel;
  }
```

Este código hay que situarlo después de calcular la media de las contribuciones y antes de convertir los resultados en números enteros. En el modelo final del robot no se ha incluido este método de cálculo de las velocidades ya que, aunque se lograba que el sistema se desplazase a mayor velocidad, el control era mucho menos preciso.

Por último, se utiliza una variable de tipo Vector3 para almacenar las velocidades obtenidas y posteriormente se publica el mensaje.

```
// Se introducen las velocidades de los tres motores en un vector
de 3 posiciones


  geometry_msgs::Vector3 msg;


  msg.x = m1_vel;
  msg.y = m2_vel;
  msg.z = m3_vel;


  // Se publican las velocidades
  chatter_pub.publish(msg);
```

Al tópico en el que se publica la información se le ha dado el nombre de “motors_speed”.

```
ros::Publisher                    chatter_pub                    =
n2.advertise<geometry_msgs::Vector3>(

"motors_speed",1000);
```

El último paso para poder enlazar todo el sistema que se ha desarrollado con el nodo del Ros_Arduino_Bridge, es que este último esté subscrito al tópico que se acaba de generar. Para ello se ha añadido la siguiente línea al código del fichero arduino_node.py.

```
rospy.Subscriber("motors_speed", Vector3, self.callback)
```

Por otro lado, la función que se desarrolla ante la llegada de un mensaje de ese tópico es la siguiente.

```
# Topico de la velocidad de los 3 motores
    def callback(self, data):
      self.controller.motors_speed(data.x,data.y,data.z)
        rospy.loginfo("El mensaje recibido es: [%f,%f,%f]",
data.x,data.y,data.z)
        return
```

A partir de este punto el funcionamiento es el que se expuso en el apartado en el que se detalló el modo de funcionamiento del control del robot mediante Ros_Arduino_Bridge. Se consiguió de este modo que el control del robot omnidireccional partiese de un mando controlado por el usuario.

3.7. Resumen de cambios en el sistema de control

El modelo de control que se ha implementado para el robot omnidireccional ha pasado por varias etapas a lo largo del proyecto. En este apartado se pretende expresar de forma concisa los avances que se lograron con cada modelo, y los criterios por los que se decidió cambiar de sistema.

En primer lugar, y con el objetivo de percibir cómo era el comportamiento del diseño de robot realizado en la práctica, se creó un código que permitía controlar el robot mediante el envío de una serie de comandos a través de una comunicación serial. En ella el robot debía estar conectado a través de un cable USB tipo B al ordenador. Para ello se instaló un Arduino UNO, pero tras verificar que la navegación del robot era correcta, hubo que sustituir el microcontrolador por uno nuevo con el que se pudiese establecer una conexión a un punto de acceso mediante Wifi.

De este modo, se instaló un Arduino WeMos D1, el cual estaba basado en un chip ESP-8266, gracias al cual se pudo diseñar un sistema de control basado en la creación de un servidor web. El Arduino creaba el servidor HTTP, al cual se podía conectar un cliente introduciendo la dirección IP donde se alojaba dicho servidor en un navegador. De este modo, se logró que el robot pudiese recibir datos de forma inalámbrica. Sin embargo, esta placa no contaba con el número de pines necesarios para controlar los 3 motores paso a paso. La solución pasó por sustituir el microcontrolador por un Arduino MEGA, al que hubo que incorporar un módulo externo de comunicaciones Wifi, el ESP-01. Se modificó el programa con el que se contaba para reproducir el comportamiento que se había

logrado, pero en este caso utilizando los comandos AT, y pudiendo controlar el movimiento de los motores.

El siguiente objetivo del proyecto consistió en que el sistema de control permitiese explotar las posibilidades que ofrecía el diseño de robot que se había implementado. Ante esta situación, se concibió la plataforma ROS como un entorno adecuado para lograr este propósito, la cual se utilizaría para que el movimiento del robot se controlase mediante un mando conectado al ordenador. Ante estos planteamientos, se comenzó trabajando con la herramienta Rosserial. Para utilizar este paquete era necesario adaptarla al envío de datos de forma inalámbrica, lo cual, tras realizar numerosas pruebas, no se pudo lograr ya que Rosserial introduce una gran sobrecarga en el sistema, realizando un uso intensivo de la memoria del microcontrolador.

Ante la situación descrita en el apartado anterior, se trató de utilizar el paquete Ros_Arduino_Bridge para establecer la conexión entre el ordenador y la placa Arduino. Del mismo modo que para el caso del Rosserial, había que adaptar el software para que la comunicación fuese inalámbrica. No obstante, en este protocolo se emplean mensajes más simples, introduciendo una menor sobrecarga, por lo que finalmente se pudieron implementar las modificaciones necesarias para que el envío de los datos se realizase a través de una conexión Wifi. Inicialmente se utilizó el protocolo TCP/IP para el envío y recepción de la información, pero posteriormente se sustituyó por el protocolo UDP ya que este permitía realizar los envíos de las velocidades de los motores a mayor frecuencia.

Finalmente, tras haber logrado implementar un método con el que se podía transmitir información desde ROS al Arduino MEGA, de forma inalámbrica, se crearon los nodos de ROS necesarios para incluir un mando en el modelo, el cual se utilizaría para controlar la navegación del robot.

3.8. Implementación del sistema odométrico

Uno de los objetivos que se plantearon en el momento en el que se comenzó el proyecto, fue el de incorporar al robot un sistema odométrico que permitiese conocer su posición referida a un sistema de coordenadas, es decir, su pose.

Para ello, dentro del marco teórico de este documento, se presentó el modelo que se ha diseñado con tal propósito, el cual se basaba en definir las distintas contribuciones de cada motor, en base a si se estaba produciendo un movimiento de rotación, traslación o mixto.

Para aplicar este modelo, resulta preciso conocer en qué momento, cada uno de los motores paso a paso, está avanzando. De esta idea se puede decir que el mejor lugar donde se puede calcular la pose del robot es en el código de su microcontrolador, es decir, en el Arduino MEGA. De este modo se aprovecharía la ventaja que supone el haber instalado este tipo de motores en el robot, al poder identificar de forma exacta el momento en el que un motor ha avanzado un paso, y en qué sentido de giro lo ha hecho.

En las siguientes líneas del presente apartado se detallarán las modificaciones que hubo que realizar en el código del Arduino para incorporar este sistema al resto de funcionalidades con las que ya contaba el robot.

En primer lugar, se definen una serie de variables que serán necesarias en las distintas partes del código. Destacan en este caso, los parámetros avance y rotación, que definen lo que avanza o rota el robot por cada secuencia de activación de los motores. En relación a esto, es importante mencionar 2 aspectos en los que ha sido necesario modificar estas expresiones respecto a lo que se expuso en el modelo teórico. Se sumarán las contribuciones por cada secuencia del motor, en lugar de por cada paso completo, ya que de este modo es más fácil integrar este sistema en el código. Esto conlleva la necesidad de dividir por cuatro el valor del parámetro avance, ya que para que el robot se desplace esa distancia será necesario que se hayan activado las cuatro secuencias que corresponden a un paso completo del motor. Por otro lado, y en este caso con el objetivo de minimizar los efectos de aquellos factores que no se han modelado, tales como los deslizamientos que se pueden producir o las imperfecciones en las impresiones 3D, se modificaron los valores numéricos de ambas expresiones hasta que se logró que el sistema reflejase con mayor fidelidad la realidad.

```
// Variables globales para la odometría del robot


// Vector que almacenará la pose del robot (x e y en centímetros
y Theta en grados).
float Pose[3] = {0, 0, 90};
int avanceMotores[6] = {0,0,0,0,0,0}; // Vector en el que se
anotarán los pasos que den los motores
int ciclos = 0;
float avance = (2 * PI * 31)/(512 * 4);
float rotacion = (avance * 360)/(6 * PI * 68.9);
```

Como se puede apreciar en el anterior fragmento de código, se ha definido un vector de 6 posiciones con el nombre “avanceMotores”, en el cual se almacena la información de qué motores se han movido en la última interrupción software y en qué sentido. Esto se ha implementado se este modo ya que no es conveniente incluir código dentro de las funciones de interrupción, para realizar acciones distintas a las necesarias para realizar la acción para la cual se han concebido las interrupciones, que en este caso se utilizan para mover los motores de acuerdo a las velocidades recibidas desde ROS.

En el siguiente fragmento se muestra cómo se modifica el valor de la posición 0 del vector si el primer motor gira en un sentido, o el valor de la posición 1 en el caso de que gire en el sentido opuesto.

```
if(numero_cambio_secuencia_A > 0){
    Secuencia_Actual_A +=1;
    avanceMotores[0] +=1;
  }
  if(numero_cambio_secuencia_A < 0){
    Secuencia_Actual_A -=1;
    avanceMotores[1] +=1;
  }
```

Posteriormente, en el bucle principal del programa, se consulta la información del vector, determinando las modificaciones que se deben realizar a la pose del robot.

```
if(avanceMotores[0] > 0){ // Sentido positivo motor 1
    Pose[0] += (avance/10)*cos(((Pose[2]*PI)/180)-(PI/6));
    Pose[1] += (avance/10)*sin(((Pose[2]*PI)/180)-(PI/6));
    Pose[2] -= rotacion;
    avanceMotores[0] -=1;
  }
  if(avanceMotores[1] > 0){ // Sentido negativo motor 1
    Pose[0] -= (avance/10)*cos(((Pose[2]*PI)/180)-(PI/6));
    Pose[1] -= (avance/10)*sin(((Pose[2]*PI)/180)-(PI/6));
    Pose[2] += rotacion;
    avanceMotores[1] -=1;
  }
if(avanceMotores[2] > 0){ // Sentido positivo motor 2
    Pose[0] += (avance/10)*sin(((Pose[2]*PI)/180)-(PI/3));
    Pose[1] -= (avance/10)*cos(((Pose[2]*PI)/180)-(PI/3));
    Pose[2] -= rotacion;
    avanceMotores[2] -=1;
  }
if(avanceMotores[3] > 0){ // Sentido negativo motor 2
    Pose[0] -= (avance/10)*sin(((Pose[2]*PI)/180)-(PI/3));
    Pose[1] += (avance/10)*cos(((Pose[2]*PI)/180)-(PI/3));
    Pose[2] += rotacion;
    avanceMotores[3] -=1;
  }
```

```
if(avanceMotores[4] > 0){ // Sentido positivo motor 3
    Pose[0] -= (avance/10)*cos(((Pose[2]*PI/180))-(PI/2));
    Pose[1] -= (avance/10)*sin(((Pose[2]*PI/180))-(PI/2));
    Pose[2] -= rotacion;
    avanceMotores[4] -=1;
  }
  if(avanceMotores[5] > 0){ // Sentido negativo motor 3
    Pose[0] += (avance/10)*cos(((Pose[2]*PI/180))-(PI/2));
    Pose[1] += (avance/10)*sin(((Pose[2]*PI/180))-(PI/2));
    Pose[2] += rotacion;
    avanceMotores[5] -=1;
  }
```

De este modo, el vector en el que se almacena la pose estará preparado para el momento en el que se considere oportuno enviar esta información de vuelta a ROS, donde se procesará para realizar una representación gráfica de la posición de robot.

3.9. Recepción de la pose en un socket UDP y monitorización del robot

En el apartado anterior se indicó que, tras realizar el cálculo de la pose del robot, esta se envía a través del módulo Wifi al ordenador, donde se ha incluido un nodo de ROS (ros_odometry_publisher.py) de forma adicional a los nodos previos, que está basado en el código genérico para la publicación de la odometría en ROS [35], en el que se ha implementado un socket UDP para la recepción de dicha información.

El objetivo es procesar la información procedente del microcontrolador Arduino, para generar un mensaje de tipo odom, que es un estándar para el envío de la odometría de robots, el cual se utilizará en este proyecto para mostrar de forma gráfica la navegación del robot en tiempo real.

Para ello, lo primero es iniciar el nuevo nodo y el socket UDP.

```
import socket # Para poder utilizar los sockets

rospy.init_node('odometry_publisher') # Se inicia el nuevo nodo

odom_pub = rospy.Publisher("odom", Odometry, queue_size=50) # Se
publicara en el topico odom

odom_broadcaster = tf.TransformBroadcaster()


# Parametros para establecer la conexion

UDP_IP = "0.0.0.0"

UDP_PORT = 4446


# Se inicia el socket

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

sock.bind((UDP_IP, UDP_PORT))
```

En el bucle principal se espera a la llegada de un nuevo mensaje.

```
# Se espera a recibir el mensaje del Arduino
data, addr = sock.recvfrom(1024) # buffer size is 1024 bytes
# En data se encuentra el string "(x,y,Theta)”
```

Se interpreta el mensaje y se convierten las unidades de los parámetros de la pose a las del Sistema Internacional. También se calculan las velocidades en las que están cambiando estos parámetros para generar el mensaje odom completo. Para ello se calcula el tiempo que ha tardado en llegar la nueva pose y la variación que se ha producido en esta.

```
# Tras recibir el mensaje se actualiza la pose antes de
publicarla

stringX = ""
stringY = ""
stringTh = ""

i = 0 # iteracion
j = 0 # string que toca rellenar
```

```
 while data[i] != ")" and i < 100:

  if (data[i] == "(" or data[i] == ","):
    # Se cambia de string a rellenar
    j = j + 1
  else:
    if j == 1:
      stringX = stringX + data[i]
    if j == 2:
      stringY = stringY + data[i]
    if j == 3:
      stringTh = stringTh + data[i]
   i = i + 1

# Se convierten los valores a float para poder trabajar con
ellos

x = float(stringX)/100
y = float(stringY)/100
th = float(stringTh)*math.pi/180

# Se prepara el mensaje con la odometria del robot
dt = (current_time - last_time).to_sec() # Se calcula la
variacion de tiempo en la ultima ejecucion


# Se averigua cuanto ha cambiado la pose en el ultimo ciclo
delta_x = x - last_x
delta_y = y - last_y
delta_th = th - last_th

# Se calcula la velocidad como el espacio que se ha recorrido
entre el tiempo que se ha tardado

vx = delta_x/dt
vy = delta_y/dt
vth = delta_th/dt
```

A pesar de que el robot solo se puede mover en el plano, el mensaje que se está generando es para un sistema con 6 grados de libertad, por lo que hay que generar el cuaternión, a partir del cual se definen las orientaciones y rotaciones del robot. Junto a esto se prepara el paquete que se va a publicar en ROS.

```
# Al ser la odometria de 6 grados de libertad hay que crear el
cuaternión

odom_quat = tf.transformations.quaternion_from_euler(0, 0, th)
```

```
# Primero se publica la transformada del paso anterior
odom_broadcaster.sendTransform(
    (x, y, 0.),
    odom_quat,
    current_time,
    "base_link",
    "odom"
)

# Se publica el mensaje con la odometria
odom = Odometry()
odom.header.stamp = current_time
odom.header.frame_id = "odom"

# Se situa el robot en el plano
odom.pose.pose = Pose(Point(x, y, 0), Quaternion(*odom_quat))

# Se indica la velocidad del robot
odom.child_frame_id = "base_link"
odom.twist.twist = Twist(Vector3(vx, vy, 0), Vector3(0, 0, vth))
```

Por último, se publica el mensaje, se almacenan los valores de la pose, para conocer esta información en la siguiente iteración, y se almacena el dato de en qué momento en el que se realizaron estas operaciones.

```
# Finalmente se publica el mensaje
odom_pub.publish(odom)


# Se almacenan los valores de la pose en este momento para
conocerlos en la siguiente ejecucion
last_x = x
last_y = y
last_th = th


last_time = current_time # Se almacena el momento en el que se
termina este ciclo
```

Con todo lo anterior preparado, ya se puede iniciar la aplicación Rviz, para lo cual se debe introducir en la ventana de comandos la siguiente instrucción:

```
$ rosrun Rviz Rviz
```

Para configurar el programa, se utiliza la opción "Add", del menú que se presenta a la izquierda de la pantalla, en el cual se debe añadir la odometría generada en el desplegable "By topic". De este modo, una vez que el nodo comience a publicar los mensajes, se podrá seleccionar en la casilla "Fixed Frame" del menú principal, el mensaje de tipo odom que el software está reconociendo.

Una vez que el robot comienza a moverse, la representación mostrada por pantalla comienza a cambiar, mostrando la trayectoria que está describiendo el robot. Para ello se indica la dirección y sentido en el que se encuentra el robot moviéndose mediante vectores.

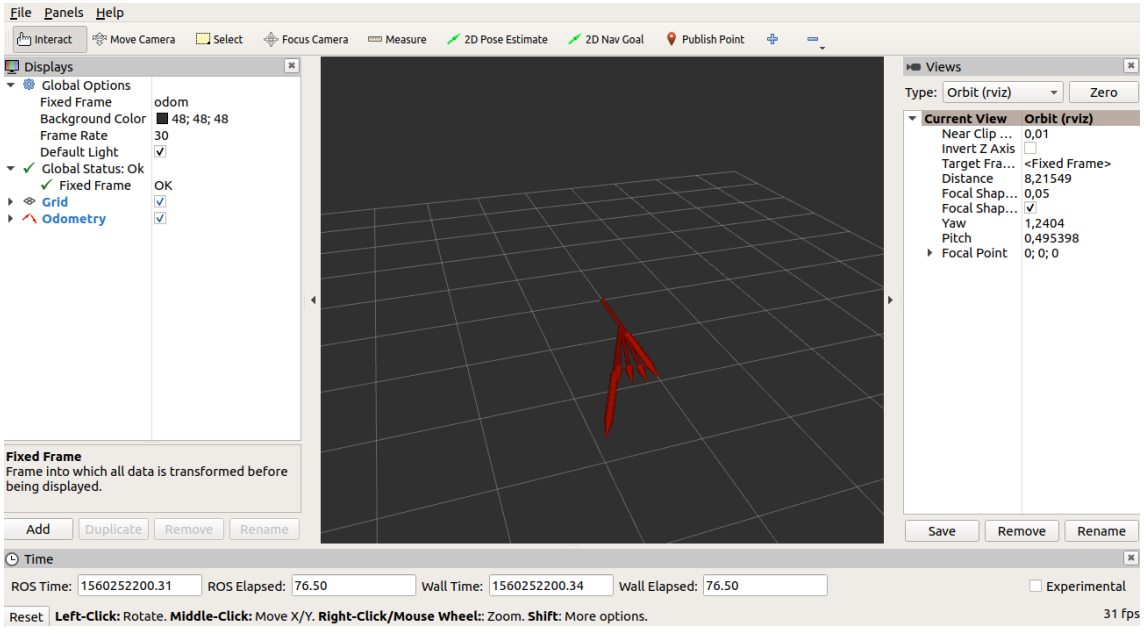


Figura 112. Visualización de la trayectoria del robot en Rviz

En el caso de la navegación representada en la anterior fotografía, el robot ha descrito un movimiento de traslación en el eje Y, posteriormente se ha producido una rotación pura, y finalmente el prototipo ha continuado con el movimiento inicial, pero en la nueva dirección.

Por último, concluyendo el presente capítulo, se debe presentar el archivo que se creó para que mediante una única instrucción se activasen todos los nodos que componen el sistema. Para ello se definió el archivo arduino.launch, que ya formaba parte del paquete Ros_Arduino_Bridge, pero en este caso se incorporaron el resto de los nodos. El archivo citado contiene lo siguiente.

```
<launch>
  <!-- joy node -->
    <node respawn="true" pkg="joy"
        type="joy_node" name="turtle_joy" >
    <param name="dev" type="string" value="/dev/input/js0" />
    <param name="deadzone" value="0.12" />
  </node>


<node name="arduino" pkg="ros_arduino_python" type="arduino_node.py"
output="screen">
      <rosparam                                              file="$(find
ros_arduino_python)/config/my_arduino_params.yaml" command="load" />
   </node>


<node pkg="ros_arduino_python" type="joy_to_speed" name="speeds"/>
<node  pkg="ros_arduino_python"  type="ros_odometry_publisher.py"
name="odom"/>


</launch>
```

Como se puede apreciar, básicamente lo que es necesario para generar un fichero de este tipo es indicar los nodos, y los paquetes en los que se encuentran. En el caso del nodo anrduino_node.py se incluye el archivo de configuración del cual debe tomar una serie de parámetros.

Capítulo IV. Conclusiones y líneas futuras

En este capítulo se presentarán los resultados que se han obtenido en el proyecto al mismo tiempo que se plantean algunas posibilidades, tanto de cara a que el proyecto sea de utilidad para otros que se lleven a cabo en el futuro, como posibles mejoras que se pueden introducir en el prototipo presentado.

4.1. Conclusiones

Sobre la base de los resultados que se han ido exponiendo a lo largo de la memoria, se puede concluir que se ha logrado llevar el proyecto hasta donde se había planteado al inicio del documento.

En relación a las piezas diseñadas para su posterior impresión en 3D, se ha conseguido dotar al robot de los medios físicos con los que poder realizar una navegación más libre que en el caso de otros tipos de robot. En este mismo sentido de las ideas, tras realizar las modificaciones oportunas, el prototipo cuenta con una estructura resistente, donde los componentes quedan fijados en su mayoría mediante tornillos.

Por otro lado, tras programar diversos sistemas de control, la versión definitiva de este, basada en el paquete Ros_Arduino_Bridge, permite que el robot realice una navegación en la que se puede realizar cualquier movimiento dentro del plano, ya sea una rotación pura, una traslación pura (hacia cualquier dirección), o un movimiento combinado. Además, se adaptó el código para que la comunicación entre el Arduino MEGA instalado en el robot, y el sistema implementado en ROS fuese a través de una red Wifi. Para ello hubo que incorporar un módulo ESP-01 al microcontrolador. Esto último, junto a la incorporación de una batería para la alimentación del prototipo, han permitido que este sea completamente inalámbrico.

Para controlar el robot, de modo que se pudiese aprovechar el que este cuente con ruedas omnidireccionales, resultaba conveniente utilizar un mando. Para conseguir que esto fuese posible se ha implementado un sistema en ROS que recopila la información de los botones y *joysticks* del mando, la procesa con el objetivo de determinar las velocidades adecuadas de los motores para describir el movimiento indicado a la velocidad solicitada, y finalmente se envía a la placa Arduino.

Del mismo modo, ha sido objeto de este proyecto, elaborar un modelo para el cálculo de la odometría del robot, el cual hubo que ajustar en el momento de realizar la implementación del mismo en el robot, para que el comportamiento reflejase con una mayor fidelidad la realidad.

Adicionalmente, la pose generada, en la que queda definida la posición del robot respecto a un sistema de referencia, se envía a un socket UDP integrado en el sistema ROS principal, donde se genera un mensaje de tipo Odom con el que se puede realizar una representación gráfica de la navegación del robot, gracias a la aplicación Rviz.

Todo esto supone haber satisfecho, tanto los objetivos iniciales como los que se presentaron a lo largo del desarrollo de este TFG.

4.2. Conclusions

Based on the results that have been exposed throughout the text, the conclusion to be drawn is that the main objectives of the project have been successfully achieved.

As far as the 3D designed models are concerned, the robot has a structure that allows it to perform a freer navigation than in the case of other common robots. In addition to that, after making the appropriate modifications, the prototype has a resistant structure, where the components are fixed by screws.

On the other hand, after programming a variety of control systems, the final version, based on the Ros_Arduino_Bridge package, allows the robot to perform a navigation in which any movement over a surface can be made, either a pure rotation, or a pure translation, or a combined movement. In addition, the code was adapted to make the communication between the Arduino MEGA installed in the robot, and the system implemented in ROS through a Wi-Fi network. In order to make this possible, an ESP-01 module had to be added to the microcontroller. This last idea, together with the incorporation of a battery for the power supply of the prototype, have allowed it to be completely wireless.

To control the robot, taking advantage of its movement system, it was advisable to use a controller. To make this possible, a system has been implemented in ROS, which collects the information of the buttons and joysticks of the controller, then it is processed with the aim of selecting the appropriate speeds for the motors in order to describe the requested movement, and finally it is sent to the Arduino board.

Apart from the aforementioned ideas, another objective of this project was to develop a model for the calculation of the robot odometry, which had to be adjusted when the robot was tested in order to achieve a greater fidelity in the results.

Additionally, the generated pose parameters are sent to a UDP socket integrated in the main ROS system, where an Odom-type message is generated. After that, it is possible to graph the robot navigation, thanks to the Rviz application.

In light of these arguments it is fair to say that the obtained prototype has satisfied the objectives presented at the beginning of the project.

4.3. Base para otros proyectos y líneas futuras

Existen dos aspectos que han contado con una gran relevancia en el desarrollo de este Trabajo de Fin de Grado, y que pueden ser de utilidad para otro tipo de proyectos.

En primer lugar, en este documento se ha expuesto gran cantidad de información referente al diseño de robots omnidireccionales, y en especial al modelado de sus ruedas. En relación a esto, en muchos casos, no resultaría muy complicado adaptar este sistema de movimiento a otros robots u otro tipo de elementos móviles, que cuenten con una

maniobrabilidad más convencional. De este modo se mejoraría la navegación de dicho sistema.

Esto podría ser de gran utilidad, por ejemplo, en entornos industriales, para lo cual habría que adaptar, tanto las dimensiones de las ruedas, como los materiales empleados en su fabricación, a los nuevos requerimientos.

Por otro lado, y en este caso atendiendo a la incorporación en el sistema de un módulo de bajo coste, para el establecimiento de una conexión a una red Wifi, esto supone una herramienta con un gran potencial, que se puede incorporar en aquellos proyectos en los que desea realizar una acción, controlándola o monitorizándola de forma inalámbrica.

De forma más específica, se ha mostrado en este documento la forma de utilizar un ESP-01, junto a una placa Arduino, para tanto enviar como recibir mensajes a través de una conexión Wifi. Además, se ha expuesto un modelo en el que se incorpora la plataforma ROS a esta comunicación, lo cual aumenta sustancialmente las posibilidades de emplear este sistema en otro tipo de robot, sin que estos tengan por qué ser móviles.

No obstante, existen aspectos que no se han incluido en este desarrollo, pero que supondrían una mejora del resultado obtenido.

En primer lugar, como el diseño inicial de las ruedas omnidireccionales mostró un comportamiento correcto a nivel práctico, se mantuvo el mismo diseño hasta el final. Es posible que, con un modelo distinto de rueda omnidireccional, o simplemente aumentando el número de rodillos que se colocan en las hileras, reduciendo el tamaño de estos, se pudiese conseguir una mejor navegación, incluso en terrenos más irregulares.

Por otro lado, se podría mejorar el modelo teórico para el cálculo de la odometría, incorporando al mismo, un estudio de las imprecisiones derivadas de que las ruedas pueden deslizar, de que las medidas no son exactas, y otros aspectos que lograsen que el modelo teórico se asemeje más al comportamiento que se percibe en la práctica.

En relación con la representación gráfica en Rviz, tan solo se muestra un vector que indica la dirección y sentido de avance del robot. Se podría crear un modelo del robot para el software de forma que la representación fuese más ilustrativa.

Por supuesto, sustituir los componentes instalados en el robot por otros con mayores prestaciones, permitiría mejorar el funcionamiento de los sistemas implementados, o incluso añadir nuevas funcionalidades al prototipo.

Capítulo V. Presupuestos

En este trabajo se ha tratado de elaborar un prototipo que contase con las funcionalidades marcadas por los objetivos iniciales, pero tratando de ajustar el precio del mismo lo máximo posible. Para el cálculo del presupuesto del proyecto se deben diferenciar los costes de materiales y los de desarrollo.

Para los primeros, se han seleccionado los precios que tienen los productos en las tiendas locales en el momento en el que se redacta este documento. En relación a las piezas impresas en 3D se ha tomado el precio de acuerdo al material utilizado y la cantidad de este.

Componente	Precio unitario (€/ud)	Cantidad (ud)	Total (€)
Rodillos de filaflex	0,52	24	12,48
Soporte de rodillos de PLA	2	3	6
Vértices para la estructura de PLA	0,62	3	1,86
Superficie interna de poliestireno	3	1	3
Soporte para los motores de PLA	0,71	3	2,13
Soporte para el módulo ESP-01 e interruptor	0,2	1	0,2
Soporte para batería LiPo de PLA.	0,1	2	0,2
Arduino MEGA	15,98	1	15,98
Motor paso a paso 28BYJ-48 con controlador ULN2003APG	3,73	3	11,19
Módulo ESP-01	4,47	1	4,47
Interruptor	0,32	1	0,32
Protoboard de 170 agujeros	1,28	1	1,28
Regulador de tensión MP1584EN	2,13	1	2,13
Batería LiPo 7.4 V, 1500 mAh	19	1	19
Bolsa de 30 tornillos de 4 x 40 mm	1,64	1	1,64
Bolsa de 30 tuercas para tornillo de 4 x 40 mm	0,95	1	0,95
Bolsa de 30 tornillos de 3 x 16 mm	1,64	1	1,64
Bolsa de 30 tuercas para tornillo de 3 x 16 mm	0,95	1	0,95
Bolsa de 40 clavos de 2.2 x 55 mm	1,59	1	1,59
	Presupuesto de materiales		**87,01**

Tabla 28. Presupuesto de materiales

En relación a los costes del desarrollo, estos se dividen distintas categorías atendiendo a las tareas que se han realizado en el proyecto.

Tareas realizadas	Precio/hora (€/h)	Horas totales (h)	Total (€)
Búsqueda de información	15	30	450
Pruebas	15	50	750
Diseños	20	10	200
Programación	25	160	4000
Montaje	15	2	30
Documentación	15	60	900
	Costes del desarrollo		**6330**

Tabla 29. Costes del desarrollo

Atendiendo a estos resultados, el balance total es el siguiente.

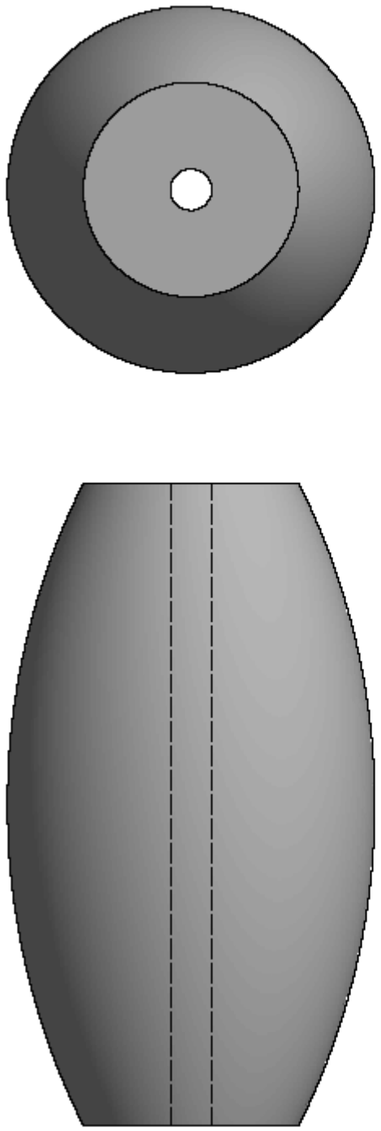
Concepto	Total
Presupuesto de materiales	88,93
Costes del desarrollo	6330
Coste total del proyecto	**6418,93**

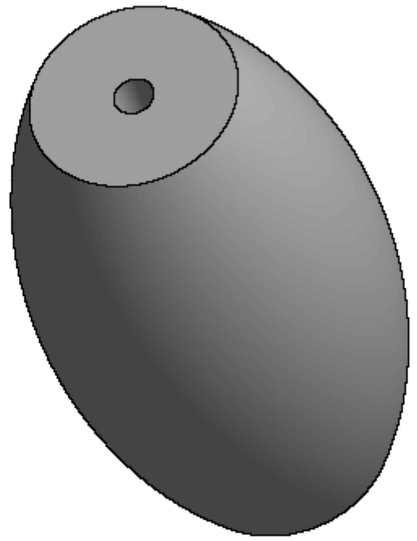
Tabla 30. Coste total del proyecto

Capítulo VI. Anexos

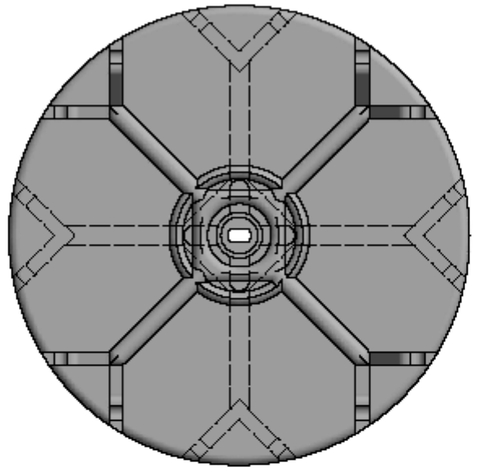
En los próximos apartados se presentará la documentación que se ha considerado relevante para documentar el proyecto. En concreto, se incluirán los planos en los que se muestran las piezas diseñadas en este trabajo. También se expondrán los principales códigos que se han desarrollado para las distintas funcionalidades del prototipo. Finalmente se exhibirán las hojas de datos de los componentes y materiales utilizados en este Trabajo de Fin de Grado.

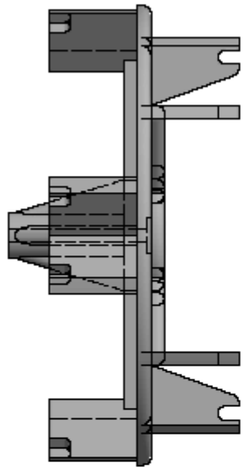
6.1. Planos de las piezas modeladas en 3D

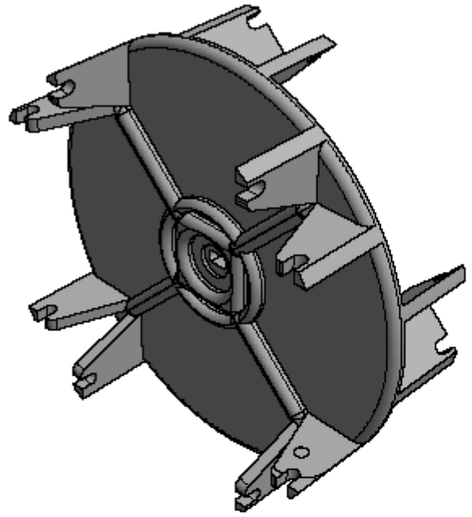


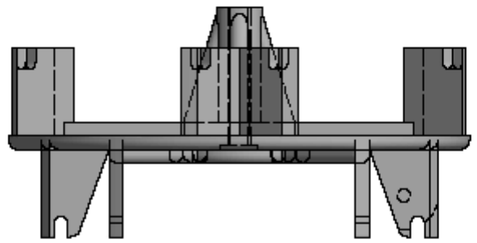


TRABAJO DE FIN DE GRADO				
	Fecha	Autor	Universidad de La Laguna	ESCUELA SUPERIOR DE INGENIERÍA Y TECNOLOGÍA Ingeniería Electrónica Industrial y Automática Universidad de La Laguna
Dibujado	06/01/19	Alberto Díaz		
Comprobado				
Id. s. normas	UNE-EN-DIN			
ESCALA: 3:1	Rodillo		N° P. : 1 Nom. Arch:	

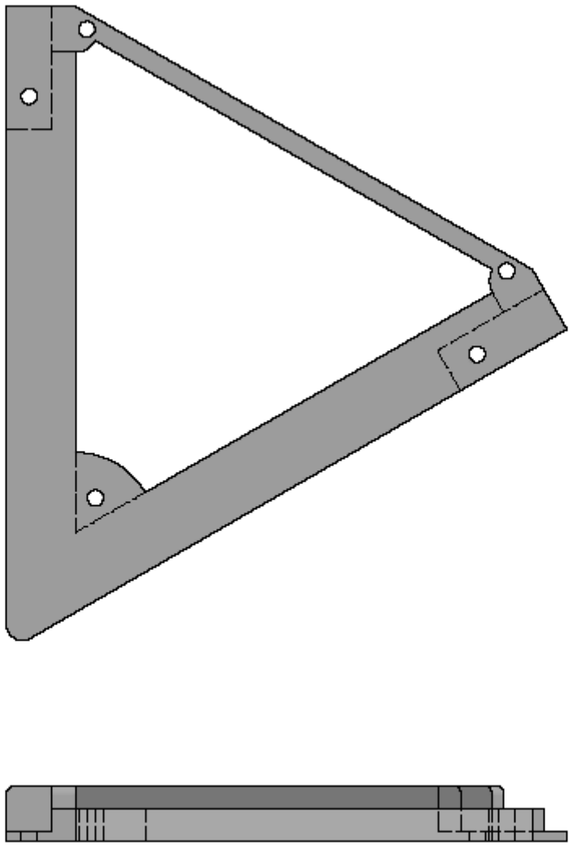


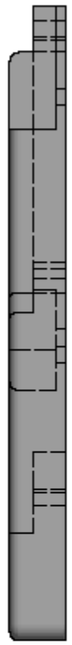


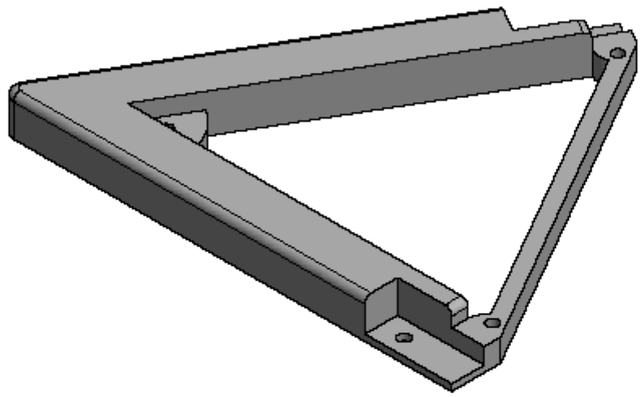




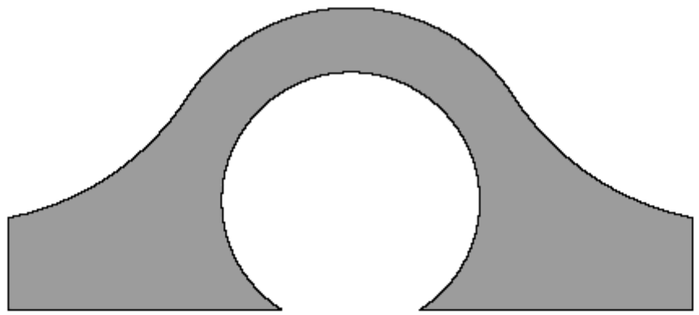
TRABAJO DE FIN DE GRADO				
	Fecha	Autor	Universidad de La Laguna	ESCUELA SUPERIOR DE INGENIERÍA Y TECNOLOGÍA Ingeniería Electrónica Industrial y Automática Universidad de La Laguna
Dibujado	06/01/19	Alberto Díaz		
Comprobado				
Id. s. normas	UNE-EN-DIN			
ESCALA: 1:1	Soporte de rodillos		N° P. : 2 Nom.Arch:	

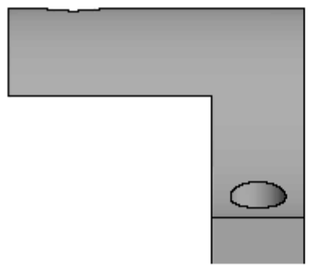


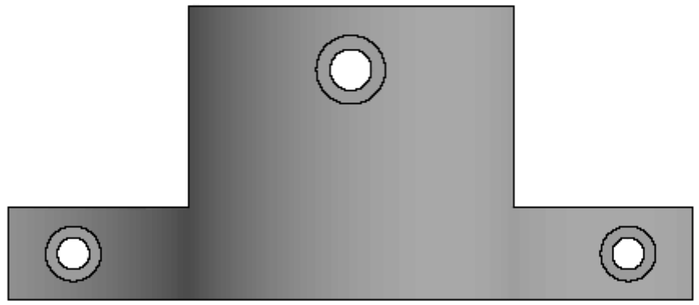


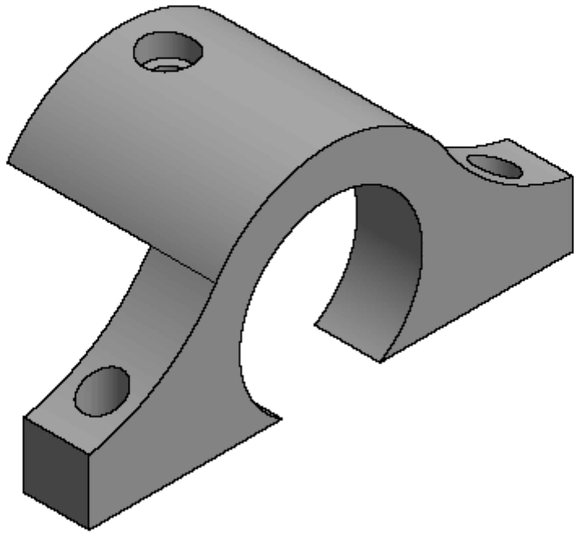


TRABAJO DE FIN DE GRADO				
	Fecha	Autor	Universidad de La Laguna	ESCUELA SUPERIOR DE INGENIERÍA Y TECNOLOGÍA Ingeniería Electrónica Industrial y Automática Universidad de La Laguna
Dibujado	9/06/19	Alberto Díaz		
Comprobado				
Id. s. normas	UNE-EN-DIN			
ESCALA: 1:1	Vértice de la estructura triangular			N° P. : 3 Nom. Arch:

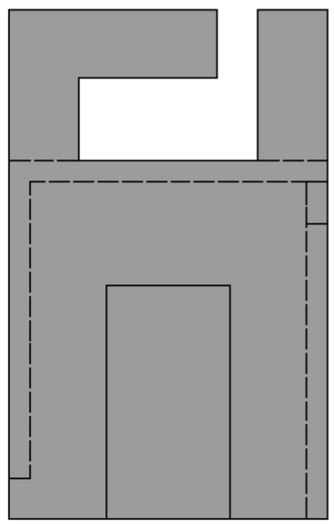


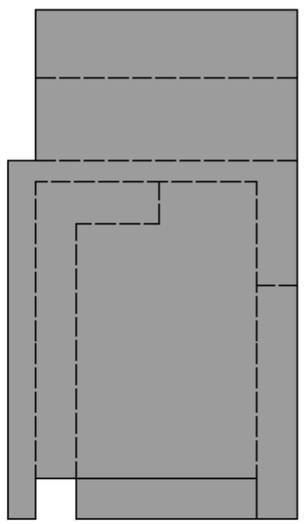


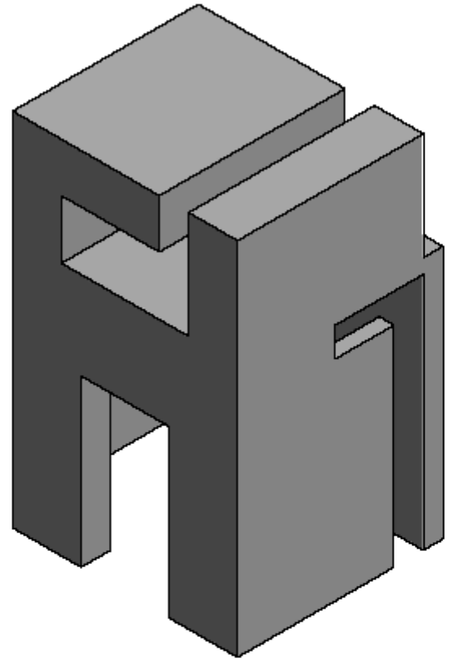


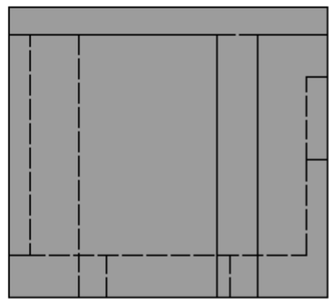


TRABAJO DE FIN DE GRADO				
	Fecha	Autor	Universidad de La Laguna	ESCUELA SUPERIOR DE INGENIERÍA Y TECNOLOGÍA Ingeniería Electrónica Industrial y Automática Universidad de La Laguna
Dibujado	09/06/19	Alberto Díaz		
Comprobado				
Id. s. normas	UNE-EN-DIN			
ESCALA: 2:1	Soporte para el motor		N° P. : 4 Nom.Arch:	



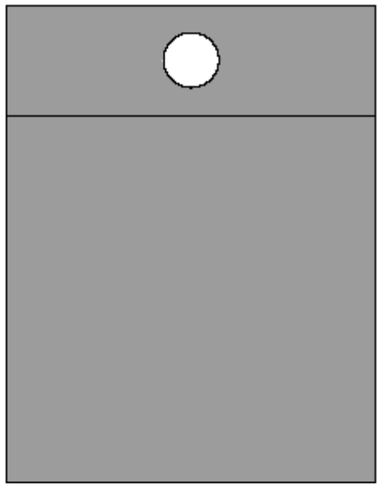


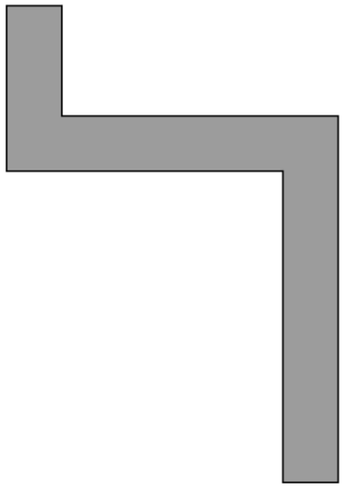


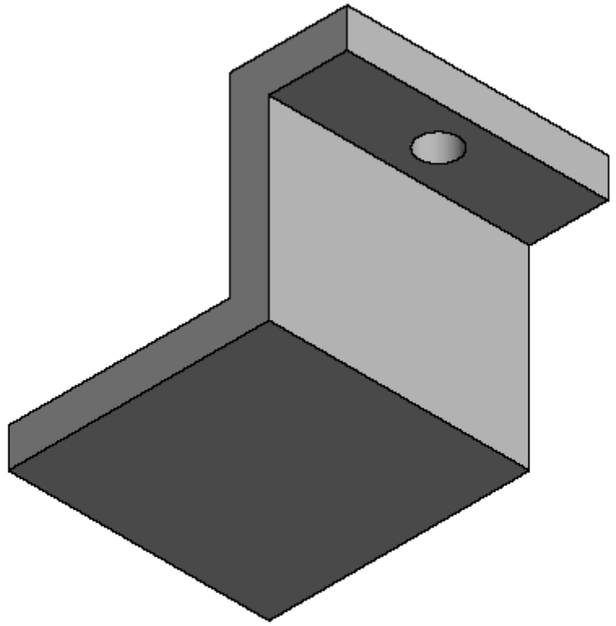


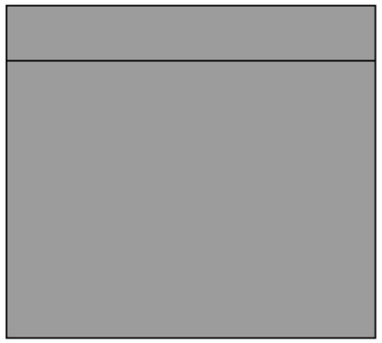
	Fecha	Autor		
Dibujado	09/08/19	Alberto Díaz	Universidad de La Laguna	ESCUELA SUPERIOR DE INGENIERÍA Y TECNOLOGÍA Ingeniería Electrónica Industrial y Automática Universidad de La Laguna
Comprobado				
Id. s. normas	UNE-EN-DIN			
ESCALA: 3:1	Soporte de interruptor y ESP-01			Nº P. : 5 Nom. Arch:

CREADO CON UNA VERSION PARA ESTUDIANTES DE AUTODESK



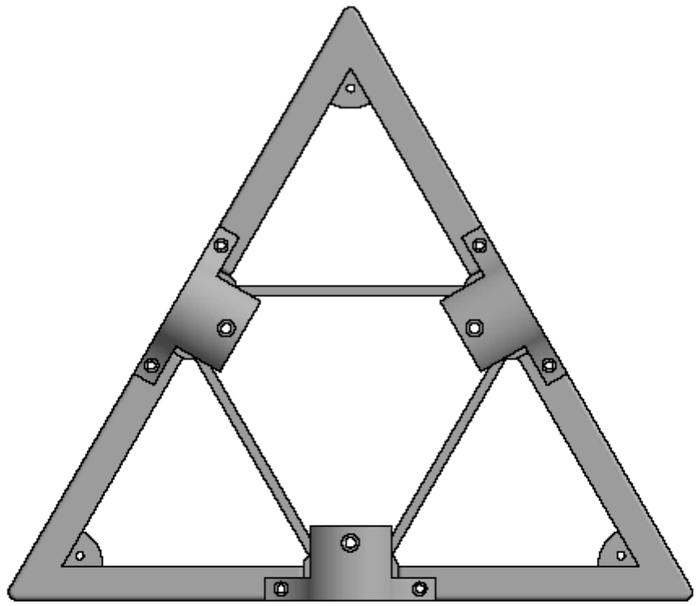


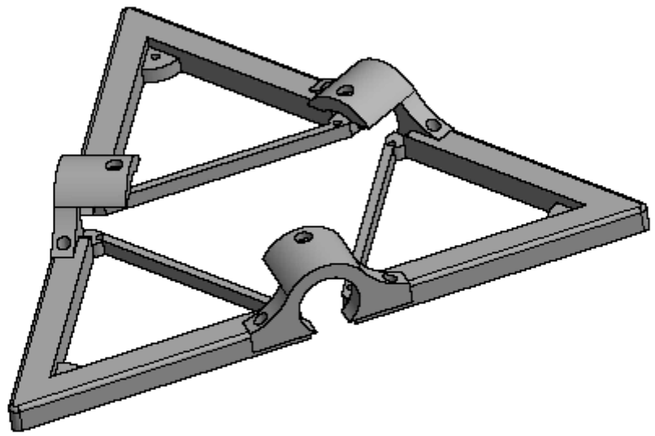


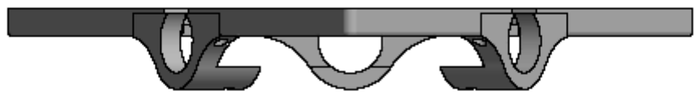


TRABAJO DE FIN DE GRADO				
	Fecha	Autor	Universidad de La Laguna	ESCUELA SUPERIOR DE INGENIERÍA Y TECNOLOGÍA
Dibujado	09/08/19	Alberto Díaz		Ingeniería Electrónica Industrial y Automática
Comprobado				Universidad de La Laguna
Id. s. normas	UNE-EN-DIN			
ESCALA: 4:1	Soporte para la batería		Nº P. : 6	Nom. Arch:

CREADO CON UNA VERSION PARA ESTUDIANTES DE AUTODESK







TRABAJO DE FIN DE GRADO				
	Fecha	Autor	Universidad de La Laguna	ESCUELA SUPERIOR DE INGENIERÍA Y TECNOLOGÍA
Dibujado	09/06/19	Alberto Díaz		Ingeniería Electrónica Industrial y Automática
Comprobado				Universidad de La Laguna
Id. s. normas	UNE-EN-DIN			
ESCALA: 1:2	Ensamblaje de la estructura		N° P. : 7	
			Nom. Arch:	

6.2. Códigos

En este apartado se expondrán los códigos que se han elaborado para los distintos sistemas de control del robot. Del mismo modo, se incluirán los ficheros que han sido necesarios para el sistema implementado en ROS en el que se obtienen las velocidades que se deben asignar a los motores cuando el mensaje correspondiente llegue al microcontrolador.

De forma adicional a lo anterior, se ofrece un enlace a la web GitHub en el que se ha creado un repositorio con el paquete de ROS utilizado, y una carpeta en la que se incluyen los códigos de Arduino empleados en este TFG. El enlace es el siguiente:

https://github.com/albertodrguez/Ros_Arduino_Bridge_Omnidireccional.git

6.2.1. Código del control mediante comunicación serial

```
// Código para el control de los motores mediante identificadores enviados a
través del monitor serial
// Autor: Alberto Díaz Rodríguez
// Fecha: 31/10/18
// Universidad de La Laguna


// El código de números será el siguiente:
// -IDENTIFICADOR --------MOTORES ACTIVADOS -------- SENTIDO DE GIRO-
//                        M1      M2      M3         M1      M2      M3
//        0               NO      NO      NO         -       -       -
//        1               SI      SI      SI         D       D       D
//        2               SI      SI      SI         I       I       I
//        3               SI      SI      NO         D       I       -
//        4               SI      NO      SI         D       -       I
//        5               NO      SI      SI         -       D       I
//        6               SI      SI      NO         I       D       -
//        7               SI      NO      SI         I       -       D
//        8               NO      SI      SI         -       I       D
//-----------------------------------------------------------------------


// Nombres de los pines
const int M11 = 2;
const int M12 = 3;
const int M13 = 4;
const int M14 = 5;
```

```
const int M21 = 6;
const int M22 = 7;
const int M23 = 8;
const int M24 = 9;

const int M31 = 10;
const int M32 = 11;
const int M33 = 12;
const int M34 = 13;

int identificador = 0; // Variable que almacenará el número de movimiento que
se debe realizar.
int valorDelay = 2; // Valor que determina la velocidad de giro de los
motores.

// Variables para el movimiento de los motores
int Secuencia_Actual_A = 0;
int Secuencia_Actual_B = 0;
int Secuencia_Actual_C = 0;



// Configuración inicial
void setup() {
  // Se establecen los pines como salidas.
  pinMode(M11, OUTPUT);
  pinMode(M12, OUTPUT);
  pinMode(M13, OUTPUT);
  pinMode(M14, OUTPUT);
  pinMode(M21, OUTPUT);
  pinMode(M22, OUTPUT);
  pinMode(M23, OUTPUT);
  pinMode(M24, OUTPUT);
  pinMode(M31, OUTPUT);
  pinMode(M32, OUTPUT);
  pinMode(M33, OUTPUT);
  pinMode(M34, OUTPUT);

  // Se inicia la comunicación serial.
  Serial.begin(9600);
```

```
  delay(2000);
}



// Bucle principal
void loop() {


  if (Serial.available() > 0){
      String str = Serial.readStringUntil('\n');
      identificador = str.toInt();
      Serial.println(identificador);
   }


  // Se llama a la función que controla el movimiento de los motores, una vez
para cada etapa de la secuencia
  Secuencia_Actual_A = 0;
  Secuencia_Actual_B = 0;
  Secuencia_Actual_C = 0;


  for(int i = 0; i < 4; i++){
    moverMotores(identificador);
    delay(valorDelay);
  }


}



// Función para el movimiento de los motores
void moverMotores(int identificador){


  // Motor 1 (M1)


  // Los identificadores que indican movimiento positivo de M1
  if(identificador == 1 || identificador == 3|| identificador == 4){
    Secuencia_Actual_A +=1;
  }
  // Los identificadores que indican movimiento negativo de M1
  if(identificador == 2 || identificador == 6 || identificador == 7){
```

```
  Secuencia_Actual_A -=1;
}

if (Secuencia_Actual_A == 4){
   Secuencia_Actual_A = 0;
}
if (Secuencia_Actual_A == -1){
   Secuencia_Actual_A = 3;
}

switch(Secuencia_Actual_A) {
case 0:
    digitalWrite(M11, LOW);
    digitalWrite(M12, LOW);
    digitalWrite(M13, HIGH);
    digitalWrite(M14, HIGH);
    break;
case 1:
    digitalWrite(M11, LOW);
    digitalWrite(M12, HIGH);
    digitalWrite(M13, HIGH);
    digitalWrite(M14, LOW);
    break;
case 2:
    digitalWrite(M11, HIGH);
    digitalWrite(M12, HIGH);
    digitalWrite(M13, LOW);
    digitalWrite(M14, LOW);
    break;
case 3:
    digitalWrite(M11, HIGH);
    digitalWrite(M12, LOW);
    digitalWrite(M13, LOW);
    digitalWrite(M14, HIGH);
    break;
default:
    digitalWrite(M11, LOW);
    digitalWrite(M12, LOW);
```

```
    digitalWrite(M13, LOW);
    digitalWrite(M14, LOW);
    break;

}

// Motor 2 (M2)

// Los identificadores que indican movimiento positivo de M2
if(identificador == 1 || identificador == 5|| identificador == 6){
  Secuencia_Actual_B +=1;
}
// Los identificadores que indican movimiento negativo de M2
if(identificador == 2 || identificador == 3 || identificador == 8){
  Secuencia_Actual_B -=1;
}

if (Secuencia_Actual_B == 4){
   Secuencia_Actual_B = 0;
}
if (Secuencia_Actual_B == -1){
   Secuencia_Actual_B = 3;
}

switch(Secuencia_Actual_B) {
case 0:
    digitalWrite(M21, LOW);
    digitalWrite(M22, LOW);
    digitalWrite(M23, HIGH);
    digitalWrite(M24, HIGH);
    break;
case 1:
    digitalWrite(M21, LOW);
    digitalWrite(M22, HIGH);
    digitalWrite(M23, HIGH);
    digitalWrite(M24, LOW);
    break;
case 2:
```

```
    digitalWrite(M21, HIGH);
    digitalWrite(M22, HIGH);
    digitalWrite(M23, LOW);
    digitalWrite(M24, LOW);
    break;
case 3:
    digitalWrite(M21, HIGH);
    digitalWrite(M22, LOW);
    digitalWrite(M23, LOW);
    digitalWrite(M24, HIGH);
    break;
default:
    digitalWrite(M21, LOW);
    digitalWrite(M22, LOW);
    digitalWrite(M23, LOW);
    digitalWrite(M24, LOW);
    break;
}

// Motor 3 (M3)

// Los identificadores que indican movimiento positivo de M3
if(identificador == 1 || identificador == 7|| identificador == 8){
  Secuencia_Actual_C +=1;
}
// Los identificadores que indican movimiento negativo de M3
if(identificador == 2 || identificador == 4 || identificador == 5){
  Secuencia_Actual_C -=1;
}

if (Secuencia_Actual_C == 4){
   Secuencia_Actual_C = 0;
}
if (Secuencia_Actual_C == -1){
   Secuencia_Actual_C = 3;
}

switch(Secuencia_Actual_C) {
```

```
  case 0:
      digitalWrite(M31, LOW);
      digitalWrite(M32, LOW);
      digitalWrite(M33, HIGH);
      digitalWrite(M34, HIGH);
      break;
  case 1:
      digitalWrite(M31, LOW);
      digitalWrite(M32, HIGH);
      digitalWrite(M33, HIGH);
      digitalWrite(M34, LOW);
      break;
  case 2:
      digitalWrite(M31, HIGH);
      digitalWrite(M32, HIGH);
      digitalWrite(M33, LOW);
      digitalWrite(M34, LOW);
      break;
  case 3:
      digitalWrite(M31, HIGH);
      digitalWrite(M32, LOW);
      digitalWrite(M33, LOW);
      digitalWrite(M34, HIGH);
      break;
  default:
      digitalWrite(M31, LOW);
      digitalWrite(M32, LOW);
      digitalWrite(M33, LOW);
      digitalWrite(M34, LOW);
      break;

  }

}
```

6.2.2. Código del control mediante servidor web. Arduino WeMos

```
// Implementación de un servidor web utilizando la placa Arduino WeMos D1

// Autor: Alberto Díaz Rodríguez

// Fecha: 9/11/18

// Universidad de La Laguna


// Librerías

#include <ESP8266WiFi.h>

#include <ESP8266mDNS.h>

#include <WiFiUdp.h>


// Variables


int identificador = 0; // Variable que almacenará el número de movimiento que
se debe realizar.

int valorDelay = 2; // Valor que determina la velocidad de giro de los
motores, en este caso es una simulación.


// Variables que almacenan los pasos que da cada motor (el signo indica el
sentido en el que ha girado).

float pasosM1 = 0;

float pasosM2 = 0;

float pasosM3 = 0;


// Variables que almacenan los grados que ha girado cada motor (el signo
indica el sentido en el que ha girado). Se trabaja en grados.

float gradosM1 = 0;

float gradosM2 = 0;

float gradosM3 = 0;


// Se introducen los datos del punto de acceso

const char* ssid = "WifiMovil";

const char* password = "12345678";


String orden =  "";


WiFiServer server(80); //Se crea el servidor en el puerto 80
```

```
// Configuración inicial
void setup() {

  // Se inicia la cominicación serial
  Serial.begin(74880);

  // Se realiza la conexión a la red WIFI
  Serial.print("\nConectando a la red: ");
  Serial.println(ssid);

  WiFi.mode(WIFI_STA);  // ESP8266 en modo estación
  WiFi.begin(ssid, password); // ssid y contraseña de la red

  while (WiFi.status() != WL_CONNECTED) { // Se espera hasta que se haya
establecido la conexión
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi conectada"); // Se ha realizado la conexión
correctamente

  // Se inicia el servidor
  server.begin();
  Serial.println("Servidor creado.");

  // Se muestra la dirección IP a la que hay que conectarse para interactuar
con el software
  Serial.print("URL : http://");
  Serial.println(WiFi.localIP());

  delay(6000); // Tiempo que se ofrece para introducir la IP en un navegador
en milisegundos
}


// Bucle principal
void loop() {
```

```
// Se busca un cliente activo
WiFiClient client = server.available();

// Se llama a la función que simula el movimiento de los motores
for(int i = 0; i < 4; i++){
  moverMotores(identificador);
  delay(valorDelay);
}

if (!client) {
  return;
}


// Se espera la respuesta del cliente
while(!client.available()){
  delay(1);
}

// Se lee la orden enviada por el cliente
orden = client.readStringUntil('\r');
client.flush();

// Se averigua el identificador enviado

if (orden.indexOf("/ORDEN=0") != -1) {
  identificador = 0;
}
if (orden.indexOf("/ORDEN=1") != -1) {
  identificador = 1;
}
if (orden.indexOf("/ORDEN=2") != -1) {
  identificador = 2;
}
if (orden.indexOf("/ORDEN=3") != -1) {
  identificador = 3;
}
if (orden.indexOf("/ORDEN=4") != -1) {
```

```
  identificador = 4;
}
if (orden.indexOf("/ORDEN=5") != -1) {
  identificador = 5;
}
if (orden.indexOf("/ORDEN=6") != -1) {
  identificador = 6;
}
if (orden.indexOf("/ORDEN=7") != -1) {
  identificador = 7;
}
if (orden.indexOf("/ORDEN=8") != -1) {
  identificador = 8;
}

// Configuración servidor HTTP
client.println("HTTP/1.1 200 OK");
client.println("Content-Type: text/html");
client.println("");
client.println("<!DOCTYPE HTML>");
client.println("<html>");


// Se muestra al cliente el identificador actual

switch(identificador) {
  case 0: client.print("Identificador actual: 0");break;
  case 1: client.print("Identificador actual: 1");break;
  case 2: client.print("Identificador actual: 2");break;
  case 3: client.print("Identificador actual: 3");break;
  case 4: client.print("Identificador actual: 4");break;
  case 5: client.print("Identificador actual: 5");break;
  case 6: client.print("Identificador actual: 6");break;
  case 7: client.print("Identificador actual: 7");break;
  case 8: client.print("Identificador actual: 8");break;
}

// Se muestran las opciones que puede escoger el cliente para enviar órdenes
```

```
  client.println("<br><br>");

  client.println("Haz click <a href=\"/ORDEN=0\">aqui</a> para enviar un
0<br>");

  client.println("Haz click <a href=\"/ORDEN=1\">aqui</a> para enviar un
1<br>");

  client.println("Haz click <a href=\"/ORDEN=2\">aqui</a> para enviar un
2<br>");

  client.println("Haz click <a href=\"/ORDEN=3\">aqui</a> para enviar un
3<br>");

  client.println("Haz click <a href=\"/ORDEN=4\">aqui</a> para enviar un
4<br>");

  client.println("Haz click <a href=\"/ORDEN=5\">aqui</a> para enviar un
5<br>");

  client.println("Haz click <a href=\"/ORDEN=6\">aqui</a> para enviar un
6<br>");

  client.println("Haz click <a href=\"/ORDEN=7\">aqui</a> para enviar un
7<br>");

  client.println("Haz click <a href=\"/ORDEN=8\">aqui</a> para enviar un
8<br>");


  // Se muestran también los grados girados

  // La relación entre los pasos y los grados girados es: 512 pasos son 360
grados. Por tanto:


  gradosM1 = float(pasosM1 * 360 / 4 * 512);

  gradosM2 = float(pasosM2 * 360 / 4 * 512);

  gradosM3 = float(pasosM3 * 360 / 4 * 512);


  client.println("<br>");

  client.println("<br>");

  client.println("Los grados girados por cada motor son:<br>");

  client.print("GradosM1: ");

  client.println(gradosM1);

  client.println("<br>");

  client.print("GradosM2: ");

  client.println(gradosM2);

  client.println("<br>");

  client.print("GradosM3: ");

  client.println(gradosM3);

  client.println("</html>");

  Serial.println("");


}
```

```
// Función para la simulación del movimiento de los motores
void moverMotores(int identificador){

  // Motor 1 (M1)

  // Los identificadores que indican movimiento positivo de M1
  if(identificador == 1 || identificador == 3|| identificador == 4){
    pasosM1 ++;
  }
  // Los identificadores que indican movimiento negativo de M1
  if(identificador == 2 || identificador == 6 || identificador == 7){
    pasosM1 --;
  }

  // Motor 2 (M2)

  // Los identificadores que indican movimiento positivo de M2
  if(identificador == 1 || identificador == 5|| identificador == 6){
    pasosM2 ++;
  }
  // Los identificadores que indican movimiento negativo de M2
  if(identificador == 2 || identificador == 3 || identificador == 8){
    pasosM2 --;
  }

  // Motor 3 (M3)

  // Los identificadores que indican movimiento positivo de M3
  if(identificador == 1 || identificador == 7|| identificador == 8){
    pasosM3 ++;
  }
  // Los identificadores que indican movimiento negativo de M3
  if(identificador == 2 || identificador == 4 || identificador == 5){
    pasosM3 --;
  }

}
```

6.2.3. Código del control mediante servidor web. Arduino MEGA

```
// Control del movimiento del robot mediante servidor web implementado en
Arduino MEGA con módulo ESP-01

// Autor: Alberto Díaz Rodríguez

// Fecha: 11/02/19

// Universidad de La Laguna


// El código de números será el siguiente:

// -IDENTIFICADOR --------MOTORES ACTIVADOS -------- SENTIDO DE GIRO-

//                        M1      M2      M3          M1      M2      M3

//        0               NO      NO      NO          -       -       -

//        1               SI      SI      SI          D       D       D

//        2               SI      SI      SI          I       I       I

//        3               SI      SI      NO          D       I       -

//        4               SI      NO      SI          D       -       I

//        5               NO      SI      SI          -       D       I

//        6               SI      SI      NO          I       D       -

//        7               SI      NO      SI          I       -       D

//        8               NO      SI      SI          -       I       D

//---------------------------------------------------------------------


// Librería

#include <LiquidCrystal_I2C.h> // Libreria LCD_I2C


// Nombres de los pines

const int M11 = 2;

const int M12 = 3;

const int M13 = 4;

const int M14 = 5;


const int M21 = 6;

const int M22 = 7;

const int M23 = 8;

const int M24 = 9;


const int M31 = 10;

const int M32 = 11;
```

```
const int M33 = 12;

const int M34 = 13;



// Otras variables

bool primeraEjecucion = true; // Sirve para que en la primera ejecución del
programa se envie la información al cliente.

String URL = "";


int identificador = 0; // Variable que almacenará el número de movimiento que
se debe realizar.

int valorDelay = 2; // Valor que determina la velocidad de giro de los
motores.


int Secuencia_Actual_A = 0;

int Secuencia_Actual_B = 0;

int Secuencia_Actual_C = 0;


int mostrar = 0; // Para que solo muestre 1 vez el valor de los grados girados


// Configuración inicial

void setup(){

  // Se establece la dirección para la pantalla LCD con I2C, y se enciende y
limpia la pantalla.

  LiquidCrystal_I2C lcd(0x27,16,2);

  lcd.init();

  lcd.backlight();

  lcd.clear();


  // Se establecen los pines como salidas.

  pinMode(M11, OUTPUT);

  pinMode(M12, OUTPUT);

  pinMode(M13, OUTPUT);

  pinMode(M14, OUTPUT);

  pinMode(M21, OUTPUT);

  pinMode(M22, OUTPUT);

  pinMode(M23, OUTPUT);

  pinMode(M24, OUTPUT);
```

```
  pinMode(M31, OUTPUT);
  pinMode(M32, OUTPUT);
  pinMode(M33, OUTPUT);
  pinMode(M34, OUTPUT);

  delay(1000);
  Serial3.begin(115200);

 // Se envian todos los comandos AT necesarios para configurar el módulo ESP-
01.
 sendData("AT+CWMODE=3\r\n",1000); // ESP-01 en modo (3): enviar y recibir
datos.
 sendData("AT+CIPMUX=1\r\n",1000); // Se configura para múltiples conexiones,
aunque en principio solo se producirá una.
 sendData("AT+CIPSERVER=1,80\r\n",1000); // Se activa el servidor en el puerto
80.
 delay(1000);
 sendData("AT+CWJAP=\"WifiMovil\",\"12345678\"\r\n",1000);//Se indica el SSID
y la contraseña de la red WIFI para establecer la conexión.
 delay(8000);
 sendDataURL("AT+CIFSR\r\n",1000); // Se obtiene la dirección IP a la que
conectarse.
 delay(1000);
 // Se muestra la dirección IP por la pantalla LCD para que el usuario se
conecte.
 lcd.setCursor(0,0);
 lcd.print("URL:");
 lcd.setCursor (0,1);
 lcd.print(URL);
 lcd.display();
 delay(500);
}

// Bucle principal
void loop(){

  if(Serial3.available()){
```

```
    if(primeraEjecucion == true){ // La primera vez que se reciben datos del
ESP-01 se muestra la web


      LiquidCrystal_I2C lcd(0x27,16,2);

      lcd.init();

      lcd.noDisplay(); // Se apaga la pantalla una vez se ha conectado un
cliente.


      if(Serial3.find("+IPD,")){

        delay(1000); // Este delay parece que se puede quitar.


        int connectionId = Serial3.read() - 48; // Se resta 48 porque la
función read() devuelve el código ASCII


        // Se prepara el servidor HTTP

        String webpage = "<head></head>"; // Cabecera de la página.

        webpage+="<h1 align=\"center\">Control del robot
omnidireccional</h1>";

        webpage+="<br />";

        webpage+="<h2 align=\"center\"><button
onclick=location=\"/ORDEN=2\">Enviar un
2</button>             &nb
sp  ";

        webpage+="<button onclick=location=\"/ORDEN=1\">Enviar un
1</button></h2>";

        webpage+="<h2 align=\"center\"><button
onclick=location=\"/ORDEN=6\">Enviar un 6</button>    ";

        webpage+="<button onclick=location=\"/ORDEN=3\">Enviar un
3</button></h2>";

        webpage+="<h2 align=\"center\"><button
onclick=location=\"/ORDEN=0\">Enviar un 0</button></h2>";

        webpage+="<h2 align=\"center\"><button
onclick=location=\"/ORDEN=8\">Enviar un 8</button>    ";

        webpage+="<button onclick=location=\"/ORDEN=5\">Enviar un
5</button>        ";

        webpage+="<button onclick=location=\"/ORDEN=7\">Enviar un
7</button>    ";

        webpage+="<button onclick=location=\"/ORDEN=4\">Enviar un
4</button></h2>";

        webpage+="<br/>";


        sendData("AT+CIPSEND=" + String(connectionId) + "," + webpage.length()
+ "\r\n", 500);

        sendData(webpage, 1000);
```

```
        sendData("AT+CIPCLOSE=" + String(connectionId) + "\r\n", 1000);


        primeraEjecucion = false; // Cuando ya se ha mostrado la web se pasa a
comprobar las órdenes que se mandan desde la misma.
     }


    }else{
      // Tras la primera ejecución se intenta recibir información procedente
del ESP8266
      getData("AT+CIPRECVMODE=1\r\n", 50);
    }
  }


  // A partir de la orden recogida se activan los motores según proceda


  Secuencia_Actual_A = 0;
  Secuencia_Actual_B = 0;
  Secuencia_Actual_C = 0;


  for(int i = 0; i < 4; i++){
    moverMotores(identificador);
    delay(valorDelay);
  }
}


// Función para enviar datos a la URL
String sendData(String command, const int timeout){
    String response = "";
    Serial3.print(command);
    long int time = millis();
    while( (time+timeout) > millis()){
      while(Serial3.available()){
        char c = Serial3.read();
        response+=c;
      }
    }
    return response;
}
```

```
// Función para obtener la URL
String sendDataURL(String command, const int timeout){
    String response = "";
    Serial3.print(command);
    long int time = millis();
    while( (time+timeout) > millis()){
      while(Serial3.available()){
        char c = Serial3.read();
        response+=c;
      }
    }

    for(int j = 121; j < 135; j++){
      URL += response[j];
    }

    if (response[135] != 34){
      URL += response[135];
    }

    return response;
}

// Función para recibir datos de la URL
void getData(String command, const int timeout){
    String response = "";
    Serial3.print(command);
    long int time = millis();
    while( (time+timeout) > millis()){
      while(Serial3.available()){
        char c = Serial3.read();
        response+=c;
      }
    }

  // Se identifica la orden recibida
```

```
  if (response.indexOf("/ORDEN=0") != -1) {
    identificador = 0;
  }
  if (response.indexOf("/ORDEN=1") != -1) {
   identificador = 1;
  }
  if (response.indexOf("/ORDEN=2") != -1) {
    identificador = 2;
  }
  if (response.indexOf("/ORDEN=3") != -1) {
    identificador = 3;
  }
  if (response.indexOf("/ORDEN=4") != -1) {
    identificador =  4;
  }
  if (response.indexOf("/ORDEN=5") != -1) {
    identificador = 5;
  }
  if (response.indexOf("/ORDEN=6") != -1) {
    identificador = 6;
  }
  if (response.indexOf("/ORDEN=7") != -1) {
    identificador = 7;
  }
  if (response.indexOf("/ORDEN=8") != -1) {
    identificador = 8;
  }

}


// Función para el movimiento de los motores
void moverMotores(int identificador){

  // Motor 1 (M1)

  // Los identificadores que indican movimiento positivo de M1
```

```
if(identificador == 1 || identificador == 3|| identificador == 4){
  Secuencia_Actual_A +=1;
}
// Los identificadores que indican movimiento negativo de M1
if(identificador == 2 || identificador == 6 || identificador == 7){
  Secuencia_Actual_A -=1;
}

if (Secuencia_Actual_A == 4){
   Secuencia_Actual_A = 0;
}
if (Secuencia_Actual_A == -1){
   Secuencia_Actual_A = 3;
}

switch(Secuencia_Actual_A) {
case 0:
    digitalWrite(M11, LOW);
    digitalWrite(M12, LOW);
    digitalWrite(M13, HIGH);
    digitalWrite(M14, HIGH);
    break;
case 1:
    digitalWrite(M11, LOW);
    digitalWrite(M12, HIGH);
    digitalWrite(M13, HIGH);
    digitalWrite(M14, LOW);
    break;
case 2:
    digitalWrite(M11, HIGH);
    digitalWrite(M12, HIGH);
    digitalWrite(M13, LOW);
    digitalWrite(M14, LOW);
    break;
case 3:
    digitalWrite(M11, HIGH);
    digitalWrite(M12, LOW);
```

```
    digitalWrite(M13, LOW);
    digitalWrite(M14, HIGH);
    break;
default:
    digitalWrite(M11, LOW);
    digitalWrite(M12, LOW);
    digitalWrite(M13, LOW);
    digitalWrite(M14, LOW);
    break;

}


// Motor 2 (M2)

// Los identificadores que indican movimiento positivo de M2
if(identificador == 1 || identificador == 5|| identificador == 6){
  Secuencia_Actual_B +=1;
}
// Los identificadores que indican movimiento negativo de M2
if(identificador == 2 || identificador == 3 || identificador == 8){
  Secuencia_Actual_B -=1;
}

if (Secuencia_Actual_B == 4){
   Secuencia_Actual_B = 0;
}
if (Secuencia_Actual_B == -1){
   Secuencia_Actual_B = 3;
}

switch(Secuencia_Actual_B) {
case 0:
    digitalWrite(M21, LOW);
    digitalWrite(M22, LOW);
    digitalWrite(M23, HIGH);
    digitalWrite(M24, HIGH);
```

```
    break;
case 1:
    digitalWrite(M21, LOW);
    digitalWrite(M22, HIGH);
    digitalWrite(M23, HIGH);
    digitalWrite(M24, LOW);
    break;
case 2:
    digitalWrite(M21, HIGH);
    digitalWrite(M22, HIGH);
    digitalWrite(M23, LOW);
    digitalWrite(M24, LOW);
    break;
case 3:
    digitalWrite(M21, HIGH);
    digitalWrite(M22, LOW);
    digitalWrite(M23, LOW);
    digitalWrite(M24, HIGH);
    break;
default:
    digitalWrite(M21, LOW);
    digitalWrite(M22, LOW);
    digitalWrite(M23, LOW);
    digitalWrite(M24, LOW);
    break;

}

// Motor 3 (M3)

// Los identificadores que indican movimiento positivo de M3
if(identificador == 1 || identificador == 7|| identificador == 8){
  Secuencia_Actual_C +=1;
}
// Los identificadores que indican movimiento negativo de M3
if(identificador == 2 || identificador == 4 || identificador == 5){
  Secuencia_Actual_C -=1;
```

```
}

if (Secuencia_Actual_C == 4){
   Secuencia_Actual_C = 0;
}
if (Secuencia_Actual_C == -1){
   Secuencia_Actual_C = 3;
}

switch(Secuencia_Actual_C) {
case 0:
    digitalWrite(M31, LOW);
    digitalWrite(M32, LOW);
    digitalWrite(M33, HIGH);
    digitalWrite(M34, HIGH);
    break;
case 1:
    digitalWrite(M31, LOW);
    digitalWrite(M32, HIGH);
    digitalWrite(M33, HIGH);
    digitalWrite(M34, LOW);
    break;
case 2:
    digitalWrite(M31, HIGH);
    digitalWrite(M32, HIGH);
    digitalWrite(M33, LOW);
    digitalWrite(M34, LOW);
    break;
case 3:
    digitalWrite(M31, HIGH);
    digitalWrite(M32, LOW);
    digitalWrite(M33, LOW);
    digitalWrite(M34, HIGH);
    break;
default:
    digitalWrite(M31, LOW);
    digitalWrite(M32, LOW);
```

```
        digitalWrite(M33, LOW);
        digitalWrite(M34, LOW);
        break;

    }
}
```

6.2.4. Código del control mediante Ros_Arduino_Bridge

En este caso, se ha expuesto en la memoria que se realizó un diseño utilizando el protocolo TCP/IP, y otro para el protocolo UDP. No obstante, el primero no se terminó utilizando, por lo que a continuación se expone el código en el que se emplea el protocolo UDP.

El modelo lo componen tres ficheros. El primero de ellos es el archivo de comandos.

```
// Se definen una serie de comandos identificador por un único carácter


#ifndef COMMANDS_H
#define COMMANDS_H


#define ANALOG_READ    'a'
#define GET_BAUDRATE   'b'
#define PIN_MODE       'c'
#define DIGITAL_READ   'd'
#define DIGITAL_WRITE  'w'
#define ANALOG_WRITE   'v'
#define MOTORS_SPEED   'x'
#define LEFT            0
#define RIGHT           1


#endif
```

El siguiente fichero es en el que se atiende a las interrupciones software.

```
// Código en el que se implementa el movimiento de los motores utilizando las
interrupciones software


// Nombres de los pines
const int M11 = 2;
const int M12 = 3;
const int M13 = 4;
const int M14 = 5;


const int M21 = 6;
const int M22 = 7;
const int M23 = 8;
const int M24 = 9;
```

```
const int M31 = 10;
const int M32 = 11;
const int M33 = 12;
const int M34 = 13;

// Función para deshabilitar el Timer3
void disableTimer3(){
  TCCR3A = 0;
  TCCR3B = 0;
}

// Configuración del Timer3
void setupTimer3_25mS(){
  // Se deshabilitan las interrupciones
  cli();

  disableTimer3();

  // Se ajusta el parámetro para el tiempo de cuenta deseado
  OCR3A = 575;

  // Se activa el modo CTC. WGM32 establece los valores de los bits CS10 y
CS12 para un preescalado del Timer3 de 1024
  TCCR3B = bit (WGM32) | bit (WGM32) | bit (CS30);

  // Se activa el temporizador
  TIMSK3 |= (1 << OCIE3A);

  // Se habilitan las interrupciones
  sei();
}

ISR(TIMER3_COMPA_vect){ // Cada vez que pasa el tiempo establecido se ejecuta
esta interrupción.


// Motor A
Interrupciones_A = Interrupciones_A -1;
```

```
if (Interrupciones_A <= abs(numero_cambio_secuencia_A)) {
  Interrupciones_A = 255;
  // Toca la siguiente secuencia

  if(numero_cambio_secuencia_A > 0){
    Secuencia_Actual_A +=1;
    avanceMotores[0] +=1;
  }
  if(numero_cambio_secuencia_A < 0){
    Secuencia_Actual_A -=1;
    avanceMotores[1] +=1;
  }

  if (Secuencia_Actual_A == 4){
     Secuencia_Actual_A = 0;
  }
  if (Secuencia_Actual_A == -1){
     Secuencia_Actual_A = 3;
  }


  switch(Secuencia_Actual_A) {
  case 3:
      digitalWrite(M11, LOW);
      digitalWrite(M12, LOW);
      digitalWrite(M13, HIGH);
      digitalWrite(M14, HIGH);
      break;
  case 2:
      digitalWrite(M11, LOW);
      digitalWrite(M12, HIGH);
      digitalWrite(M13, HIGH);
      digitalWrite(M14, LOW);
      break;
  case 1:
      digitalWrite(M11, HIGH);
      digitalWrite(M12, HIGH);
      digitalWrite(M13, LOW);
```

```
      digitalWrite(M14, LOW);
      break;
  case 0:
      digitalWrite(M11, HIGH);
      digitalWrite(M12, LOW);
      digitalWrite(M13, LOW);
      digitalWrite(M14, HIGH);
      break;
  }
}


// Motor B
Interrupciones_B = Interrupciones_B -1;

if (Interrupciones_B <= abs(numero_cambio_secuencia_B)) {
  Interrupciones_B = 255;
  // Toca la siguiente secuencia

  if(numero_cambio_secuencia_B > 0){
    Secuencia_Actual_B +=1;
    avanceMotores[2] +=1;
  }
  if(numero_cambio_secuencia_B < 0){
    Secuencia_Actual_B -=1;
    avanceMotores[3] +=1;
  }

  if (Secuencia_Actual_B == 4){
     Secuencia_Actual_B = 0;
  }
  if (Secuencia_Actual_B == -1){
     Secuencia_Actual_B = 3;
  }


  switch(Secuencia_Actual_B) {
  case 3:
```

```
      digitalWrite(M21, LOW);
      digitalWrite(M22, LOW);
      digitalWrite(M23, HIGH);
      digitalWrite(M24, HIGH);
      break;
  case 2:
      digitalWrite(M21, LOW);
      digitalWrite(M22, HIGH);
      digitalWrite(M23, HIGH);
      digitalWrite(M24, LOW);
      break;
  case 1:
      digitalWrite(M21, HIGH);
      digitalWrite(M22, HIGH);
      digitalWrite(M23, LOW);
      digitalWrite(M24, LOW);
      break;
  case 0:
      digitalWrite(M21, HIGH);
      digitalWrite(M22, LOW);
      digitalWrite(M23, LOW);
      digitalWrite(M24, HIGH);
      break;
  }
}

// Motor C
Interrupciones_C = Interrupciones_C -1;

if (Interrupciones_C <= abs(numero_cambio_secuencia_C)) {
  Interrupciones_C = 255;
  // Toca la siguiente secuencia

  if(numero_cambio_secuencia_C > 0){
    Secuencia_Actual_C +=1;
    avanceMotores[4] +=1;
  }
  if(numero_cambio_secuencia_C < 0){
```

```
  Secuencia_Actual_C -=1;
  avanceMotores[5] +=1;
}

if (Secuencia_Actual_C == 4){
   Secuencia_Actual_C = 0;
}
if (Secuencia_Actual_C == -1){
   Secuencia_Actual_C = 3;
}


switch(Secuencia_Actual_C) {
case 3:
    digitalWrite(M31, LOW);
    digitalWrite(M32, LOW);
    digitalWrite(M33, HIGH);
    digitalWrite(M34, HIGH);
    break;
case 2:
    digitalWrite(M31, LOW);
    digitalWrite(M32, HIGH);
    digitalWrite(M33, HIGH);
    digitalWrite(M34, LOW);
    break;
case 1:
    digitalWrite(M31, HIGH);
    digitalWrite(M32, HIGH);
    digitalWrite(M33, LOW);
    digitalWrite(M34, LOW);
    break;
case 0:
    digitalWrite(M31, HIGH);
    digitalWrite(M32, LOW);
    digitalWrite(M33, LOW);
    digitalWrite(M34, HIGH);
    break;
}
```

```
}
}
```

Por último, se incluye el fichero principal de este método de control.

```
/*********************************************************************
 *  ROSArduinoBridge

    A set of simple serial commands to control a differential drive
    robot and receive back sensor and odometry data. Default
    configuration assumes use of an Arduino Mega + Pololu motor
    controller shield + Robogaia Mega Encoder shield.  Edit the
    readEncoder() and setMotorSpeed() wrapper functions if using
    different motor controller or encoder method.

    Created for the Pi Robot Project: http://www.pirobot.org
    and the Home Brew Robotics Club (HBRC): http://hbrobotics.org

    Authors: Patrick Goebel, James Nugen

    Inspired and modeled after the ArbotiX driver by Michael Ferguson

    Software License Agreement (BSD License)

    Copyright (c) 2012, Patrick Goebel.
    All rights reserved.

    Redistribution and use in source and binary forms, with or without
    modification, are permitted provided that the following conditions
    are met:

     * Redistributions of source code must retain the above copyright
       notice, this list of conditions and the following disclaimer.
     * Redistributions in binary form must reproduce the above
       copyright notice, this list of conditions and the following
       disclaimer in the documentation and/or other materials provided
       with the distribution.

    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
    "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
```

```
    LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
    FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
    COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
    INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
    BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
    LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
    CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
    LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
    ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 *  POSSIBILITY OF SUCH DAMAGE.
 *********************************************************************/

// Modificaciones realizadas por: Alberto Díaz Rodríguez
// Porgrama para la recepción de las velocidades de los motores, cálculo de la odometría y envío de la pose
// Fecha 07/05/2019
// Universidad de La Laguna

// Velocidad de la comunicación en el puerto serial
#define BAUDRATE     57600

#include <LiquidCrystal_I2C.h> // Libreria LCD_I2C
LiquidCrystal_I2C lcd(0x27,16,2); // Dirección de la pantalla



// Se incluye el archivo con la definición de los comandos
#include "commands.h"

// Iniciación de las variables
// Variables para la recepción del mensaje de ROS
int arg = 0;
int index = 0;

// Almacenará el caracter que se esté identificando del mensaje recibido
char chr;

// Se almacena el comando identificado
char cmd;
```

```
// Arrays para almacenar los 3 argumentos recibidos
char argv1[16];
char argv2[16];
char argv3[16];

// Se convierten los argumentos a variables de tipo long
long arg1;
long arg2;
long arg3;

// String para la respuesta
String respuesta = "";

// Variables globales para la odometría del robot

// Vector que almacenará la pose del robot (x e y en centímetros y Theta en
grados).
float Pose[3] = {0, 0, 90};
int avanceMotores[6] = {0,0,0,0,0,0}; // Vector en el que se anotarán los
pasos que den los motores
int ciclos = 0;
float avance = (2 * PI * 31)/(512 * 4);
float rotacion = (avance * 360)/(6 * PI * 68.9);

// Variables globales para la función que activa el movimiento de los motores

// Motor 1
int Interrupciones_A = 255;
int numero_cambio_secuencia_A = 0;
int Secuencia_Actual_A = 0;

// Motor 2
int Interrupciones_B = 255;
int numero_cambio_secuencia_B = 0;
int Secuencia_Actual_B = 0;

// Motor 3
int Interrupciones_C = 255;
int numero_cambio_secuencia_C = 0;
```

```
int Secuencia_Actual_C = 0;


// Función para vaciar las variables en las que se almacenan los datos
recibidos
void resetCommand() {
  cmd = NULL;
  memset(argv1, 0, sizeof(argv1));
  memset(argv2, 0, sizeof(argv2));
  memset(argv3, 0, sizeof(argv3));
  arg1 = 0;
  arg2 = 0;
  arg3 = 0;
  arg = 0;
  index = 0;
}


// Función para la activación de la orden recibida
int runCommand() {
  int i = 0;
  char *p = argv1;
  char *str;
  int pid_args[4];
  arg1 = atoi(argv1);
  arg2 = atoi(argv2);
  arg3 = atoi(argv3);


  // Estructura switch-case en la que se determina que acciones realizar según
la orden recibida
  switch(cmd) {
    case MOTORS_SPEED: //Función para la asignación de las velocidades de los
motores
      numero_cambio_secuencia_A = arg1;
      numero_cambio_secuencia_B = arg2;
      numero_cambio_secuencia_C = arg3;
    break;
    case GET_BAUDRATE:
      Serial.println(BAUDRATE);
    break;
    case ANALOG_READ:
      Serial.println(analogRead(arg1));
```

```
      Serial.println("El pin que se esta leyendo es el: ");
      Serial.print(arg1);
    break;
    case DIGITAL_READ:
      Serial.println(digitalRead(arg1));
    break;
    case ANALOG_WRITE:
      analogWrite(arg1, arg2);
      Serial.println("OK");
      break;
    case DIGITAL_WRITE:
      if (arg2 == 0){
        digitalWrite(arg1, LOW);
      }else if (arg2 == 1){
        digitalWrite(arg1, HIGH);
      };
      Serial.println("OK");
    break;
    case PIN_MODE:
      if (arg2 == 0){
        pinMode(arg1, INPUT);
      }else if (arg2 == 1){
        pinMode(arg1, OUTPUT);
      }
      Serial.println("OK");
    break;
  default:
    Serial.println("Comando recibido inválido");
    break;
  }
}


String Stringchr = "";


// Una vez se ha configurado todo lo necesario del Ros_Arduino_Bridge se
inician las interrupciones


// Interrupciones para el movimiento de los motores
#include "Interrupciones.h"
```

```
// Configuración inicial
void setup() {
   // Se establecen los pines de control de los motores como salidas.
  pinMode(M11, OUTPUT);
  pinMode(M12, OUTPUT);
  pinMode(M13, OUTPUT);
  pinMode(M14, OUTPUT);
  pinMode(M21, OUTPUT);
  pinMode(M22, OUTPUT);
  pinMode(M23, OUTPUT);
  pinMode(M24, OUTPUT);
  pinMode(M31, OUTPUT);
  pinMode(M32, OUTPUT);
  pinMode(M33, OUTPUT);
  pinMode(M34, OUTPUT);
  Serial.begin(BAUDRATE);


  Serial3.begin(115200);
  sendData("AT+CWMODE=3\r\n",1000); // configuración punto de acceso
  sendData("AT+CWJAP=\"WifiMovil\",\"12345678\"\r\n",1000);//red wifi, usuario
y clave


  delay(5000);
  sendData("AT+CIPMUX=1\r\n",1000); // multiples conexiones


  sendData("AT+CIFSR\r\n",1000); // se obtienen direcciones IP


  sendData("AT+CIPSTART=0,\"UDP\",\"0.0.0.0\",4445,4445,2\r\n",1000); // Para
recibir la información procedente de ROS en el puerto 4445
  sendData("AT+CIPSTART=1,\"UDP\",\"192.168.43.189\",4446,4446,2\r\n",1000);
// IP del ordenador destino. Se envía al puerto 4446
  sendData("AT+CIPDINFO=0\r\n",1000);


  delay(500);


  setupTimer3_25mS();
  delay(1000);
```

```
  // Se muestra por la pantalla LCD un mensaje para que el usuario sepa que la
configuración ha terminado
  lcd.begin(); // En windows es lcd.init();
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("Robot");
  lcd.setCursor (0,1);
  lcd.print("Omnidireccional");
  delay(1000);
  lcd.noBacklight(); // Se apaga la pantalla tras el aviso para reducir el
consumo de batería
}


// Bucle principal
void loop() {
  // Se actualiza la odometría.


  ciclos += 1;


  if(avanceMotores[0] > 0){ // Sentido positivo motor 1
    Pose[0] += (avance/10)*cos(((Pose[2]*PI)/180)-(PI/6));
    Pose[1] += (avance/10)*sin(((Pose[2]*PI)/180)-(PI/6));
    Pose[2] -= rotacion;
    avanceMotores[0] -=1;
  }
  if(avanceMotores[1] > 0){ // Sentido negativo motor 1
    Pose[0] -= (avance/10)*cos(((Pose[2]*PI)/180)-(PI/6));
    Pose[1] -= (avance/10)*sin(((Pose[2]*PI)/180)-(PI/6));
    Pose[2] += rotacion;
    avanceMotores[1] -=1;
  }
  if(avanceMotores[2] > 0){ // Sentido positivo motor 2
    Pose[0] += (avance/10)*sin(((Pose[2]*PI)/180)-(PI/3));
    Pose[1] -= (avance/10)*cos(((Pose[2]*PI)/180)-(PI/3));
    Pose[2] -= rotacion;
    avanceMotores[2] -=1;
  }
  if(avanceMotores[3] > 0){ // Sentido negativo motor 2
    Pose[0] -= (avance/10)*sin(((Pose[2]*PI)/180)-(PI/3));
```

```
    Pose[1] += (avance/10)*cos(((Pose[2]*PI)/180)-(PI/3));
    Pose[2] += rotacion;
    avanceMotores[3] -=1;
  }
  if(avanceMotores[4] > 0){ // Sentido positivo motor 3
    Pose[0] -= (avance/10)*cos(((Pose[2]*PI/180))-(PI/2));
    Pose[1] -= (avance/10)*sin(((Pose[2]*PI/180))-(PI/2));
    Pose[2] -= rotacion;
    avanceMotores[4] -=1;
  }
  if(avanceMotores[5] > 0){ // Sentido negativo motor 3
    Pose[0] += (avance/10)*cos(((Pose[2]*PI/180))-(PI/2));
    Pose[1] += (avance/10)*sin(((Pose[2]*PI/180))-(PI/2));
    Pose[2] += rotacion;
    avanceMotores[5] -=1;
  }

  // Se corrige el ángulo de la pose para que este entre 0 y 360 grados
  if(Pose[2] < 0){
    Pose[2] += 360;
  }
  if(Pose[2] > 360){
    Pose[2] -= 360;
  }

  if (Serial3.available()) { // Si ha legado un mensaje se almacena el
contenido del mismo
    Serial3.print("AT+CIPRECVMODE=1\r\n"); // Modo de recepción de datos
    getData(10);

    for(int j = 0; j < Stringchr.length(); j++){ // Se analiza el mensaje
letra por letra hasta el caracter CR
    chr = Stringchr[j];

    if (chr == 13) {
      if (arg == 1) argv1[index] = NULL;
      else if (arg == 2) argv2[index] = NULL;
        else if (arg == 3) argv3[index] = NULL;
        Serial.print("Command ");
```

```
    Serial.println(cmd);
    runCommand();
    resetCommand();
}
// Se utilizan espacios para delimitar las partes del mensaje
else if (chr == ' ') {
  // Se cambia al segundo argumento
  if (arg == 0) arg = 1;
  else if (arg == 1)  {
    argv1[index] = NULL;
    arg = 2;
    index = 0;
  } Se cambia al tercer argumento
  else if (arg == 2)  {
    argv2[index] = NULL;
    arg = 3;
    index = 0;
  }
  continue;
}
else {
  if (arg == 0) {
    // El primer argumento es un comando de una letra
    cmd = chr;
  }
  else if (arg == 1) {
    argv1[index] = chr;
    index++;
  }
  else if (arg == 2) {
    argv2[index] = chr;
    index++;
  }
  else if (arg == 3) {
    argv3[index] = chr;
    index++;
  }
}
```

```
  }


  // Se envía la nueva pose


  if(ciclos > 6000){ // Solo se realiza el envío si han pasado un número de
ciclos para reducir la sobrecarga

    respuesta = "(" + String(Pose[0]) + "," + String(Pose[1]) + "," +
String(Pose[2]) + ")";

    sendData("AT+CIPSEND=" + String(1) + "," + respuesta.length() + "\r\n",
10); // Modo de envío de datos

    sendData(respuesta, 10);

    ciclos = 0;

  }


  }

}


// Función para enviar datos

String sendData(String command, const int timeout){


    String response = "";

    Serial3.print(command);

    long int time = millis();

    while( (time+timeout) > millis()){

      while(Serial3.available()){

        char c = Serial3.read();

        response+=c;

      }

    }


    return response;

}


// Función para recibir datos

void getData( const int timeout){


    Stringchr = "";

    long int time = millis();

    while( (time+timeout) > millis()){

      while(Serial3.available()){
```

```
        char c = Serial3.read();
        Stringchr +=c;
      }
    }
}
```

6.2.5. Nodo de generación de las velocidades de los motores (joy_to_speed)

```
// Programa para la generacion de un topico en el que se publiquen las velocidades de los motores

// a partir de los datos recibidos del nodo joy_node


// Autor: Alberto Diaz Rodriguez

// Universidad de La Laguna



// Se incluyen las librerias necesarias

#include <ros/ros.h> // Incluye algunas cabeceras de uso habitual en ROS.

#include <geometry_msgs/Twist.h> // Para poder trabajar con este tipo de mensaje.

#include <sensor_msgs/Joy.h> // Para poder leer los mensajes enviados por el joystick.

#include "std_msgs/String.h" // Variables tipo string

#include <sstream> // Para la salida de datos

#include <cmath> // Para utilizar las funciones con las que calcular raices y arcotangentes

#include <geometry_msgs/Vector3.h> // Variable de tipo Vector3


#define MODMAX 100 // El valor maximo que se puede recibir del eje de un joystick

#define MAXVEL 206 // Valor maximo que se permite que sean las velocidades generadas


// Variables globales

geometry_msgs::Twist twistJoystickIzq;

geometry_msgs::Twist twistJoystickDer;


// Variables para la velocidad de los motores

float m1_vel = 0;

float m2_vel = 0;

float m3_vel = 0;


// Variable para corregir valores mayores al maximo

float corregirVel = 0;


class ControlRemoto
```

```
{
public:
  ControlRemoto(); // Constructor por defecto de la clase


private:
  // Metodo de la clase
  void joyCallback(const sensor_msgs::Joy::ConstPtr& joy); // Lee el mensaje
enviado por el joystick




  // Atributos de la clase
  ros::NodeHandle n_; // Punto de acceso para comunicaciones ROS.


  int linear_, angular_;




  ros::Subscriber joy_sub_; // Nodo
};


// Se inicializan variables con las que se podra acceder a los datos del mando
deseados
ControlRemoto::ControlRemoto():
  linear_(1),
  angular_(2)
{


  n_.param("axis_linear", linear_, linear_);


  // Se subscribe el nodo al topico joy para recibir los datos del mando
  joy_sub_ = n_.subscribe<sensor_msgs::Joy>("joy", 10,
&ControlRemoto::joyCallback, this);


}


// Funcion en la que se recogen los valores de los ejes de los josticks
void ControlRemoto::joyCallback(const sensor_msgs::Joy::ConstPtr& joy)
{
  // Ejes X e Y del joystick izquierdo
  twistJoystickIzq.linear.x = -100 * joy->axes[linear_ - 1];
  twistJoystickIzq.linear.y = 100 * joy->axes[linear_];
```

```
  // Eje X del joystick derecho
  twistJoystickDer.linear.x = -100 * joy->axes[linear_ + 2];

}

// Funcion para la determinacion de los valores de las velocidades de los
motores
void getVel(){
  // Se inicializan las velocidades a 0
  m1_vel = 0;
  m2_vel = 0;
  m3_vel = 0;

  // La velocidad de cada motor se determinara en base a la contribucion de
cada eje.

  // Contribucion del eje X del joystick derecho (Rotacion pura).

  m1_vel = 255*twistJoystickDer.linear.x/MODMAX;
  m2_vel = 255*twistJoystickDer.linear.x/MODMAX;
  m3_vel = 255*twistJoystickDer.linear.x/MODMAX;

  // Contribucion del eje Y del joystick izquierdo

  m1_vel += 255*twistJoystickIzq.linear.y/MODMAX;
  m2_vel -= 255*twistJoystickIzq.linear.y/MODMAX;
  m3_vel += 0;

  // Contribucion del eje X del joystick izquierdo

  m1_vel += 147.23*twistJoystickIzq.linear.x/MODMAX;
  m2_vel += 147.23*twistJoystickIzq.linear.x/MODMAX;
  m3_vel -= 294.46*twistJoystickIzq.linear.x/MODMAX;

  // Se reescalan los resultados para hacer que el robot no se mueva
excesivamente lento
  // en caso de rotacion pura o traslacion pura

  // Rotacion pura
```

```
    if(twistJoystickIzq.linear.x == 0 && twistJoystickIzq.linear.y == 0){
      m1_vel *= 2.4;
      m2_vel *= 2.4;
      m3_vel *= 2.4;
    }

  // Traslacion pura
  if(twistJoystickDer.linear.x == 0){
      m1_vel *= 1.7;
      m2_vel *= 1.7;
      m3_vel *= 1.7;
    }

  // Se hace la media de las 3 contribuciones
  m1_vel = m1_vel/3;
  m2_vel = m2_vel/3;
  m3_vel = m3_vel/3;

  /*
  // Este fragmento es si se quiere utilizar el metodo de aproximar al maximo
posible los valores
  // Es posible que tras el reescalado el resultado supere el maximo valor de
velocidad. Se corrige

  if(abs(m1_vel) > MAXVEL || abs(m2_vel) > MAXVEL || abs(m3_vel) > MAXVEL){
    // Se averigua cual es el mayor
    if(abs(m1_vel) >= abs(m2_vel)){
      if(abs(m1_vel) >= abs(m3_vel)){
        corregirVel = abs(m1_vel)/MAXVEL;
      }
      else{
        corregirVel = abs(m3_vel)/MAXVEL;
      }
    }else{
      if(fabs(m2_vel) >= fabs(m3_vel)){
        corregirVel = abs(m2_vel)/MAXVEL;
      }else{
        corregirVel = abs(m3_vel)/MAXVEL;
      }
```

```
    }
    m1_vel = m1_vel/corregirVel;
    m2_vel = m2_vel/corregirVel;
    m3_vel = m3_vel/corregirVel;
  }
  */

  // Se convierte el resultado en un entero antes de enviarlo
  m1_vel = int(m1_vel);
  m2_vel = int(m2_vel);
  m3_vel = int(m3_vel);

}

// Se inicia el main
int main(int argc, char** argv)
{

  ros::init(argc, argv, "joy_to_speed"); // Se inicializa el nodo ROS.
  ros::NodeHandle n2;

  // Se publicara en el topico motors_speed la velocidad de los motores
  ros::Publisher chatter_pub =
n2.advertise<geometry_msgs::Vector3>("motors_speed", 1000);
  ros::Rate loop_rate(5); // Frecuencia de publicacion de los mensajes
  ControlRemoto mando; //Se crea un objeto de la clase ControlRemoto



  while (ros::ok())
  {

  // Se determinan las velocidades de los motores
  getVel();



  // Se introducen las velocidades de los tres motores en un vector de 3
posiciones

  geometry_msgs::Vector3 msg;
```

```
  msg.x = m1_vel;
  msg.y = m2_vel;
  msg.z = m3_vel;

  // Se publican las velocidades
  chatter_pub.publish(msg);

  ros::spinOnce();

  loop_rate.sleep();
  }
}
```

6.2.6. Nodo para la recepción de la pose y publicación de la odometría del robot

```
#!/usr/bin/env python


# Programa en el que se implementa un nodo que publica un mensaje con la
odometria

# del robot. Para ello se incorpora un socket UDP que recibe la pose


# Librerias


# Funciones matematicas

import math

from math import sin, cos, pi


# Librerias para la publicacion de la odometria

import rospy

import tf

from nav_msgs.msg import Odometry

from geometry_msgs.msg import Point, Pose, Quaternion, Twist, Vector3


import socket # Para poder utilizar los sockets


rospy.init_node('odometry_publisher') # Se inicia el nuevo nodo


odom_pub = rospy.Publisher("odom", Odometry, queue_size=50) # Se publicara en
el topico odom

odom_broadcaster = tf.TransformBroadcaster()


# Variables necesarias para los calculos posteriores

x = 0.0

y = 0.0

th = math.pi/2


last_x = 0.0

last_y = 0.0

last_th = math.pi/2


vx = 0.0

vy = 0.0
```

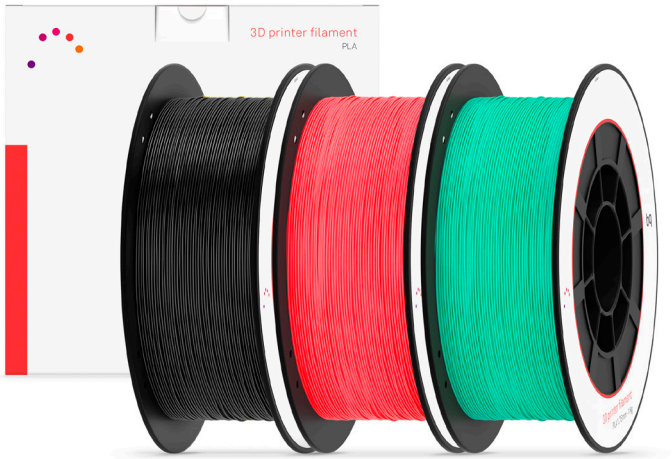
```
vth = 0.0

# Se llevara un control del tiempo para calcular la velocidad del robot
current_time = rospy.Time.now()
last_time = rospy.Time.now()

r = rospy.Rate(1.0)

# Parametros para establecer la conexion
UDP_IP = "0.0.0.0"
UDP_PORT = 4446

# Se inicia el socket
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((UDP_IP, UDP_PORT))

while not rospy.is_shutdown():
    current_time = rospy.Time.now() # Se almacena el tiempo actual

    # Se espera a recibir el mensaje del Arduino
    data, addr = sock.recvfrom(1024) # buffer size is 1024 bytes
    # En data se encuentra el string "(x,y,Theta)

    # Tras recibir el mensaje se actualiza la pose antes de publicarla

    stringX = ""
    stringY = ""
    stringTh = ""

    i = 0 # iteracion
    j = 0 # string que toca rellenar
    while data[i] != ")" and i < 100:

      if (data[i] == "(" or data[i] == ","):
        # Se cambia de string a rellenar
        j = j + 1
      else:
        if j == 1:
```

```
          stringX = stringX + data[i]
        if j == 2:
          stringY = stringY + data[i]
        if j == 3:
          stringTh = stringTh + data[i]

      i = i + 1

    # Se convierten los valores a float para poder trabajar con ellos

    x = float(stringX)/100
    y = float(stringY)/100
    th = float(stringTh)*math.pi/180

    # Se prepara el mensaje con la odometria del robot
    dt = (current_time - last_time).to_sec() # Se calcula la variacion de
tiempo en la ultima ejecucion

    # Se averigua cuanto ha cambiado la pose en el ultimo ciclo
    delta_x = x - last_x
    delta_y = y - last_y
    delta_th = th - last_th

    # Se calcula la velocidad como el espacio que se ha recorrido entre el
tiempo que se ha tardado

    vx = delta_x/dt
    vy = delta_y/dt
    vth = delta_th/dt

    # Al ser la odometria de 6 grados de libertad hay que crear el cuaternion
    odom_quat = tf.transformations.quaternion_from_euler(0, 0, th)

    # Primero se publica la transformada del paso anterior
    odom_broadcaster.sendTransform(
        (x, y, 0.),
        odom_quat,
        current_time,
```

```
        "base_link",
        "odom"
    )

    # Se publica el mensaje con la odometria
    odom = Odometry()
    odom.header.stamp = current_time
    odom.header.frame_id = "odom"

    # Se situa el robot en el plano
    odom.pose.pose = Pose(Point(x, y, 0), Quaternion(*odom_quat))

    # Se indica la velocidad del robot
    odom.child_frame_id = "base_link"
    odom.twist.twist = Twist(Vector3(vx, vy, 0), Vector3(0, 0, vth))

    # Finalmente se publica el mensaje
    odom_pub.publish(odom)

    # Se almacenan los valores de la pose en este momento para conocerlos en
la siguiente ejecucion
    last_x = x
    last_y = y
    last_th = th

    last_time = current_time # Se almacena el momento en el que se termina
este ciclo
```

6.3. Datasheets

PLA Filament 1.75 mm



PLA (Polylactic acid) is the best material for getting started with your 3D printer, as it:

– Hardens quickly
– Has minimal thermal tension
– Has minimal deformation
– Does not require Kapton tape
– Does not require heated bed
–Acetone-resistant

BQ PLA filament is made from 100% PLA.
PLA is a biodegradable product obtained from plant-derived sugars.

Flexural elastic modulus: 3600 MPa (ISO 178)
Flexural strength: 108 MPa (ISO 178)
Hardness: 85 Sh D (ASTM D2240)

	Injection-moulded test pieces	Printed test pieces[1]	Printed test pieces[2]
Tensile strength at break*	52 MPa	50 MPa	39 MPa
Tensile elongation at break*	5%	9%	4%
Tensile modulus*	1320 MPa	1230 MPa	1120 MPa

[1] Stretched parallel to layers
[2] Stretched perpendicular to layers
* ISO 527

Filament Diameter: 1.75 mm
Thickness: 1.24 g/cm³ (ASTM D792)
Weight: 1 kg
Spool Size: 195 mm x 73 mm

Recommended printing temperature: 200/220 ºC
Heat distortion temperature: 56 ºC (ISO 75/2B)
Melting temperature: 145/160 ºC (ASTM D3418)
Glass Transition Temperature: 56/64 ºC (ASTM D3418)

Compatible with: any printer that uses 1.75 mm filament

SKU: F000097
8 435439 881425
Coral

SKU: F000098
8 435439 881432
Turquoise

SKU: F000099
8 435439 881449
Violet

SKU: F000100
8 435439 881456
Sulphur yellow

SKU: F000101
8 435439 881463
Topaz blue

SKU: 05BQFIL023
8 436545 513651
Aubergine

SKU: 05BQFIL024
8 436545 513668
Magenta

SKU: 05BQFIL025
8 436545 513743
Sky blue

SKU: 05BQFIL026
8 436545 513682
Coal black

SKU: 05BQFIL027
8 436545 513699
Pure white

SKU: 05BQFIL028
8 436545 513705
Vitamine orange

SKU: 05BQFIL029
8 436545 513712
Ruby red

SKU: 05BQFIL030
8 436545 513729
Grass green

SKU: 05BQFIL031
8 436545 513736
Bottle green

SKU: 05BQFIL032
8 436545 513750
Ash grey

SKU: 05BQFIL033
8 436545 513767
Transparent

SKU: 05BQFIL034
8 436545 513774
Sunshine yellow

Recreus
www.recreus.com
MADE IN SPAIN

FilaFlex

THE ORIGINAL ELASTIC FILAMENT FOR 3D PRINTING

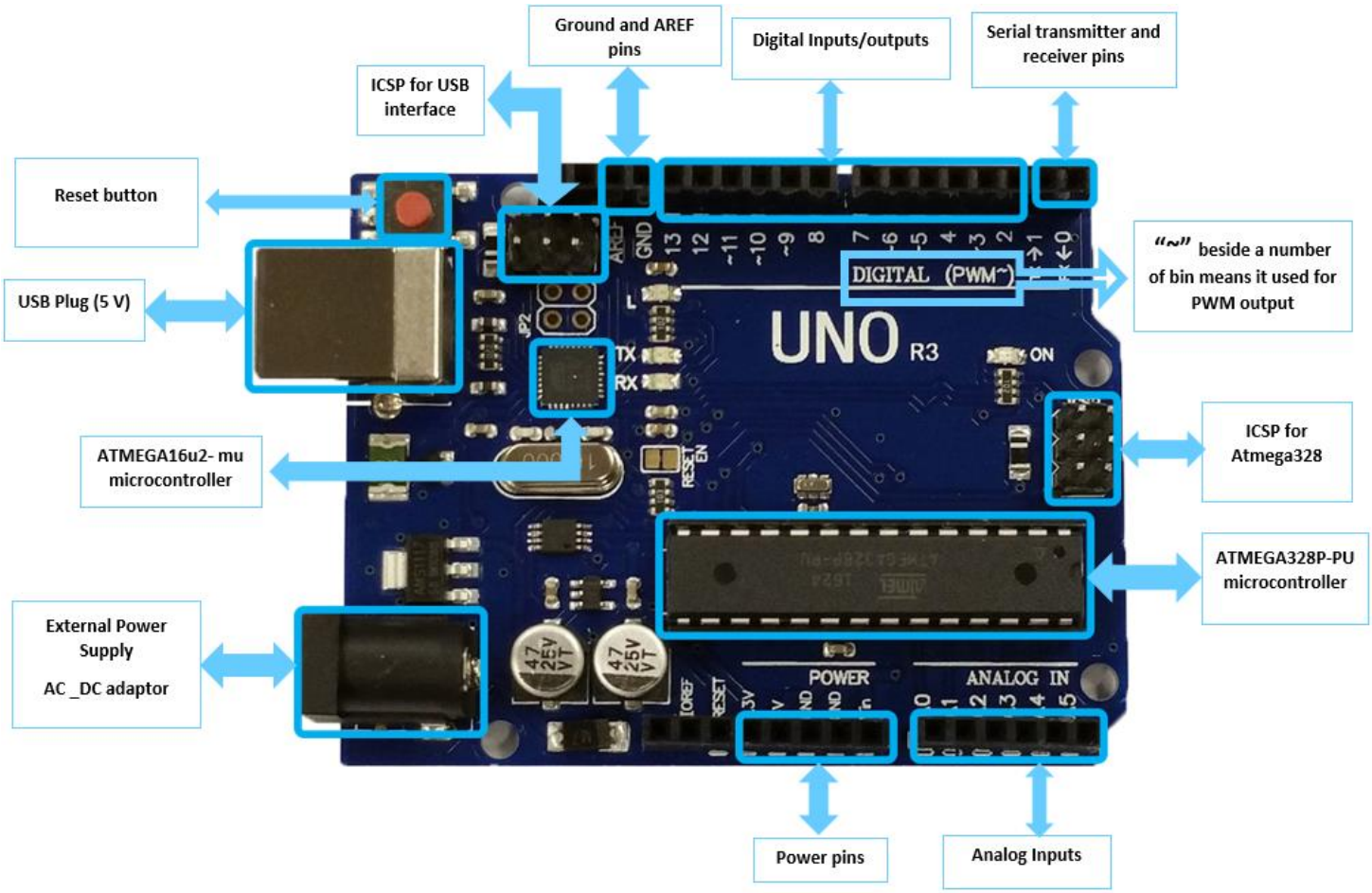
PROPERTIES	STANDARD	VALUE	UNIT	TEST CONDITION
Shore hardness, method A	ISO 868	**82**	**SHORE A**	
Ultimate tensile strength	DIN 53504	**54**	**Mpa**	
Elongation to break	DIN 53504	**700**	**%**	200 mm/min
Compression set	ISO 815	**25**	**%**	72 h; 23 °C
Impact resilence	ISO 4662	**42**	**%**	
Abrasion resistance	ISO 4649 method A	**30**	**mm^3**	
Tear propagation resistance	ISO 34-1	**70**	**kN/m**	500mm/min
Density	ISO 1183-1	**1200**	**kg/m^3**	
Tensile storage modulus	ISO 6721-1,-4	**48**	**MPa**	20 °C
Tensile storage modulus	ISO 6721-1,-5	**33**	**MPa**	60 °C
Extrusion-Melt Temperature		**200-260**	**°C**	

CONSIDERATIONS

Filaflex ***is not a Medical grade material***, cannot be used with direct contact with body fluids including direct contact with blood

Filaflex ***is not designated to food contact or cosmestics applications***.

Arduino Uno R3



INTRODUCTION

Arduino is used for building different types of electronic circuits easily using of both a physical programmable circuit board usually microcontroller and piece of code running on computer with USB connection between the computer and Arduino.

Programming language used in Arduino is just a simplified version of C++ that can easily replace thousands of wires with words.

ARDUINO UNO-R3 PHYSICAL COMPONENTS

ATMEGA328P-PU microcontroller

The most important element in Arduino Uno R3 is ATMEGA328P-PU is an 8-bit Microcontroller with flash memory reach to 32k bytes. It's features as follow:

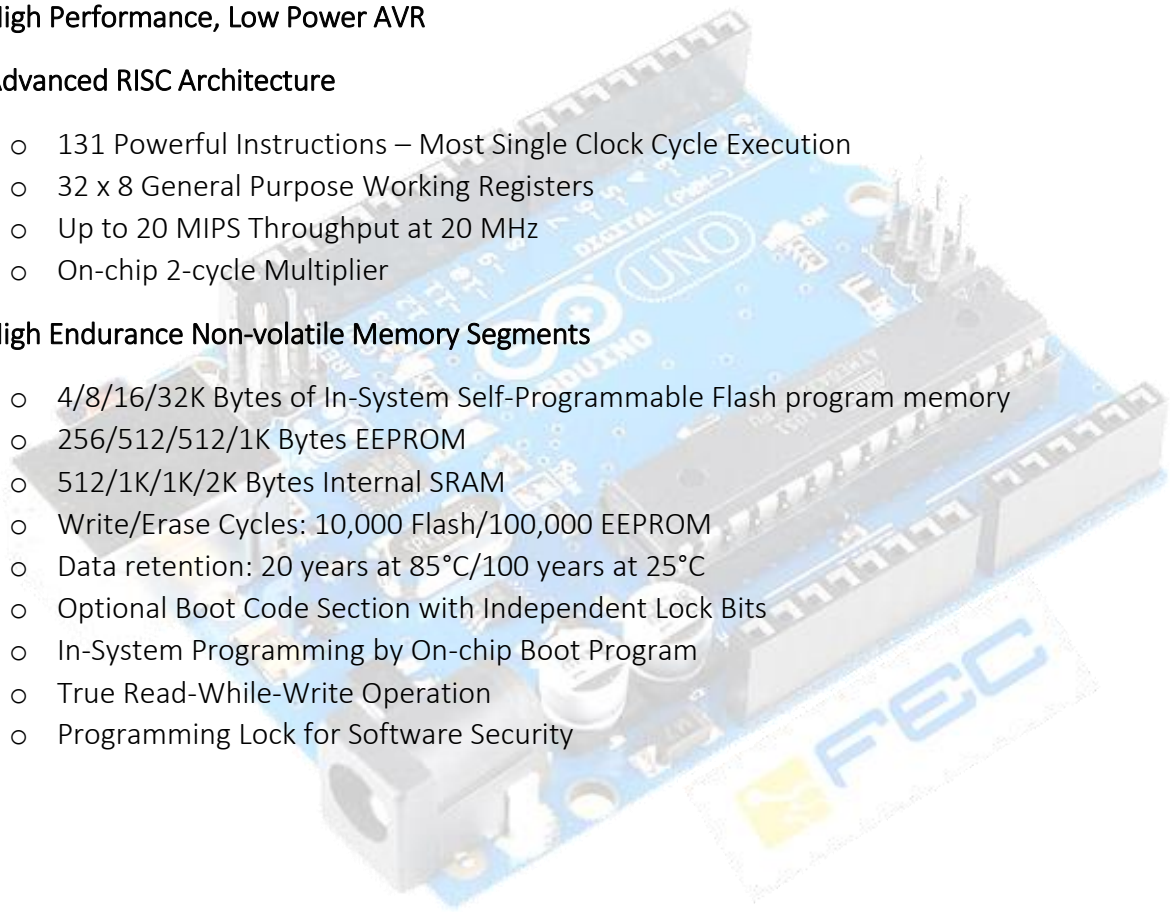
• High Performance, Low Power AVR

• Advanced RISC Architecture

- 131 Powerful Instructions – Most Single Clock Cycle Execution
- 32 x 8 General Purpose Working Registers
- Up to 20 MIPS Throughput at 20 MHz
- On-chip 2-cycle Multiplier

• High Endurance Non-volatile Memory Segments

- 4/8/16/32K Bytes of In-System Self-Programmable Flash program memory
- 256/512/512/1K Bytes EEPROM
- 512/1K/1K/2K Bytes Internal SRAM
- Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
- Data retention: 20 years at 85°C/100 years at 25°C
- Optional Boot Code Section with Independent Lock Bits
- In-System Programming by On-chip Boot Program
- True Read-While-Write Operation
- Programming Lock for Software Security



• Peripheral Features

- Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
- One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
- Real Time Counter with Separate Oscillator
- Six PWM Channels
- 8-channel 10-bit ADC in TQFP and QFN/MLF package
- Temperature Measurement
- 6-channel 10-bit ADC in PDIP Package
- Temperature Measurement
- Programmable Serial USART

- Master/Slave SPI Serial Interface
- Byte-oriented 2-wire Serial Interface (Philips I2 C compatible)
- Programmable Watchdog Timer with Separate On-chip Oscillator
- On-chip Analog Comparator
- Interrupt and Wake-up on Pin Change

• Special Microcontroller Features

- Power-on Reset and Programmable Brown-out Detection
- Internal Calibrated Oscillator
- External and Internal Interrupt Sources
- Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby, and Extended Standby

• I/O and Packages

- 23 Programmable I/O Lines
- 28-pin PDIP, 32-lead TQFP, 28-pad QFN/MLF and 32-pad QFN/MLF

• Operating Voltage:

- 1.8 - 5.5V

• Temperature Range:

- -40°C to 85°C

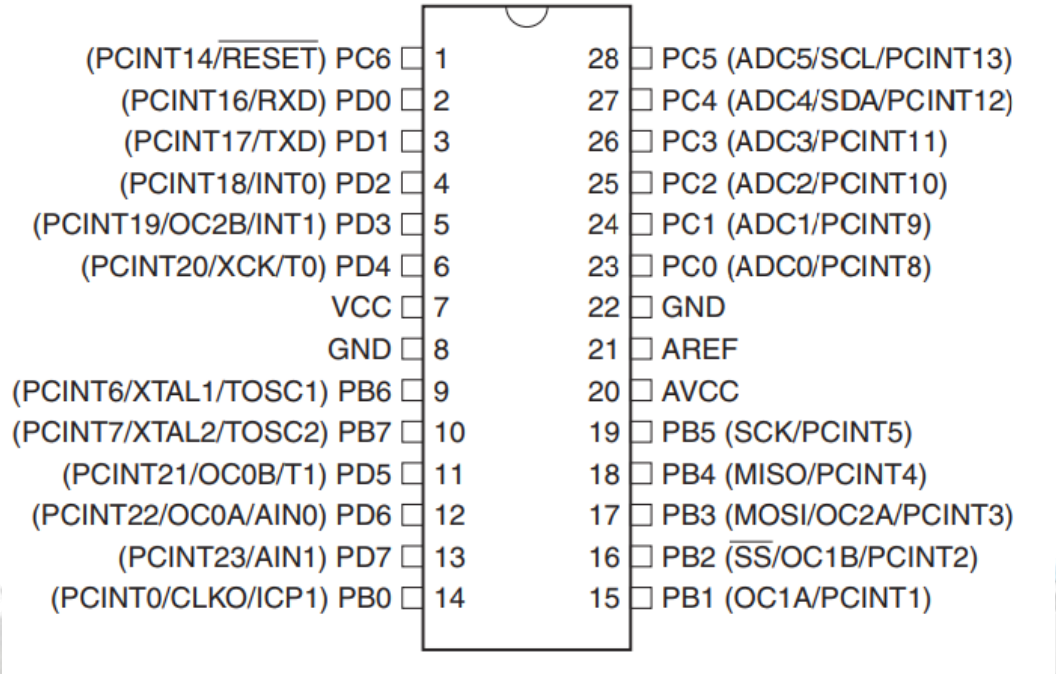
• Speed Grade:

- 0 - 4 MHz@1.8 - 5.5V, 0 - 10 MHz@2.7 - 5.5.V, 0 - 20 MHz @ 4.5 - 5.5V

• Power Consumption at 1 MHz, 1.8V, 25°C

- Active Mode: 0.2 mA
- Power-down Mode: 0.1 µA
- Power-save Mode: 0.75 µA (Including 32 kHz RTC)

- Pin configuration



ATMEGA16u2- mu microcontroller

Is a 8-bit microcontroller used as USB driver in Arduino uno R3 it's features as follow:

- High Performance, Low Power AVR

• Advanced RISC Architecture

- 125 Powerful Instructions – Most Single Clock Cycle Execution
- 32 x 8 General Purpose Working Registers
- Fully Static Operation
- Up to 16 MIPS Throughput at 16 MHz

• Non-volatile Program and Data Memories

- 8K/16K/32K Bytes of In-System Self-Programmable Flash
- 512/512/1024 EEPROM
- 512/512/1024 Internal SRAM
- Write/Erase Cycles: 10,000 Flash/ 100,000 EEPROM
- Data retention: 20 years at 85˚C/ 100 years at 25˚C

- Optional Boot Code Section with Independent Lock Bits
- In-System Programming by on-chip Boot Program hardware-activated after reset
- Programming Lock for Software Security

• USB 2.0 Full-speed Device Module with Interrupt on Transfer Completion

- Complies fully with Universal Serial Bus Specification REV 2.0
- 48 MHz PLL for Full-speed Bus Operation: data transfer rates at 12 Mbit/s
- Fully independent 176 bytes USB DPRAM for endpoint memory allocation
- Endpoint 0 for Control Transfers: from 8 up to 64-bytes
- 4 Programmable Endpoints:
  - IN or Out Directions
  - Bulk, Interrupt and Isochronous Transfers
  - Programmable maximum packet size from 8 to 64 bytes
  - Programmable single or double buffer
- Suspend/Resume Interrupts
- Microcontroller reset on USB Bus Reset without detach
- USB Bus Disconnection on Microcontroller Request

• Peripheral Features

- One 8-bit Timer/Counters with Separate Prescaler and Compare Mode (two 8-bit PWM channels)
- One 16-bit Timer/Counter with Separate Prescaler, Compare and Capture Mode(three 8-bit PWM channels)
- USART with SPI master only mode and hardware flow control (RTS/CTS)
- Master/Slave SPI Serial Interface
- Programmable Watchdog Timer with Separate On-chip Oscillator
- On-chip Analog Comparator
- Interrupt and Wake-up on Pin Change

• On Chip Debug Interface (debug WIRE)

• Special Microcontroller Features

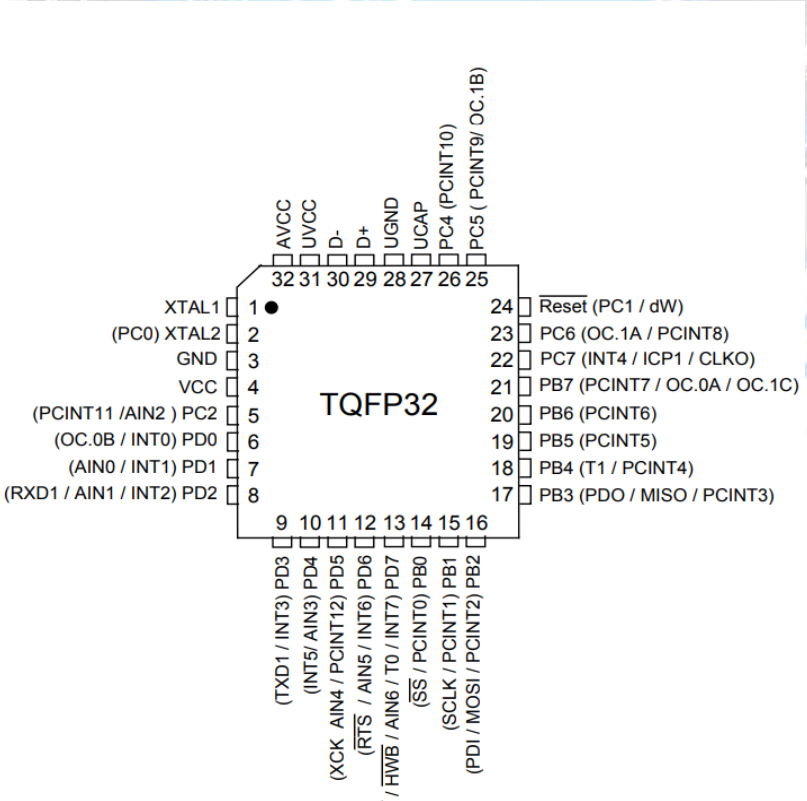
- Power-On Reset and Programmable Brown-out Detection
- Internal Calibrated Oscillator
- External and Internal Interrupt Sources
- Five Sleep Modes: Idle, Power-save, Power-down, Standby, and Extended Standby

• I/O and Packages

- 22 Programmable I/O Lines
- QFN32 (5x5mm) / TQFP32 packages

- Operating Voltages
  - 2.7 - 5.5V
- Operating temperature
  - Industrial (-40°C to +85°C)
- Maximum Frequency
  - 8 MHz at 2.7V - Industrial range
  - 16 MHz at 4.5V - Industrial range

- Pin configuration



OTHER ARDUINO UNO R3 PARTS

Input and Output

Each of the 14 digital pins on the Uno can be used as an input or output, using pinMode (), digitalWrite(), and digitalRead() functions. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 k Ohms. In addition, some pins have specialized functions:

- Serial: 0 (RX) and 1 (TX). Used to receive (RX) and transmit (TX) TTL serial data. These pins are connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip.
- External Interrupts: 2 and 3. These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value.
- PWM: 3, 5, 6, 9, 10, and 11. Provide 8-bit PWM output with the analogWrite() function.
- SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). These pins support SPI communication using the SPI library.
- LED: 13. There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.

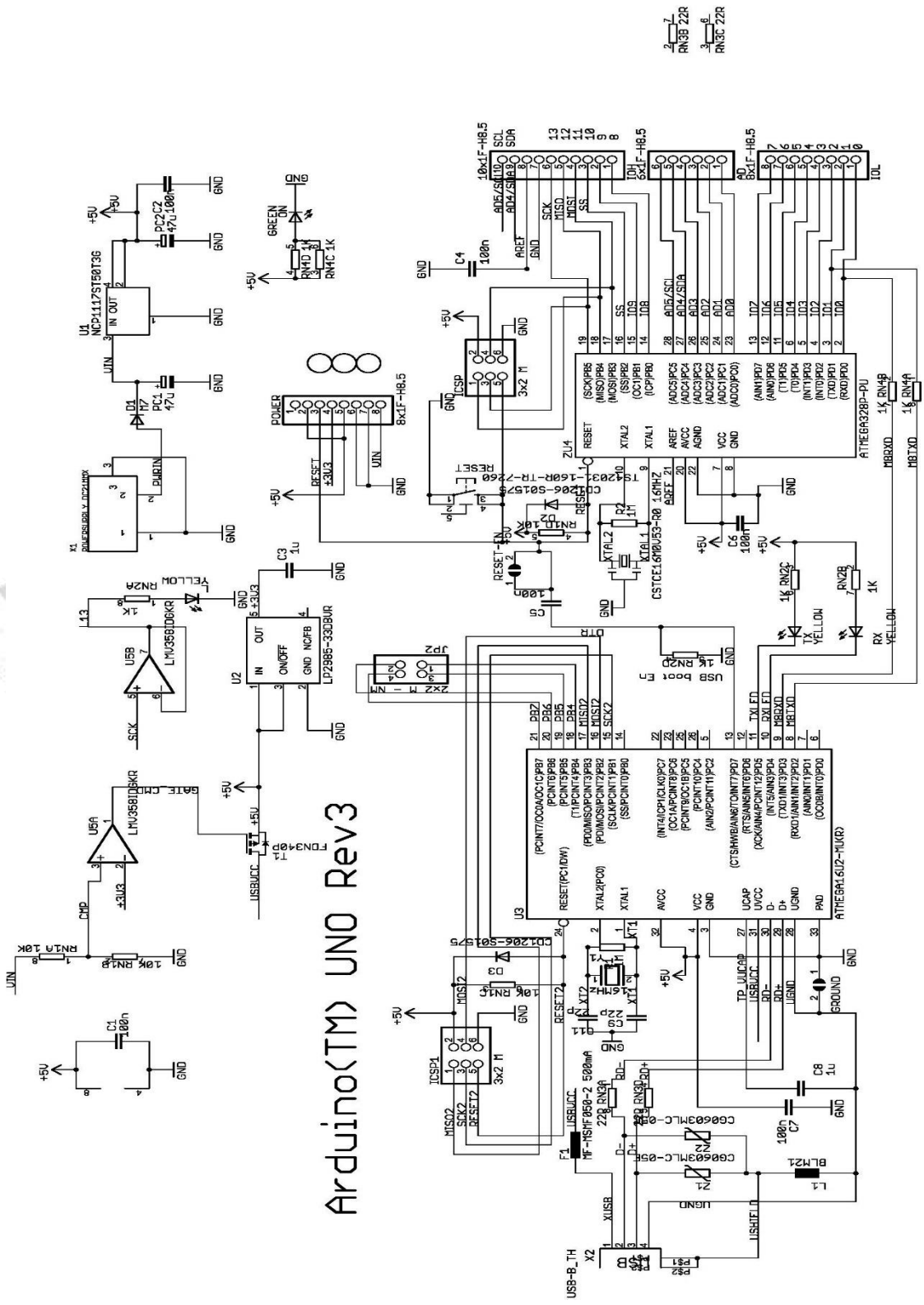
The Uno has 6 analog inputs, labeled A0 through A5, each of which provide 10 bits of resolution (i.e.1024 different values). By default they measure from ground to 5 volts, though is it possible to change the upper end of their range using the AREF pin and the analogReference() function. Additionally, some pins have specialized functionality:

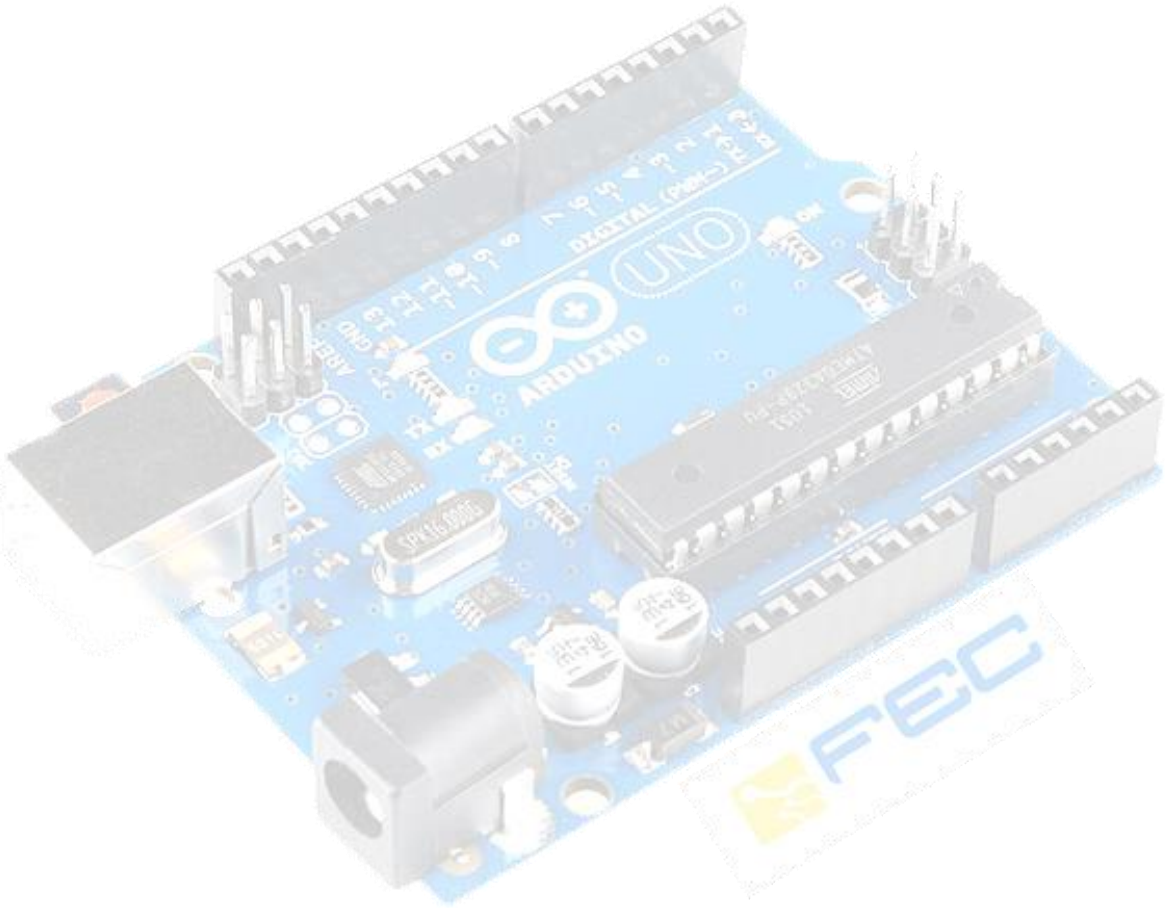
- TWI: A4 or SDA pin and A5 or SCL pin. Support TWI communication using the Wire library.

There are a couple of other pins on the board:

- AREF: Reference voltage for the analog inputs. Used with analogReference().
- Reset: Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

ARDUINO UNO R3 SCHEMATIC DIAGRAM




UNO
ARDUINO

ESP8266EX

Datasheet

Version 6.0

Espressif Systems

About This Guide

This document introduces the specifications of ESP8266EX.

Release Notes

Date	Version	Release Notes
2015.12	V4.6	Updated Chapter 3.
2016.02	V4.7	Updated Section 3.6 and Section 4.1.
2016.04	V4.8	Updated Chapter 1.
2016.08	V4.9	Updated Chapter 1.
2016.11	V5.0	Added Appendix II “Learning Resources”.
2016.11	V5.1	Changed the power consumption during Deep-sleep from 10 μA to 20 μA in Table 5-2.
2016.11	V5.2	Changed the crystal frequency range from “26 MHz to 52 MHz” to “24 MHz to 52 MHz” in Section 3.3.
2016.12	V5.3	Changed the minimum working voltage from 3.0V to 2.5V.
2017.04	V5.4	Changed chip input and output impedance from 50Ω to 39+j6 Ω.
2017.10	V5.5	Updated Chapter 3 regarding the range of clock amplitude to 0.8 ~ 1.5V.
2017.11	V5.6	Updated VDDPST from 1.8V ~ 3.3V to 1.8V ~ 3.6V.
2017.11	V5.7	• Corrected a typo in the description of SDIO_DATA_0 in Table 2-1; • Added the testing conditions for the data in Table 5-2.

Date	Version	Release Notes
2018.02	V5.8	• Updated Wi-Fi protocols in Section 1.1; • Updated description of the integrated Tensilica processor in 3.1.
2018.09	V5.9	• Update document cover; • Added a note for Table 1-1; • Updated Wi-Fi key features in Section 1.1; • Updated description of the Wi-Fi function in 3.5; • Updated pin layout diagram; • Fixed a typo in Table 2-1; • Removed Section AHB and AHB module; • Restructured Section Power Management; • Fixed a typo in Section UART; • Removed description of transmission angle in Section IR Remote Control; • Other optimization (wording).
2018.11	V6.0	• Added an SPI pin in Table 4-2; • Updated the diagram of packing information.

Documentation Change Notification

Espressif provides email notifications to keep customers updated on changes to technical documentation. Please subscribe at *https://www.espressif.com/en/subscribe*.

Certification

Download certificates for Espressif products from *https://www.espressif.com/en/certificates*.

Table of Contents

1. Overview

Espressif's ESP8266EX delivers highly integrated Wi-Fi SoC solution to meet users' continuous demands for efficient power usage, compact design and reliable performance in the Internet of Things industry.

With the complete and self-contained Wi-Fi networking capabilities, ESP8266EX can perform either as a standalone application or as the slave to a host MCU. When ESP8266EX hosts the application, it promptly boots up from the flash. The integrated high-speed cache helps to increase the system performance and optimize the system memory. Also, ESP8266EX can be applied to any microcontroller design as a Wi-Fi adaptor through SPI/SDIO or UART interfaces.

ESP8266EX integrates antenna switches, RF balun, power amplifier, low noise receive amplifier, filters and power management modules. The compact design minimizes the PCB size and requires minimal external circuitries.

Besides the Wi-Fi functionalities, ESP8266EX also integrates an enhanced version of Tensilica's L106 Diamond series 32-bit processor and on-chip SRAM. It can be interfaced with external sensors and other devices through the GPIOs. Software Development Kit (SDK) provides sample codes for various applications.

Espressif Systems' Smart Connectivity Platform (ESCP) enables sophisticated features including:

- Fast switch between sleep and wakeup mode for energy-efficient purpose;
- Adaptive radio biasing for low-power operation
- Advance signal processing
- Spur cancellation and RF co-existence mechanisms for common cellular, Bluetooth, DDR, LVDS, LCD interference mitigation

1.1. Wi-Fi Key Features

- 802.11 b/g/n support
- 802.11n support (2.4 GHz), up to 72.2 Mbps
- Defragmentation
- 2 x virtual Wi-Fi interface
- Automatic beacon monitoring (hardware TSF)
- Support Infrastructure BSS Station mode/SoftAP mode/Promiscuous mode
- Antenna diversity

1.2. Specifications

Table 1-1. Specifications

Categories	Items	Parameters
Wi-Fi	Certification	Wi-Fi Alliance
	Protocols	802.11 b/g/n (HT20)
	Frequency Range	2.4G ~ 2.5G (2400M ~ 2483.5M)
	TX Power	802.11 b: +20 dBm
		802.11 g: +17 dBm
		802.11 n: +14 dBm
	Rx Sensitivity	802.11 b: –91 dbm (11 Mbps)
		802.11 g: –75 dbm (54 Mbps)
		802.11 n: –72 dbm (MCS7)
	Antenna	PCB Trace, External, IPEX Connector, Ceramic Chip
Hardware	CPU	Tensilica L106 32-bit processor
	Peripheral Interface	UART/SDIO/SPI/I2C/I2S/IR Remote Control
		GPIO/ADC/PWM/LED Light & Button
	Operating Voltage	2.5V ~ 3.6V
	Operating Current	Average value: 80 mA
	Operating Temperature Range	–40°C ~ 125°C
	Package Size	QFN32-pin (5 mm x 5 mm)
	External Interface	-
Software	Wi-Fi Mode	Station/SoftAP/SoftAP+Station
	Security	WPA/WPA2
	Encryption	WEP/TKIP/AES
	Firmware Upgrade	UART Download / OTA (via network)
	Software Development	Supports Cloud Server Development / Firmware and SDK for fast on-chip programming
	Network Protocols	IPv4, TCP/UDP/HTTP
	User Configuration	AT Instruction Set, Cloud Server, Android/iOS App

📖 ***Note:***

The TX power can be configured based on the actual user scenarios.

1.3. Applications

- Home appliances
- Home automation
- Smart plugs and lights
- Industrial wireless control
- Baby monitors
- IP cameras
- Sensor networks
- Wearable electronics
- Wi-Fi location-aware devices
- Security ID tags
- Wi-Fi position system beacons

2. Pin Definitions

Figure 2-1 shows the pin layout for 32-pin QFN package.

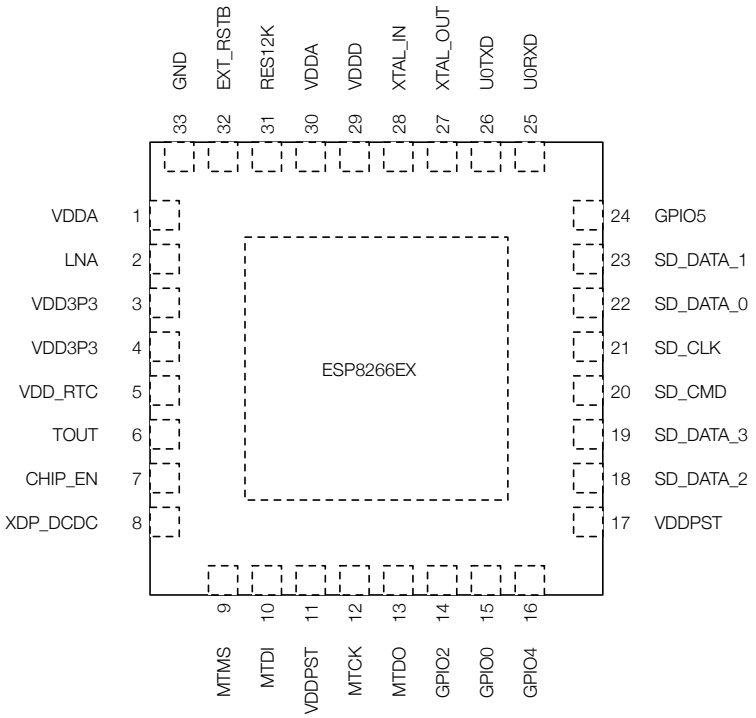


Figure 2-1. Pin Layout (Top View)

Table 2-1 lists the definitions and functions of each pin.

Table 2-1. ESP8266EX Pin Definitions

Pin	Name	Type	Function
1	VDDA	P	Analog Power 2.5V ~ 3.6V
2	LNA	I/O	RF antenna interface Chip output impedance=39+j6 Ω. It is suggested to retain the π-type matching network to match the antenna.
3	VDD3P3	P	Amplifier Power 2.5V ~ 3.6V
4	VDD3P3	P	Amplifier Power 2.5V ~ 3.6V
5	VDD_RTC	P	NC (1.1V)
6	TOUT	I	ADC pin. It can be used to test the power-supply voltage of VDD3P3 (Pin3 and Pin4) and the input power voltage of TOUT (Pin 6). However, these two functions cannot be used simultaneously.

Pin	Name	Type	Function
7	CHIP_EN	I	Chip Enable High: On, chip works properly Low: Off, small current consumed
8	XPD_DCDC	I/O	Deep-sleep wakeup (need to be connected to EXT_RSTB); GPIO16
9	MTMS	I/O	GPIO 14; HSPI_CLK
10	MTDI	I/O	GPIO 12; HSPI_MISO
11	VDDPST	P	Digital/IO Power Supply (1.8V ~ 3.6V)
12	MTCK	I/O	GPIO 13; HSPI_MOSI; UART0_CTS
13	MTDO	I/O	GPIO 15; HSPI_CS; UART0_RTS
14	GPIO2	I/O	UART TX during flash programming; GPIO2
15	GPIO0	I/O	GPIO0; SPI_CS2
16	GPIO4	I/O	GPIO4
17	VDDPST	P	Digital/IO Power Supply (1.8V ~ 3.6V)
18	SDIO_DATA_2	I/O	Connect to SD_D2 (Series R: 200Ω); SPIHD; HSPIHD; GPIO9
19	SDIO_DATA_3	I/O	Connect to SD_D3 (Series R: 200Ω); SPIWP; HSPIWP; GPIO10
20	SDIO_CMD	I/O	Connect to SD_CMD (Series R: 200Ω); SPI_CS0; GPIO11
21	SDIO_CLK	I/O	Connect to SD_CLK (Series R: 200Ω); SPI_CLK; GPIO6
22	SDIO_DATA_0	I/O	Connect to SD_D0 (Series R: 200Ω); SPI_MISO; GPIO7
23	SDIO_DATA_1	I/O	Connect to SD_D1 (Series R: 200Ω); SPI_MOSI; GPIO8
24	GPIO5	I/O	GPIO5
25	U0RXD	I/O	UART Rx during flash programming; GPIO3
26	U0TXD	I/O	UART TX during flash programming; GPIO1; SPI_CS1
27	XTAL_OUT	I/O	Connect to crystal oscillator output, can be used to provide BT clock input
28	XTAL_IN	I/O	Connect to crystal oscillator input
29	VDDD	P	Analog Power 2.5V ~ 3.6V
30	VDDA	P	Analog Power 2.5V ~ 3.6V
31	RES12K	I	Serial connection with a 12 kΩ resistor and connect to the ground
32	EXT_RSTB	I	External reset signal (Low voltage level: active)

📖 ***Note:***

1. *GPIO2, GPIO0, and MTDO are used to select booting mode and the SDIO mode;*
2. *U0TXD should not be pulled externally to a low logic level during the powering-up.*

3. Functional Description

The functional diagram of ESP8266EX is shown as in Figure 3-1.

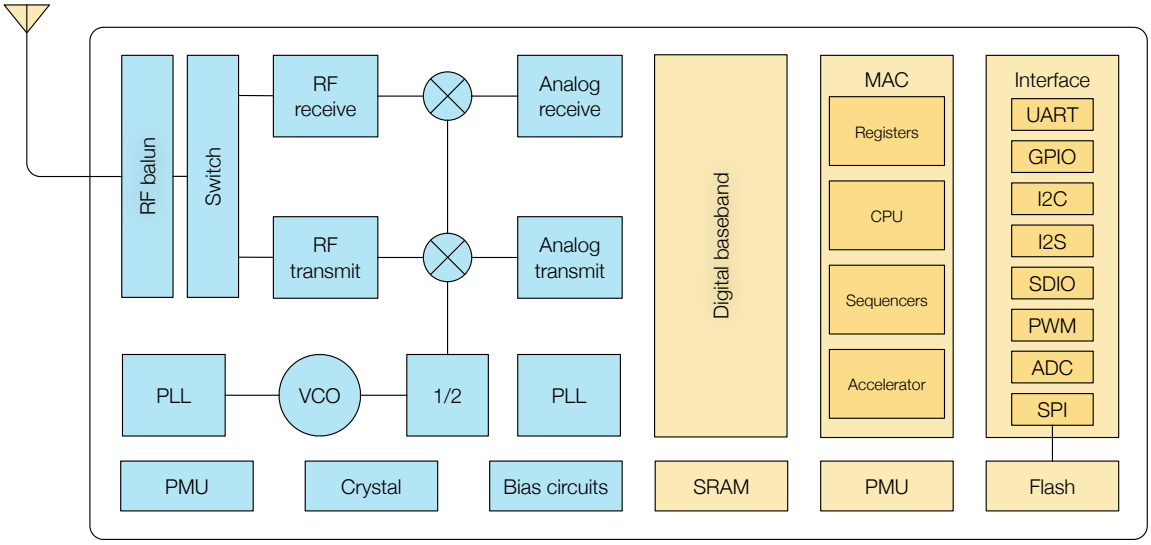


Figure 3-1. Functional Block Diagram

3.1. CPU, Memory, and Flash

3.1.1. CPU

The ESP8266EX integrates a Tensilica L106 32-bit RISC processor, which achieves extra-low power consumption and reaches a maximum clock speed of 160 MHz. The Real-Time Operating System (RTOS) and Wi-Fi stack allow 80% of the processing power to be available for user application programming and development. The CPU includes the interfaces as below:

- Programmable RAM/ROM interfaces (iBus), which can be connected with memory controller, and can also be used to visit flash.
- Data RAM interface (dBus), which can connected with memory controller.
- AHB interface which can be used to visit the register.

3.1.2. Memory

ESP8266EX Wi-Fi SoC integrates memory controller and memory units including SRAM and ROM. MCU can access the memory units through iBus, dBus, and AHB interfaces. All memory units can be accessed upon request, while a memory arbiter will decide the running sequence according to the time when these requests are received by the processor.

According to our current version of SDK, SRAM space available to users is assigned as below.

- RAM size < 50 kB, that is, when ESP8266EX is working under the Station mode and connects to the router, the maximum programmable space accessible in Heap + Data section is around 50 kB.
- There is no programmable ROM in the SoC. Therefore, user program must be stored in an external SPI flash.

3.1.3. External Flash

ESP8266EX uses external SPI flash to store user programs, and supports up to 16 MB memory capacity theoretically.

The minimum flash memory of ESP8266EX is shown below:

- OTA disabled: 512 kB at least
- OTA enabled: 1 MB at least

> ⚠ **Notice:**
>
> *SPI mode supported: Standard SPI, Dual SPI and Quad SPI. The correct SPI mode should be selected when flashing bin files to ESP8266. Otherwise, the downloaded firmware/program may not be working properly.*

3.2. Clock

3.2.1. High Frequency Clock

The high frequency clock on ESP8266EX is used to drive both transmit and receive mixers. This clock is generated from internal crystal oscillator and external crystal. The crystal frequency ranges from 24 MHz to 52 MHz.

The internal calibration inside the crystal oscillator ensures that a wide range of crystals can be used, nevertheless the quality of the crystal is still a factor to consider to have reasonable phase noise and good Wi-Fi sensitivity. Refer to Table 3-1 to measure the frequency offset.

Table 3-1. High Frequency Clock Specifications

Parameter	Symbol	Min	Max	Unit
Frequency	FXO	24	52	MHz
Loading capacitance	CL	-	32	pF
Motional capacitance	CM	2	5	pF
Series resistance	RS	0	65	Ω
Frequency tolerance	ΔFXO	–15	15	ppm
Frequency vs temperature (–25°C ~ 75°C)	ΔFXO,Temp	–15	15	ppm

3.2.2. External Clock Requirements

An externally generated clock is available with the frequency ranging from 24 MHz to 52 MHz. The following characteristics are expected to achieve good performance of radio.

Table 3-2. External Clock Reference

Parameter	Symbol	Min	Max	Unit
Clock amplitude	VXO	0.8	1.5	Vpp
External clock accuracy	ΔFXO,EXT	–15	15	ppm
Phase noise @1-kHz offset, 40-MHz clock	-	-	–120	dBc/Hz
Phase noise @10-kHz offset, 40-MHz clock	-	-	–130	dBc/Hz
Phase noise @100-kHz offset, 40-MHz clock	-	-	–138	dBc/Hz

3.3. Radio

ESP8266EX radio consists of the following blocks.

- 2.4 GHz receiver
- 2.4 GHz transmitter
- High speed clock generators and crystal oscillator
- Bias and regulators
- Power management

3.3.1. Channel Frequencies

The RF transceiver supports the following channels according to IEEE802.11b/g/n standards.

Table 3-3. Frequency Channel

Channel No.	Frequency (MHz)	Channel No.	Frequency (MHz)
1	2412	8	2447
2	2417	9	2452
3	2422	10	2457
4	2427	11	2462
5	2432	12	2467
6	2437	13	2472
7	2442	14	2484

3.3.2. 2.4 GHz Receiver

The 2.4 GHz receiver down-converts the RF signals to quadrature baseband signals and converts them to the digital domain with 2 high resolution high speed ADCs. To adapt to varying signal channel conditions, RF filters, automatic gain control (AGC), DC offset cancelation circuits and baseband filters are integrated within ESP8266EX.

3.3.3. 2.4 GHz Transmitter

The 2.4 GHz transmitter up-converts the quadrature baseband signals to 2.4 GHz, and drives the antenna with a high-power CMOS power amplifier. The function of digital calibration further improves the linearity of the power amplifier, enabling a state of art performance of delivering +19.5 dBm average TX power for 802.11b transmission and +18 dBm for 802.11n (MSC0) transmission.

Additional calibrations are integrated to offset any imperfections of the radio, such as:

- Carrier leakage
- I/Q phase matching
- Baseband nonlinearities

These built-in calibration functions reduce the product test time and make the test equipment unnecessary.

3.3.4. Clock Generator

The clock generator generates quadrature 2.4 GHz clock signals for the receiver and transmitter. All components of the clock generator are integrated on the chip, including all inductors, varactors, loop filters, linear voltage regulators and dividers.

The clock generator has built-in calibration and self test circuits. Quadrature clock phases and phase noise are optimized on-chip with patented calibration algorithms to ensure the best performance of the receiver and transmitter.

3.4. Wi-Fi

ESP8266EX implements TCP/IP and full 802.11 b/g/n WLAN MAC protocol. It supports Basic Service Set (BSS) STA and SoftAP operations under the Distributed Control Function (DCF). Power management is handled with minimum host interaction to minimize active-duty period.

3.4.1. Wi-Fi Radio and Baseband

The ESP8266EX Wi-Fi Radio and Baseband support the following features:

- 802.11b and 802.11g
- 802.11n MCS0-7 in 20 MHz bandwidth
- 802.11n 0.4 µs guard-interval
- up to 72.2 Mbps of data rate

- Receiving STBC 2x1
- Up to 20.5 dBm of transmitting power
- Adjustable transmitting power
- Antenna diversity

3.4.2. Wi-Fi MAC

The ESP8266EX Wi-Fi MAC applies low-level protocol functions automatically, as follows:

- 2 × virtual Wi-Fi interfaces
- Infrastructure BSS Station mode/SoftAP mode/Promiscuous mode
- Request To Send (RTS), Clear To Send (CTS) and Immediate Block ACK
- Defragmentation
- CCMP (CBC-MAC, counter mode), TKIP (MIC, RC4), WEP (RC4) and CRC
- Automatic beacon monitoring (hardware TSF)
- Dual and single antenna Bluetooth co-existence support with optional simultaneous receive (Wi-Fi/Bluetooth) capability

3.5. Power Management

ESP8266EX is designed with advanced power management technologies and intended for mobile devices, wearable electronics and the Internet of Things applications.

The low-power architecture operates in the following modes:

- Active mode: The chip radio is powered on. The chip can receive, transmit, or listen.
- Modem-sleep mode: The CPU is operational. The Wi-Fi and radio are disabled.
- Light-sleep mode: The CPU and all peripherals are paused. Any wake-up events (MAC, host, RTC timer, or external interrupts) will wake up the chip.
- Deep-sleep mode: Only the RTC is operational and all other part of the chip are powered off.

Table 3-4. Power Consumption by Power Modes

Power Mode	Description	Power Consumption
Active (RF working)	Wi-Fi TX packet	Please refer to 5-2.
	Wi-Fi RX packet	
Modem-sleep①	CPU is working	15 mA
Light-sleep②	-	0.9 mA
Deep-sleep③	Only RTC is working	20 uA
Shut down	-	0.5 uA

📖 ***Notes:***

① ***Modem-sleep*** *mode is used in the applications that require the CPU to be working, as in PWM or I2S applications. According to 802.11 standards (like U-APSD), it shuts down the Wi-Fi Modem circuit while maintaining a Wi-Fi connection with no data transmission to optimize power consumption. E.g. in DTIM3, maintaining a sleep of 300 ms with a wakeup of 3 ms cycle to receive AP's Beacon packages at interval requires about 15 mA current.*

② *During* ***Light-sleep*** *mode, the CPU may be suspended in applications like Wi-Fi switch. Without data transmission, the Wi-Fi Modem circuit can be turned off and CPU suspended to save power consumption according to the 802.11 standards (U-APSD). E.g. in DTIM3, maintaining a sleep of 300 ms with a wakeup of 3ms to receive AP's Beacon packages at interval requires about 0.9 mA current.*

③ *During* ***Deep-sleep*** *mode, Wi-Fi is turned off. For applications with long time lags between data transmission, e.g. a temperature sensor that detects the temperature every 100s, sleeps for 300s and wakes up to connect to the AP (taking about 0.3 ~ 1s), the overall average current is less than 1mA. The current of 20 μA is acquired at the voltage of 2.5V.*

4. Peripheral Interface

4.1. General Purpose Input/Output Interface (GPIO)

ESP8266EX has 17 GPIO pins which can be assigned to various functions by programming the appropriate registers.

Each GPIO PAD can be configured with internal pull-up or pull-down (XPD_DCDC can only be configured with internal pull-down, other GPIO PAD can only be configured with internal pull-up), or set to high impedance. When configured as an input, the data are stored in software registers; the input can also be set to edge-trigger or level trigger CPU interrupts. In short, the IO pads are bi-directional, non-inverting and tristate, which includes input and output buffer with tristate control inputs.

These pins, when working as GPIOs, can be multiplexed with other functions such as I2C, I2S, UART, PWM, and IR Remote Control, etc.

For low power operations, the GPIOs can also be set to hold their state. For instance, when the IOs are not driven by internal and external circuits, all outputs will hold their states before the chip entered the low power modes.

The required drive strength is small— 5 µA or more is enough to pull apart the latch.

4.2. Secure Digital Input/Output Interface (SDIO)

ESP8266EX has one Slave SDIO, the definitions of which are described as Table 4-1, which supports 25 MHz SDIO v1.1 and 50 MHz SDIO v2.0, and 1 bit/4 bit SD mode and SPI mode.

Table 4-1. Pin Definitions of SDIOs

Pin Name	Pin Num	IO	Function Name
SDIO_CLK	21	IO6	SDIO_CLK
SDIO_DATA0	22	IO7	SDIO_DATA0
SDIO_DATA1	23	IO8	SDIO_DATA1
SDIO_DATA_2	18	IO9	SDIO_DATA_2
SDIO_DATA_3	19	IO10	SDIO_DATA_3
SDIO_CMD	20	IO11	SDIO_CMD

4.3. Serial Peripheral Interface (SPI/HSPI)

ESP8266EX has two SPIs.

- One general Slave/Master SPI
- One general Slave HSPI

Functions of all these pins can be implemented via hardware.

4.3.1. General SPI (Master/Slave)

Table 4-2. Pin Definitions of SPIs

Pin Name	Pin Num	IO	Function Name
SDIO_CLK	21	IO6	SPICLK
SDIO_DATA0	22	IO7	SPIQ/MISO
SDIO_DATA1	23	IO8	SPID/MOSI
SDIO_DATA_2	18	IO9	SPIHD
SDIO_DATA_3	19	IO10	SPIWP
U0TXD	26	IO1	SPICS1
GPIO0	15	IO0	SPICS2
SDIO_CMD	20	IO11	SPICS0

📖 ***Note:***

SPI mode can be implemented via software programming. The clock frequency is 80 MHz at maximum when working as a master, 20 MHz at maximum when working as a slave.

4.3.2. HSPI (Slave)

Table 4-3. Pin Definitions of HSPI (Slave)

Pin Name	Pin Num	IO	Function Name
MTMS	9	IO14	HSPICLK
MTDI	10	IO12	HSPIQ/MISO
MTCK	12	IO13	HSPID/MOSI
MTDO	13	IO15	HPSICS

📖 ***Note:***

SPI mode can be implemented via software programming. The clock frequency is 20 MHz at maximum.

4.4. I2C Interface

ESP8266EX has one I2C, which is realized via software programming, used to connect with other microcontrollers and other peripheral equipments such as sensors. The pin definition of I2C is as below.

Table 4-4. Pin Definitions of I2C

Pin Name	Pin Num	IO	Function Name
MTMS	9	IO14	I2C_SCL
GPIO2	14	IO2	I2C_SDA

Both I2C Master and I2C Slave are supported. I2C interface functionality can be realized via software programming, and the clock frequency is 100 kHz at maximum.

4.5. I2S Interface

ESP8266EX has one I2S data input interface and one I2S data output interface, and supports the linked list DMA. I2S interfaces are mainly used in applications such as data collection, processing, and transmission of audio data, as well as the input and output of serial data. For example, LED lights (WS2812 series) are supported. The pin definition of I2S is shown in Table 4-5.

Table 4-5. Pin Definitions of I2S

I2S Data Input			
Pin Name	Pin Num	IO	Function Name
MTDI	10	IO12	I2SI_DATA
MTCK	12	IO13	I2SI_BCK
MTMS	9	IO14	I2SI_WS
MTDO	13	IO15	I2SO_BCK
U0RXD	25	IO3	I2SO_DATA
GPIO2	14	IO2	I2SO_WS

4.6. Universal Asynchronous Receiver Transmitter (UART)

ESP8266EX has two UART interfaces UART0 and UART1, the definitions are shown in Table 4-6.

Table 4-6. Pin Definitions of UART

Pin Type	Pin Name	Pin Num	IO	Function Name
UART0	U0RXD	25	IO3	U0RXD
	U0TXD	26	IO1	U0TXD
	MTDO	13	IO15	U0RTS
	MTCK	12	IO13	U0CTS
UART1	GPIO2	14	IO2	U1TXD
	SD_D1	23	IO8	U1RXD

Data transfers to/from UART interfaces can be implemented via hardware. The data transmission speed via UART interfaces reaches 115200 x 40 (4.5 Mbps).

UART0 can be used for communication. It supports flow control. Since UART1 features only data transmit signal (TX), it is usually used for printing log.

> 📖 ***Note:***
>
> *By default, UART0 outputs some printed information when the device is powered on and booting up. The baud rate of the printed information is relevant to the frequency of the external crystal oscillator. If the frequency of the crystal oscillator is 40 MHz, then the baud rate for printing is 115200; if the frequency of the crystal oscillator is 26 MHz, then the baud rate for printing is 74880. If the printed information exerts any influence on the functionality of the device, it is suggested to block the printing during the power-on period by changing (U0TXD, U0RXD) to (MTDO, MTCK).*

4.7. Pulse-Width Modulation (PWM)

ESP8266EX has four PWM output interfaces. They can be extended by users themselves. The pin definitions of the PWM interfaces are defined as below.

Table 4-7. Pin Definitions of PWM

Pin Name	Pin Num	IO	Function Name
MTDI	10	IO12	PWM0
MTDO	13	IO15	PWM1
MTMS	9	IO14	PWM2
GPIO4	16	IO4	PWM3

The functionality of PWM interfaces can be implemented via software programming. For example, in the LED smart light demo, the function of PWM is realized by interruption of the timer, the minimum resolution reaches as high as 44 ns. PWM frequency range is adjustable from 1000 µs to 10000 µs, i.e., between 100 Hz and 1 kHz. When the PWM frequency is 1 kHz, the duty ratio will be 1/22727, and a resolution of over 14 bits will be achieved at 1 kHz refresh rate.

4.8. IR Remote Control

ESP8266EX currently supports one infrared remote control interface. For detailed pin definitions, please see Table 4-8 below.

Table 4-8. Pin Definitions of IR Remote Control

Pin Name	Pin Num	IO	Function Name
MTMS	9	IO14	IR TX
GPIO5	24	IO 5	IR Rx

The functionality of Infrared remote control interface can be implemented via software programming. NEC coding, modulation, and demodulation are supported by this interface. The frequency of modulated carrier signal is 38 kHz, while the duty ratio of the square wave is 1/3. The transmission range is around 1m which is determined by two factors: one is the maximum current drive output, the other is internal current-limiting resistance value in the infrared receiver. The larger the resistance value, the lower the current, so is the power, and vice versa.

4.9. ADC (Analog-to-Digital Converter)

ESP8266EX is embedded with a 10-bit precision SAR ADC. TOUT (Pin6) is defined as below:

Table 4-9. Pin Definition of ADC

Pin Name	Pin Num	Function Name
TOUT	6	ADC Interface

The following two measurements can be implemented using ADC (Pin6). However, they cannot be implemented at the same time.

- Measure the power supply voltage of VDD3P3 (Pin3 and Pin4).

Hardware Design	TOUT must be floating.
RF Initialization Parameter	The 107th byte of ***esp_init_data_default.bin*** (0 ~ 127 bytes), `vdd33_const` must be set to `0xFF`.
RF Calibration Process	Optimize the RF circuit conditions based on the testing results of VDD3P3 (Pin3 and Pin4).
User Programming	Use `system_get_vdd33` instead of `system_adc_read`.

- Measure the input voltage of TOUT (Pin6).

Hardware Design	The input voltage range is 0 to 1.0V when TOUT is connected to external circuit.

RF Initialization Parameter	The value of the 107th byte of ***esp_init_data_default.bin*** (0 ~ 127 bytes), `vdd33_const` must be set to the real power supply voltage of Pin3 and Pin4. The unit and effective value range of `vdd33_const` is 0.1V and 18 to 36, respectively, thus making the working power voltage range of ESP8266EX between 1.8V and 3.6V,
RF Calibration Process	Optimize the RF circuit conditions based on the value of `vdd33_const`. The permissible error is ±0.2V.
User Programming	Use `system_adc_read` instead of `system_get_vdd33`.

📖 ***Notes:***

esp_init_data_default.bin *is provided in SDK package which contains RF initialization parameters (0 ~ 127 bytes). The name of the 107th byte in* ***esp_init_data_default.bin*** *is* `vdd33_const`*, which is defined as below:*

- *When vdd33_const = 0xff, the power voltage of Pin3 and Pin4 will be tested by the internal self-calibration process of ESP8266EX itself. RF circuit conditions should be optimized according to the testing results.*
- *When 18 =< vdd33_const =< 36, ESP8266EX RF Calibration and optimization process is implemented via (vdd33_const/10).*
- *When vdd33_const < 18 or 36 < vdd33_const < 255, vdd33_const is invalid. ESP8266EX RF Calibration and optimization process is implemented via the default value 3.3V.*

5. Electrical Specifications

5.1. Electrical Characteristics

Table 5-1. Electrical Characteristics

Parameters		Conditions	Min	Typical	Max	Unit
Operating Temperature Range		-	–40	Normal	125	°C
Maximum Soldering Temperature		IPC/JEDEC J-STD-020	-	-	260	°C
Working Voltage Value		-	2.5	3.3	3.6	V
I/O	V_{IL}	-	–0.3	-	$0.25V_{IO}$	V
	V_{IH}		$0.75V_{IO}$		3.6	
	V_{OL}	-	-	-	$0.1V_{IO}$	
	V_{OH}		$0.8V_{IO}$		-	
	I_{MAX}	-	-	-	12	mA
Electrostatic Discharge (HBM)		TAMB=25°C	-	-	2	KV
Electrostatic Discharge (CDM)		TAMB=25°C	-	-	0.5	KV

5.2. RF Power Consumption

Unless otherwise specified, the power consumption measurements are taken with a 3.0V supply at 25°C of ambient temperature. All transmitters' measurements are based on a 50% duty cycle.

Table 5-2. Power Consumption

Parameters	Min	Typical	Max	Unit
TX 802.11b, CCK 11Mbps, P_{OUT}=+17 dBm	-	170	-	mA
TX 802.11g, OFDM 54Mbps, P_{OUT}=+15 dBm	-	140	-	mA
TX 802.11n, MCS7, P_{OUT}=+13dBm	-	120	-	mA
Rx 802.11b, 1024 bytes packet length , –80 dBm	-	50	-	mA
Rx 802.11g, 1024 bytes packet length, –70 dBm	-	56	-	mA
Rx 802.11n, 1024 bytes packet length, –65 dBm	-	56	-	mA

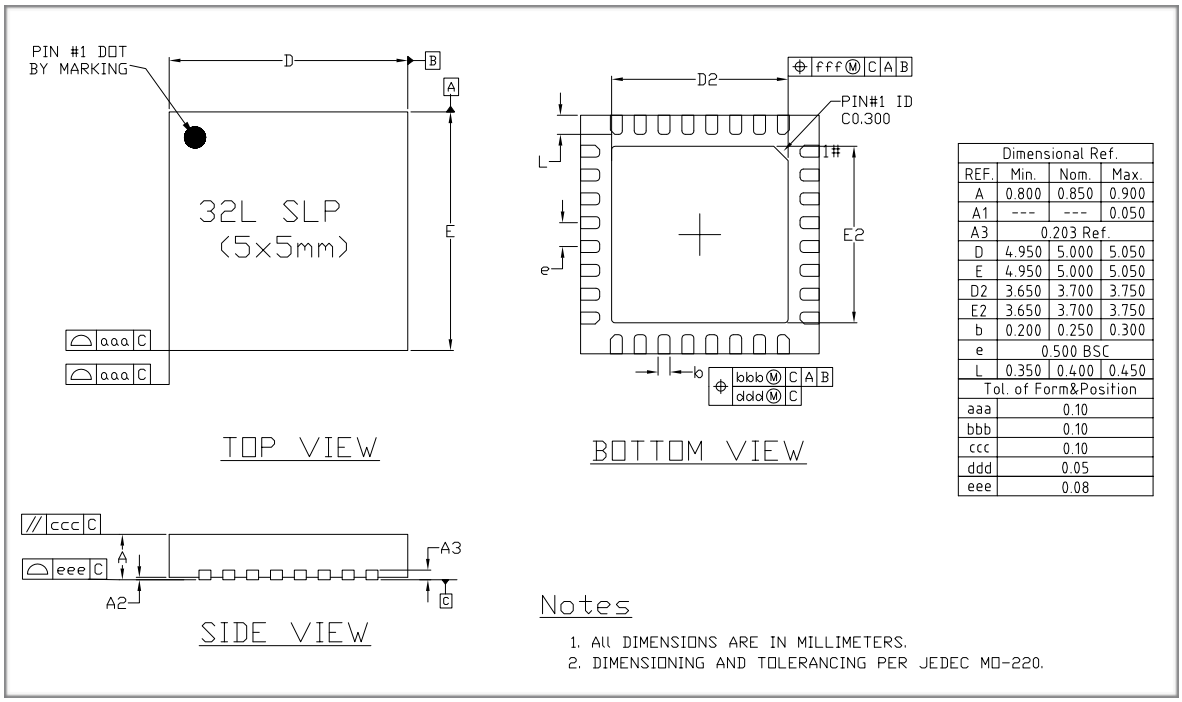
5.3. Wi-Fi Radio Characteristics

The following data are from tests conducted at room temperature, with a 3.3V power supply.

Table 5-3. Wi-Fi Radio Characteristics

Parameters	Min	Typical	Max	Unit
Input frequency	2412	-	2484	MHz
Output impedance	-	39+j6	-	Ω
Output power of PA for 72.2 Mbps	15.5	16.5	17.5	dBm
Output power of PA for 11b mode	19.5	20.5	21.5	dBm
Sensitivity				
DSSS, 1 Mbps	-	–98	-	dBm
CCK, 11 Mbps	-	–91	-	dBm
6 Mbps (1/2 BPSK)	-	–93	-	dBm
54 Mbps (3/4 64-QAM)	-	–75	-	dBm
HT20, MCS7 (65 Mbps, 72.2 Mbps)	-	–72	-	dBm
Adjacent Channel Rejection				
OFDM, 6 Mbps	-	37	-	dB
OFDM, 54 Mbps	-	21	-	dB
HT20, MCS0	-	37	-	dB
HT20, MCS7	-	20	-	dB

6. Package Information



Dimensional Ref.			
REF.	Min.	Nom.	Max.
A	0.800	0.850	0.900
A1	---	---	0.050
A3	0.203 Ref.		
D	4.950	5.000	5.050
E	4.950	5.000	5.050
D2	3.650	3.700	3.750
E2	3.650	3.700	3.750
b	0.200	0.250	0.300
e	0.500 BSC		
L	0.350	0.400	0.450
Tol. of Form&Position			
aaa	0.10		
bbb	0.10		
ccc	0.10		
ddd	0.05		
eee	0.08		

Figure 6-1. ESP8266EX Package

I. Appendix - Pin List

For detailed pin information, please see *ESP8266 Pin List*.

- Digital Die Pin List
- Buffer Sheet
- Register List
- Strapping List

> 📖 ***Notes:***
>
> - *INST_NAME refers to the IO_MUX REGISTER defined in **eagle_soc.h**, for example MTDI_U refers to PERIPHS_IO_MUX_MTDI_U.*
> - *Net Name refers to the pin name in schematic.*
> - *Function refers to the multifunction of each pin pad.*
> - *Function number 1 ~ 5 correspond to FUNCTION 0 ~ 4 in SDK. For example, set MTDI to GPIO12 as follows.*
>   - `#define FUNC_GPIO12  3 //defined in eagle_soc.h`
>   - `PIN_FUNC_SELECT(PERIPHS_IO_MUX_MTDI_U,FUNC_GPIO12)`

II. Appendix - Learning Resources

II.1. Must-Read Documents

- *ESP8266 Quick Start Guide*

  Description: This document is a quick user guide to getting started with ESP8266. It includes an introduction to the ESP-LAUNCHER, instructions on how to download firmware to the board and run it, how to compile the AT application, as well as the structure and debugging method of RTOS SDK. Basic documentation and other related resources for the ESP8266 are also provided.
- *ESP8266 SDK Getting Started Guide*

  Description: This document takes ESP-LAUNCHER and ESP-WROOM-02 as examples of how to use the ESP8266 SDK. The contents include preparations before compilation, SDK compilation and firmware download.
- *ESP8266 Pin List*

  Description: This link directs you to a list containing the type and function of every ESP8266 pin.
- *ESP8266 Hardware Design Guideline*

  Description: This document provides a technical description of the ESP8266 series of products, including ESP8266EX, ESP-LAUNCHER and ESP-WROOM.
- *ESP8266 Hardware Matching Guide*

  Description: This document introduces the frequency offset tuning and antenna impedance matching for ESP8266 in order to achieve optimal RF performance.
- *ESP8266 Technical Reference*

  Description: This document provides an introduction to the interfaces integrated on ESP8266. Functional overview, parameter configuration, function description, application demos and other pieces of information are included.
- *ESP8266 Hardware Resources*

  Description: This zip package includes manufacturing BOMs, schematics and PCB layouts of ESP8266 boards and modules.
- *FAQ*

II.2. Must-Have Resources

- *ESP8266 SDKs*

Description: This webpage provides links both to the latest version of the ESP8266 SDK and the older ones.

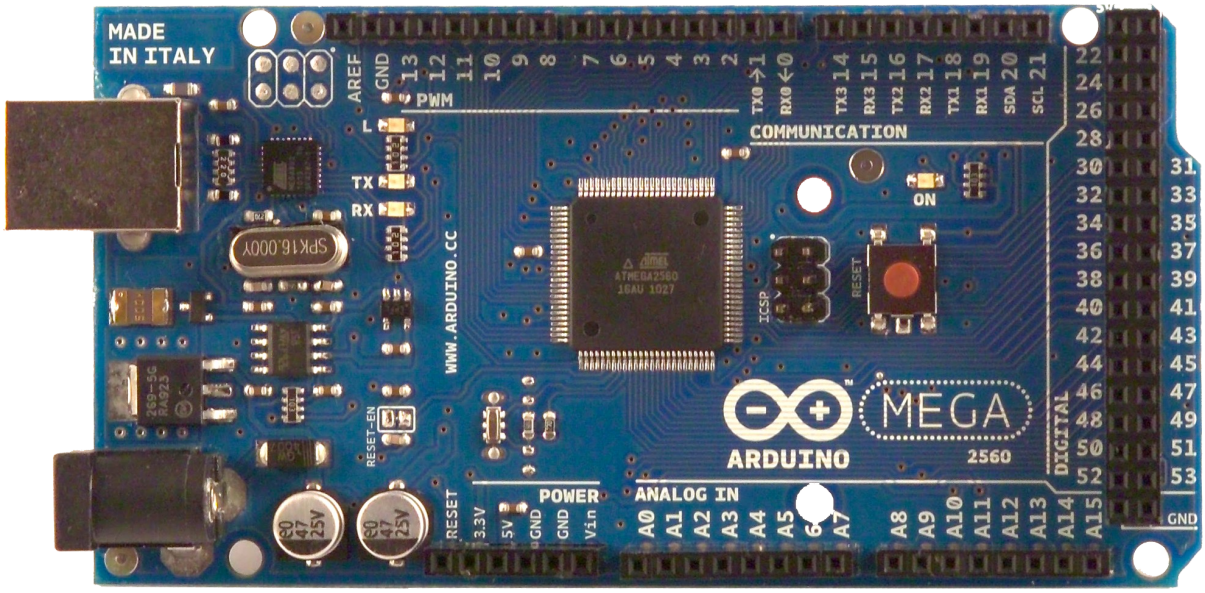
- *ESP8266 Tools*

  Description: This webpage provides links to both the ESP8266 flash download tools and the ESP8266 performance evaluation tools.
- *ESP8266 Apps*
- *ESP8266 Certification and Test Guide*
- *ESP8266 BBS*
- *ESP8266 Resources*

Espressif IOT Team

www.espressif.com

Arduino MEGA 2560



Product Overview

The Arduino Mega 2560 is a microcontroller board based on the ATmega2560 (datasheet). It has 54 digital input/output pins (of which 14 can be used as PWM outputs), 16 analog inputs, 4 UARTs (hardware serial ports), a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started. The Mega is compatible with most shields designed for the Arduino Duemilanove or Diecimila.

Index

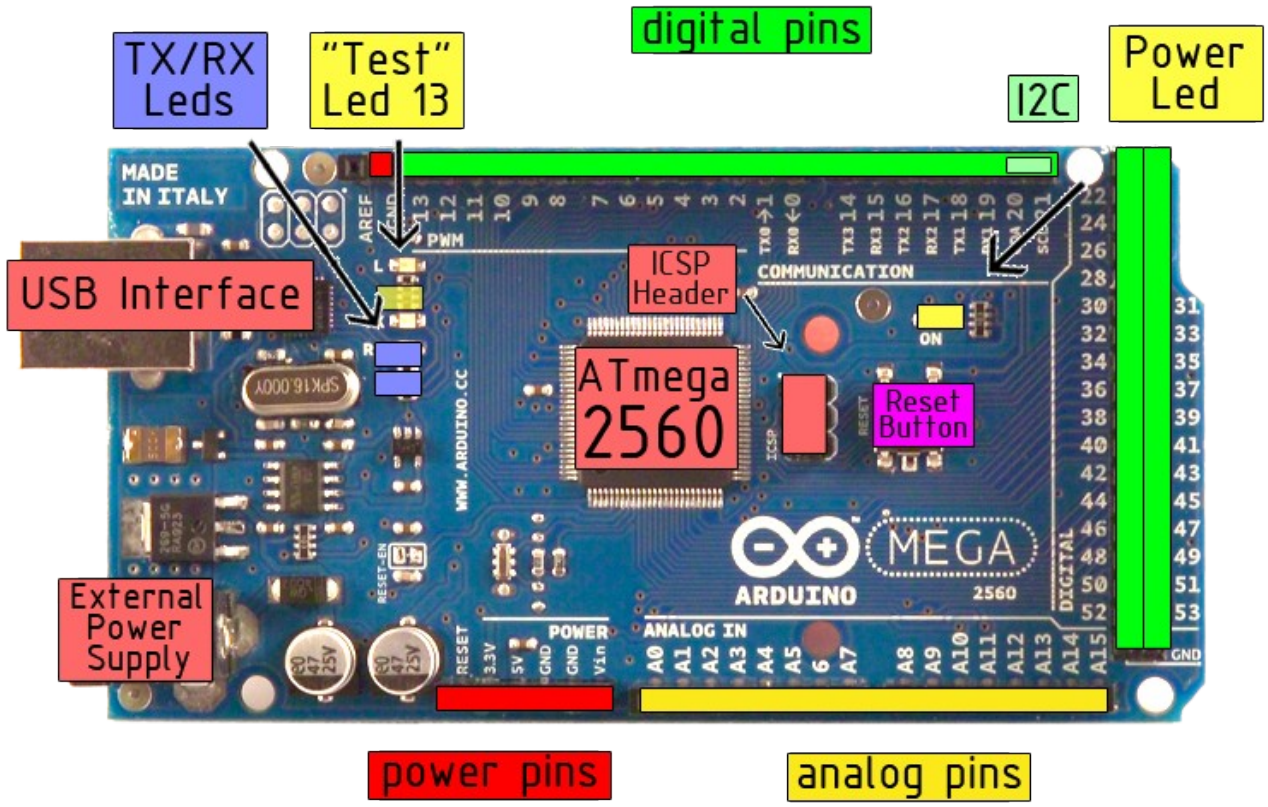
Technical Specification

EAGLE files: arduino-mega2560-reference-design.zip Schematic: arduino-mega2560-schematic.pdf

Summary

Microcontroller	ATmega2560
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	54 (of which 14 provide PWM output)
Analog Input Pins	16
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	256 KB of which 8 KB used by bootloader
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz

the board



Power

The Arduino Mega2560 can be powered via the USB connection or with an external power supply. The power source is selected automatically. External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm center-positive plug into the board's power jack. Leads from a battery can be inserted in the Gnd and Vin pin headers of the POWER connector.

The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may be unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts.

The Mega2560 differs from all preceding boards in that it does not use the FTDI USB-to-serial driver chip. Instead, it features the Atmega8U2 programmed as a USB-to-serial converter.

The power pins are as follows:

- **VIN.** The input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.
- **5V.** The regulated power supply used to power the microcontroller and other components on the board. This can come either from VIN via an on-board regulator, or be supplied by USB or another regulated 5V supply.
- **3V3.** A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50 mA.
- **GND.** Ground pins.

Memory

The ATmega2560 has 256 KB of flash memory for storing code (of which 8 KB is used for the bootloader), 8 KB of SRAM and 4 KB of EEPROM (which can be read and written with the EEPROM library).

Input and Output

Each of the 54 digital pins on the Mega can be used as an input or output, using pinMode(), digitalWrite(), and digitalRead() functions. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 kOhms. In addition, some pins have specialized functions:

- **Serial: 0 (RX) and 1 (TX); Serial 1: 19 (RX) and 18 (TX); Serial 2: 17 (RX) and 16 (TX); Serial 3: 15 (RX) and 14 (TX).** Used to receive (RX) and transmit (TX) TTL serial data. Pins 0 and 1 are also connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip .
- **External Interrupts: 2 (interrupt 0), 3 (interrupt 1), 18 (interrupt 5), 19 (interrupt 4), 20 (interrupt 3), and 21 (interrupt 2).** These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the attachInterrupt() function for details.
- **PWM: 0 to 13.** Provide 8-bit PWM output with the analogWrite() function.
- **SPI: 50 (MISO), 51 (MOSI), 52 (SCK), 53 (SS).** These pins support SPI communication, which, although provided by the underlying hardware, is not currently included in the Arduino language. The SPI pins are also broken out on the ICSP header, which is physically compatible with the Duemilanove and Diecimila.
- **LED: 13.** There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.
- **I^2C: 20 (SDA) and 21 (SCL).** Support I^2C (TWI) communication using the Wire library (documentation on the Wiring website). Note that these pins are not in the same location as the I^2C pins on the Duemilanove.

The Mega2560 has 16 analog inputs, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though is it possible to change the upper end of their range using the AREF pin and analogReference() function.

There are a couple of other pins on the board:

- **AREF.** Reference voltage for the analog inputs. Used with analogReference().
- **Reset.** Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

Communication

The Arduino Mega2560 has a number of facilities for communicating with a computer, another Arduino, or other microcontrollers. The ATmega2560 provides four hardware UARTs for TTL (5V) serial communication. An ATmega8U2 on the board channels one of these over USB and provides a virtual com port to software on the computer (Windows machines will need a .inf file, but OSX and Linux machines will recognize the board as a COM port automatically. The Arduino software includes a serial monitor which allows simple textual data to be sent to and from the board. The RX and TX LEDs on the board will flash when data is being transmitted via the ATmega8U2 chip and USB connection to the computer (but not for serial communication on pins 0 and 1).

A SoftwareSerial library allows for serial communication on any of the Mega's digital pins.

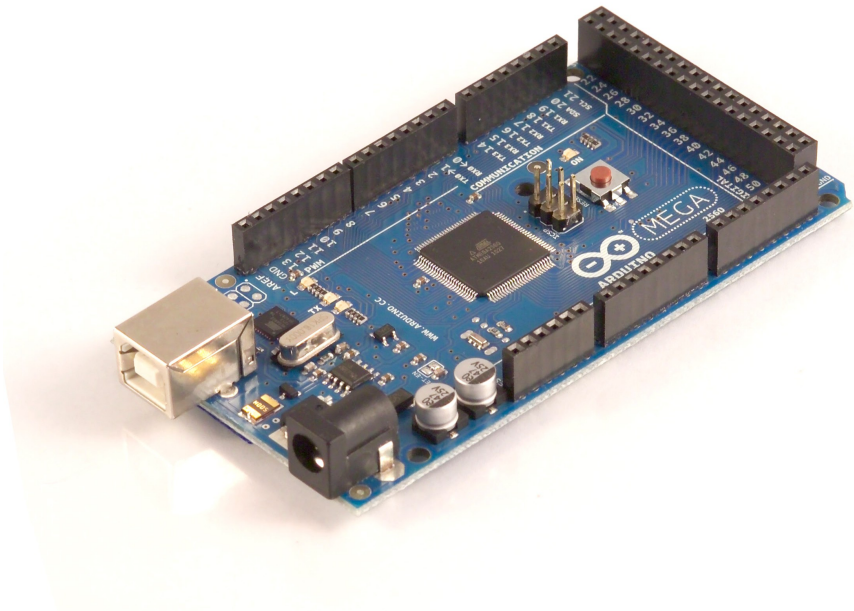
The ATmega2560 also supports I2C (TWI) and SPI communication. The Arduino software includes a Wire library to simplify use of the I2C bus; see the documentation on the Wiring website for details. To use the SPI communication, please see the ATmega2560 datasheet.

Programming

The Arduino Mega2560 can be programmed with the Arduino software (download). For details, see the reference and tutorials.

The Atmega2560 on the Arduino Mega comes preburned with a bootloader that allows you to upload new code to it without the use of an external hardware programmer. It communicates using the original STK500 protocol (reference, C header files).

You can also bypass the bootloader and program the microcontroller through the ICSP (In-Circuit Serial Programming) header; see these instructions for details.



Automatic (Software) Reset

Rather then requiring a physical press of the reset button before an upload, the Arduino Mega2560 is designed in a way that allows it to be reset by software running on a connected computer. One of the hardware flow control lines (DTR) of the ATmega8U2 is connected to the reset line of the ATmega2560 via a 100 nanofarad capacitor. When this line is asserted (taken low), the reset line drops long enough to reset the chip. The Arduino software uses this capability to allow you to upload code by simply pressing the upload button in the Arduino environment. This means that the bootloader can have a shorter timeout, as the lowering of DTR can be well-coordinated with the start of the upload.

This setup has other implications. When the Mega2560 is connected to either a computer running Mac OS X or Linux, it resets each time a connection is made to it from software (via USB). For the following half-second or so, the bootloader is running on the Mega2560. While it is programmed to ignore malformed data (i.e. anything besides an upload of new code), it will intercept the first few bytes of data sent to the board after a connection is opened. If a sketch running on the board receives one-time configuration or other data when it first starts, make sure that the software with which it communicates waits a second after opening the connection and before sending this data.

The Mega contains a trace that can be cut to disable the auto-reset. The pads on either side of the trace can be soldered together to re-enable it. It's labeled "RESET-EN". You may also be able to disable the auto-reset by connecting a 110 ohm resistor from 5V to the reset line; see this forum thread for details.

USB Overcurrent Protection

The Arduino Mega has a resettable polyfuse that protects your computer's USB ports from shorts and overcurrent. Although most computers provide their own internal protection, the fuse provides an extra layer of protection. If more than 500 mA is applied to the USB port, the fuse will automatically break the connection until the short or overload is removed.

Physical Characteristics and Shield Compatibility

The maximum length and width of the Mega PCB are 4 and 2.1 inches respectively, with the USB connector and power jack extending beyond the former dimension. Three screw holes allow the board to be attached to a surface or case. Note that the distance between digital pins 7 and 8 is 160 mil (0.16"), not an even multiple of the 100 mil spacing of the other pins.

The Mega is designed to be compatible with most shields designed for the Diecimila or Duemilanove. Digital pins 0 to 13 (and the adjacent AREF and GND pins), analog inputs 0 to 5, the power header, and ICSP header are all in equivalent locations. Further the main UART (serial port) is located on the same pins (0 and 1), as are external interrupts 0 and 1 (pins 2 and 3 respectively). SPI is available through the ICSP header on both the Mega and Duemilanove / Diecimila. **Please note that I^2C is not located on the same pins on the Mega (20 and 21) as the Duemilanove / Diecimila (analog inputs 4 and 5).**

How to use Arduino

Arduino can sense the environment by receiving input from a variety of sensors and can affect its surroundings by controlling lights, motors, and other actuators. The microcontroller on the board is programmed using the Arduino programming language (based on Wiring) and the Arduino development environment (based on Processing). Arduino projects can be stand-alone or they can communicate with software on running on a computer (e.g. Flash, Processing, MaxMSP).

Arduino is a cross-platoform program. You'll have to follow different instructions for your personal OS. Check on the Arduino site for the latest instructions. *http://arduino.cc/en/Guide/HomePage*

Linux Install

Windows Install

Mac Install

Once you have downloaded/unzipped the arduino IDE, you can Plug the Arduino to your PC via USB cable.

Blink led

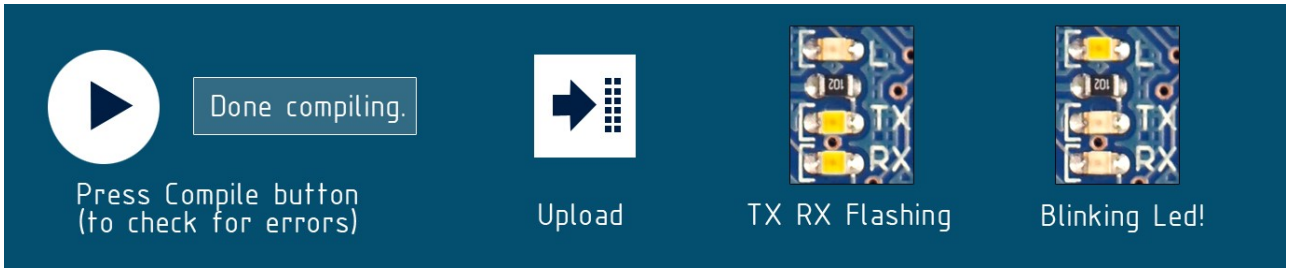
Now you're actually ready to "burn" your first program on the arduino board. To select "blink led", the physical translation of the well known programming "hello world", select

File>Sketchbook>
Arduino-0017>Examples>
Digital>Blink

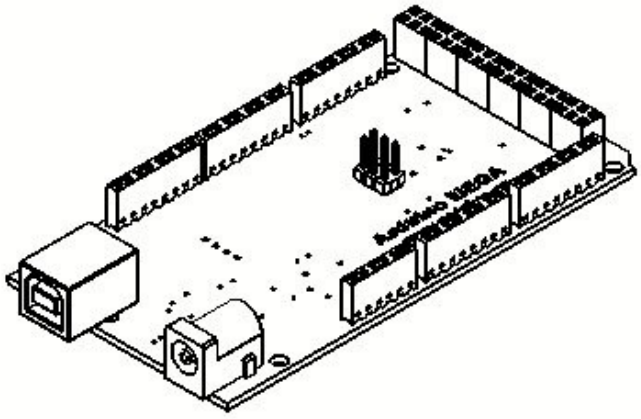
Once you have your skecth you'll see something very close to the screenshot on the right.

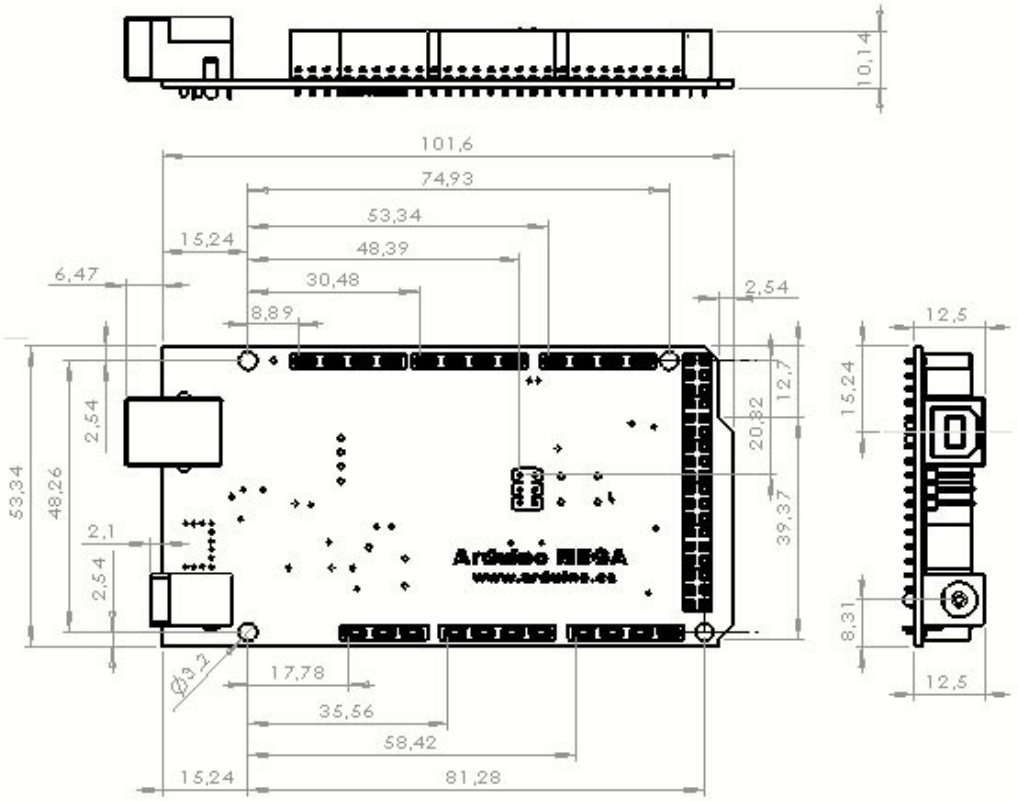
In **Tools>Board** select MEGA

Now you have to go to
Tools>SerialPort
and select the right serial port, the one arduino is attached to.



Dimensioned Drawing





Terms & Conditions

1. Warranties

1.1 The producer warrants that its products will conform to the Specifications. This warranty lasts for one (1) years from the date of the sale. The producer shall not be liable for any defects that are caused by neglect, misuse or mistreatment by the Customer, including improper installation or testing, or for any products that have been altered or modified in any way by a Customer. Moreover, The producer shall not be liable for any defects that result from Customer's design, specifications or instructions for such products. Testing and other quality control techniques are used to the extent the producer deems necessary.

1.2 If any products fail to conform to the warranty set forth above, the producer's sole liability shall be to replace such products. The producer's liability shall be limited to products that are determined by the producer not to conform to such warranty. If the producer elects to replace such products, the producer shall have a reasonable time to replacements. Replaced products shall be warranted for a new full warranty period.

1.3 EXCEPT AS SET FORTH ABOVE, PRODUCTS ARE PROVIDED "AS IS" AND "WITH ALL FAULTS." THE PRODUCER DISCLAIMS ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, REGARDING PRODUCTS, INCLUDING BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE

1.4 Customer agrees that prior to using any systems that include the producer products, Customer will test such systems and the functionality of the products as used in such systems. The producer may provide technical, applications or design advice, quality characterization, reliability data or other services. Customer acknowledges and agrees that providing these services shall not expand or otherwise alter the producer's warranties, as set forth above, and no additional obligations or liabilities shall arise from the producer providing such services.

1.5 The Arduino™ products are not authorized for use in safety-critical applications where a failure of the product would reasonably be expected to cause severe personal injury or death. Safety-Critical Applications include, without limitation, life support devices and systems, equipment or systems for the operation of nuclear facilities and weapons systems. Arduino™ products are neither designed nor intended for use in military or aerospace applications or environments and for automotive applications or environment. Customer acknowledges and agrees that any such use of Arduino™ products which is solely at the Customer's risk, and that Customer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

1.6 Customer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products and any use of Arduino™ products in Customer's applications, notwithstanding any applications-related information or support that may be provided by the producer.

2. Indemnification

The Customer acknowledges and agrees to defend, indemnify and hold harmless the producer from and against any and all third-party losses, damages, liabilities and expenses it incurs to the extent directly caused by: (i) an actual breach by a Customer of the representation and warranties made under this terms and conditions or (ii) the gross negligence or willful misconduct by the Customer.

3. Consequential Damages Waiver

In no event the producer shall be liable to the Customer or any third parties for any special, collateral, indirect, punitive, incidental, consequential or exemplary damages in connection with or arising out of the products provided hereunder, regardless of whether the producer has been advised of the possibility of such damages. This section will survive the termination of the warranty period.

4. Changes to specifications

The producer may make changes to specifications and product descriptions at any time, without notice. The Customer must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." The producer reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The product information on the Web Site or Materials is subject to change without notice. Do not finalize a design with this information.

Enviromental Policies

The producer of Arduino™ has joined the Impatto Zero® policy of LifeGate.it. For each Arduino board produced is created / looked after half squared Km of Costa Rica's forest's.

ESP-01 WiFi Module

Version1.0

sherry@aithinker.com

Disclaimer and Copyright Notice.

Notice

Product version upgrades or other reasons, possible changes in the contents of this manual. AI-Thinker reserves in the absence of any notice or indication of the circumstances the right to modify the content of this manual. This manual is used only as a guide, Ai-thinker make every effort to provide accurate information in this manual, but Ai-thinker does not ensure that manual content without error, in this manual all statements, information and advice nor does it constitute any express or implied warranty.

Table of Contents

1. Preambles

ESP-01 WiFi module is developed by Ai-thinker Team. core processor ESP8266 in smaller sizes of the module encapsulates Tensilica L106 integrates industry-leading ultra low power 32-bit MCU micro, with the 16-bit short mode, Clock speed support 80 MHz, 160 MHz, supports the RTOS, integrated Wi-Fi MAC/BB/RF/PA/LNA, on-board antenna.

The module supports standard IEEE802.11 b/g/n agreement, complete TCP/IP protocol stack. Users can use the add modules to an existing device networking, or building a separate network controller.

ESP8266 is high integration wireless SOCs, designed for space and power constrained mobile platform designers. It provides unsurpassed ability to embed Wi-Fi capabilities within other systems, or to function as a standalone application, with the lowest cost, and minimal space requirement.

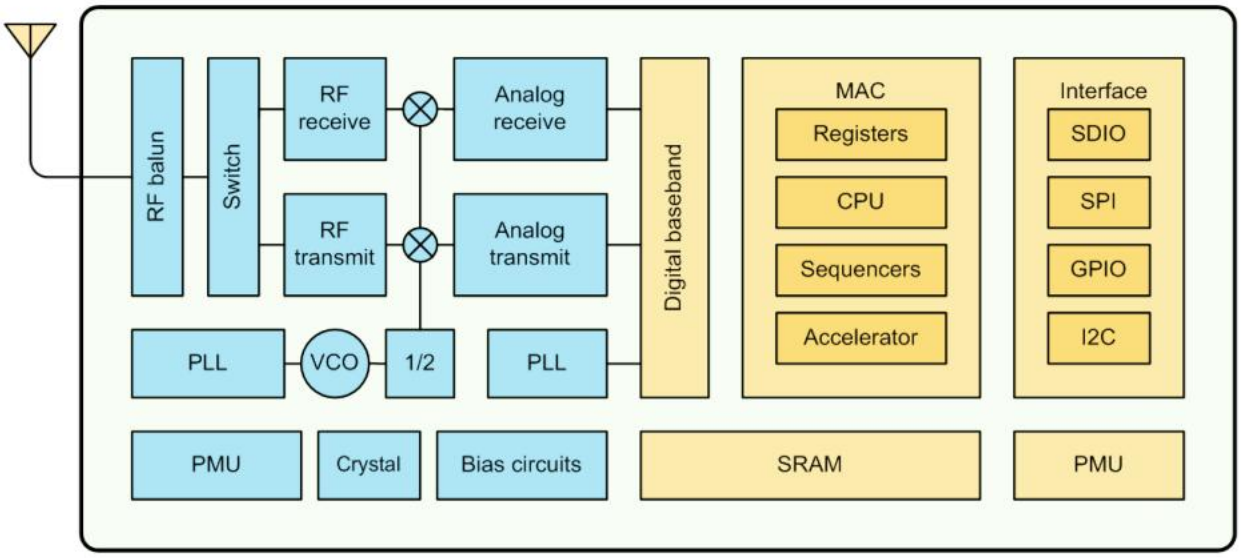


Figure 1 ESP8266EX Block Diagram

ESP8266EX offers a complete and self-contained Wi-Fi networking solution; it can be used to host the application or to offload Wi-Fi networking functions from another application processor.

When ESP8266EX hosts the application, it boots up directly from an external flash. In has integrated cache to improve the performance of the system in such applications.

Alternately, serving as a Wi-Fi adapter, wireless internet access can be added to any micro controllerbased design with simple connectivity (SPI/SDIO or I2C/UART interface).

ESP8266EX is among the most integrated WiFi chip in the industry; it integrates the antenna switches, RF balun, power amplifier, low noise receive amplifier, filters, power management modules, it requires minimal external circuitry, and the entire solution, including front-end module, is designed to occupy minimal PCB area.

ESP8266EX also integrates an enhanced version of Tensilica's L106 Diamond series 32-bit processor, with on-chip SRAM, besides the Wi-Fi functionalities. ESP8266EX is often integrated with external sensors and other application specific devices through its GPIOs; codes for such applications are provided in examples in the SDK.

Espressif Systems' Smart Connectivity Platform (ESCP) demonstrates sophisticated system-level features include fast sleep/wake context switching for energy-efficient VoIP, adaptive radio biasing. for low-power operation, advance signal processing, and spur cancellation and radio co-existence features for common cellular, Bluetooth, DDR, LVDS, LCD interference mitigation.

1.1. Features

- 802.11 b/g/n
- Integrated low power 32-bit MCU
- Integrated 10-bit ADC
- Integrated TCP/IP protocol stack
- Integrated TR switch, balun, LNA, power amplifier and matching network
- Integrated PLL, regulators, and power management units
- Supports antenna diversity
- Wi-Fi 2.4 GHz, support WPA/WPA2
- Support STA/AP/STA+AP operation modes
- Support Smart Link Function for both Android and iOS devices
- Support Smart Link Function for both Android and iOS devices
- SDIO 2.0, (H) SPI, UART, I2C, I2S, IRDA, PWM, GPIO

- STBC, 1x1 MIMO, 2x1 MIMO
- A-MPDU & A-MSDU aggregation and 0.4s guard interval
- Deep sleep power <10uA, Power down leakage current < 5uA
- Wake up and transmit packets in < 2ms
- Standby power consumption of < 1.0mW (DTIM3)
- +20dBm output power in 802.11b mode
- Operating temperature range -40C ~ 125C

1.2. Parameters

Table 1 below describes the major parameters.

Table 1 Parameters

Categories	Items	Values
WiFi Paramters	WiFi Protocles	802.11 b/g/n
	Frequency Range	2.4GHz-2.5GHz (2400M-2483.5M)
Hardware Paramaters	Peripheral Bus	UART/HSPI/I2C/I2S/Ir Remote Contorl
		GPIO/PWM
	Operating Voltage	3.0~3.6V
	Operating Current	Average value: 80mA
	Operating Temperature Range	-40°~125°
	Ambient Temperature Range	Normal temperature
	Package Size	14.3mm*24.8mm*3mm
	External Interface	N/A
Software Parameters	Wi-Fi mode	station/softAP/SoftAP+station
	Security	WPA/WPA2
	Encryption	WEP/TKIP/AES
	Firmware Upgrade	UART Download / OTA (via network) / download and write firmware via host
	Ssoftware Development	Supports Cloud Server Development / SDK for custom firmware development
	Network Protocols	IPv4, TCP/UDP/HTTP/FTP
	User Configuration	AT Instruction Set, Cloud Server, Android/iOS App

2. Pin Descriptions

There are altogether 8 pin counts, the definitions of which are described in Table 2 below。

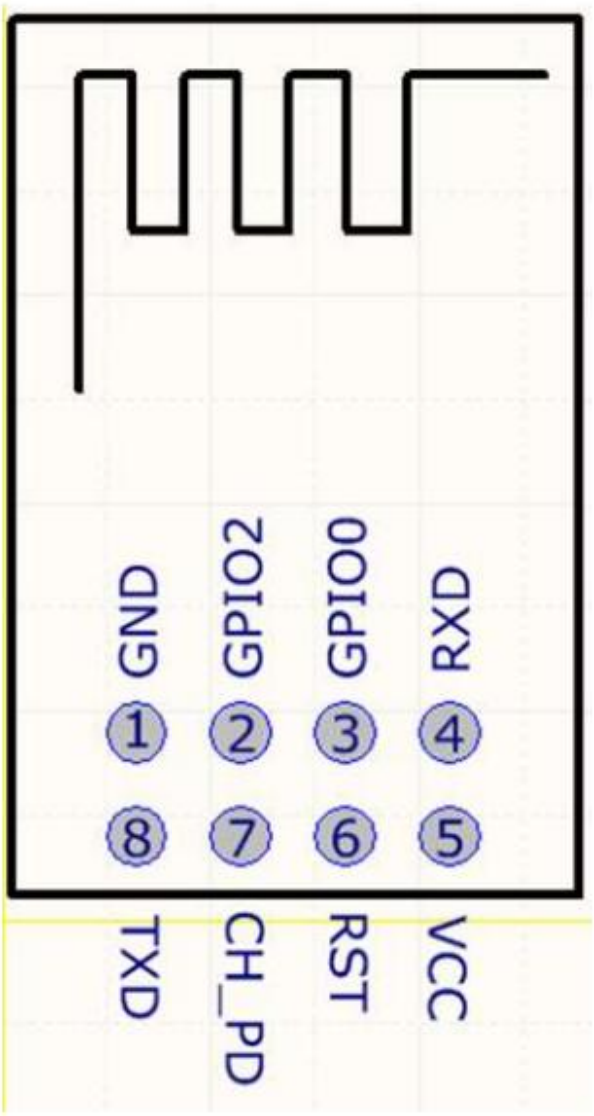


Table 2 ESP-01 Pin design

Table 2 Pin Descriptions

NO.	Pin Name	Function
1	GND	GND
2	GPIO2	GPIO,Internal Pull-up
3	GPIO0	GPIO,Internal Pull-up
4	RXD	UART0,data received pin RXD
5	VCC	3.3V power supply (VDD)
6	RST	1) External reset pin, active low 2) Can loft or external MCU ,
7	CH_PD	Chip enable pin. Active high
8	TXD	UART0,data send pin RXD

Table 3 Pin Mode

Mode	GPIO15	GPIO0	GPIO2
UART	Low	Low	High
Flash Boot	Low	High	High

Table 4 Receiver Sensitivity

Parameters	Min	Typical	Max	Unit
Input frequency	2412		2484	MHz
Input impedance		50		Ω
Input reflection			-10	dB
Output power of PA for 72.2Mbps	15.5	16.5	17.5	dBm
Output power of PA for 11b mode	19.5	20.5	21.5	dBm
Sensitivity				
DSSS, 1Mbps		-98		dBm
CCK, 11Mbps		-91		dBm
6Mbps (1/2 BPSK)		-93		dBm
54Mbps (3/4 64-QAM)		-75		dBm
HT20, MCS7 (65Mbps, 72.2Mbps)		-72		dBm
Adjacent Channel Rejection				
OFDM, 6Mbps		37		dB
OFDM, 54Mbps		21		dB
HT20, MCS0		37		dB
HT20, MCS7		20		dB

3. Packaging and Dimension

The external size of the module is 14.3mm*24.8mm*3mm, as is illustrated in Figure 3 below. The type of flash integrated in this module is an SPI flash, the capacity of which is 1 MB, and the package size of which is SOP-210mil. The antenna applied on this module is a 3DBi PCB-on-board antenna.

Figure 3 [Module Pin Counts, 8 pin, 14.3 mm *24.8 mm *3.0 mm]

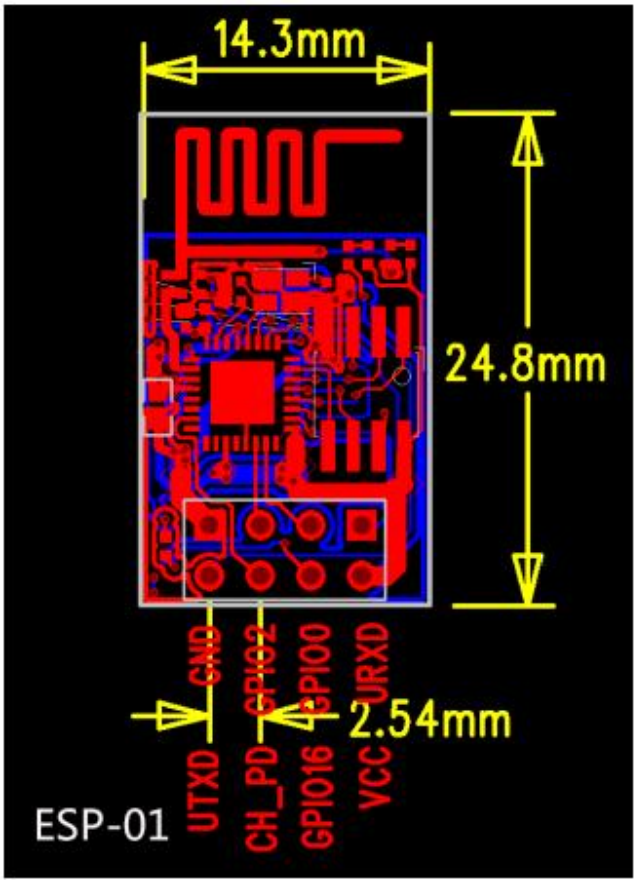


Figure 4 Top View of ESP-01 WiFi Module

Table 5 Dimension of ESP-01 WiFi Module

Length	Width	Height	PAD Size(Bottom)	Pin Pitch
14.3 mm	24.8 mm	3 mm	0.9 mm x 1.7 mm	2.54 mm

4. Functional Descriptions

4.1. MCU

ESP8266EX is embedded with Tensilica L106 32-bit micro controller (MCU), which features extra low power consumption and 16-bit RSIC. The CPU clock speed is 80MHz. It can also reach a maximum value of 160MHz. ESP8266EX is often integrated with external sensors and other specific devices through its GPIOs; codes for such applications are provided in examples in the SDK.

4.2. Memory Organization

4.2.1. Internal SRAM and ROM

ESP8266EX WiFi SoC is embedded with memory controller, including SRAM and ROM. MCU can visit the memory units through iBus, dBus, and AHB interfaces. All memory units can be visited upon request, while a memory arbiter will decide the running sequence according to the time when these requests are received by the processor.

According to our current version of SDK provided, SRAM space that is available to users is assigned as below:

▪RAM size < 36kB, that is to say, when ESP8266EX is working under the station mode and is connected to the router, programmable space accessible to user in heap and data section is around 36kB.)

▪ There is no programmable ROM in the SoC, therefore, user program must be stored in an external SPI flash.

4.2.2. External SPI Flash

This module is mounted with an 1 MB external SPI flash to store user programs. If larger definable storage space is required, a SPI flash with larger memory size is preferred. Theoretically speaking, up to 16 MB memory capacity can be supported.

Suggested SPI Flash memory capacity:

▪OTA is disabled: the minimum flash memory that can be supported is 512 kB;

▪OTA is enabled: the minimum flash memory that can be supported is 1 MB.

Several SPI modes can be supported, including Standard SPI, Dual SPI, and Quad SPI.

Therefore, please choose the correct SPI mode when you are downloading into the flash, otherwise firmwares/programs that you downloaded may not work in the right way.

4.3. Crystal

Currently, the frequency of crystal oscillators supported include 40MHz, 26MHz and 24MHz. The accuracy of crystal oscillators applied should be ±10PPM, and the operating temperature range should be between -20°C and 85°C.

When using the downloading tools, please remember to select the right crystal oscillator type. In circuit design, capacitors C1 and C2, which are connected to the earth, are added to the input and output terminals of the crystal oscillator respectively. The values of the two capacitors can be flexible, ranging from 6pF to 22pF, however, the specific capacitive values of C1 and C2 depend on further testing and adjustment on the overall performance of the whole circuit. Normally, the capacitive values of C1 and C2 are within 10pF if the crystal oscillator frequency is 26MHz, while the values of C1 and C2 are 10pF<C1, C2<22pF if the crystal oscillator frequency is 40MHz.

4.4. Interfaces

Table 6 Descriptions of Interfaces

Interface	Pin Name	Description
HSPI	IO12(MISO) IO13(MOSI) IO14(CLK) IO15(CS)	SPI Flash 2, display screen, and MCU can be connected using HSPI interface.
PWM	IO12(R) IO15(G) IO13(B)	Currently the PWM interface has four channels, but users can extend the channels according to their own needs. PWM interface can be used to control LED lights, buzzers, relays, electronic machines, and so on.
IR Remote Control	IO14(IR_T) IO5(IR_R)	The functionality of Infrared remote control interface can be implemented via software programming. NEC coding, modulation, and demodulation are used by this interface. The frequency of modulated carrier signal is 38KHz.
ADC	TOUT	ESP8266EX integrates a 10-bit analog ADC. It can be used to test the power-supply voltage of VDD3P3 (Pin3 and Pin4) and the input power voltage of TOUT (Pin 6). However, these two functions cannot be used simultaneously. This interface is typically used in sensor products.
I2C	IO14(SCL) IO2(SDA)	I2C interface can be used to connect external sensor products and display screens, etc.

Interface	Pin Name	Description
UART	**UART0:** TXD (U0TXD) RXD (U0RXD) IO15 (RTS) IO13 (CTS) **UART1:** IO2(TXD)	Devices with UART interfaces can be connected with the module. Downloading: U0TXD+U0RXD or GPIO2+U0RXD Communicating: UART0: U0TXD, U0RXD, MTDO (U0RTS), MTCK (U0CTS) Debugging: UART1_TXD (GPIO2) can be used to print debugging information. By default, UART0 will output some printed information when the device is powered on and is booting up. If this issue exerts influence on some specific applications, users can exchange the inner pins of UART when initializing, that is to say, exchange U0TXD, U0RXD with U0RTS, U0CTS.
I2S	**I2S Input:** IO12 (I2SI_DATA) ; IO13 (I2SI_BCK); IO14 (I2SI_WS); **I2S Output:** IO15 (I2SO_BCK); IO3 (I2SO_DATA); IO2 (I2SO_WS).	I2S interface is mainly used for collecting, processing, and transmission of audio data.

4.5. Absolute Maximum Ratings

Table 7 Absolute Maximum Ratings

Rating	Condition	Value	Unit
Storage Temperature		-40 to 125	°C
Maximum Soldering Temperature		260	°C
Supply Voltage	IPC/JEDEC J-STD-020	+3.0 to +3.6	V

4.6. Recommended Operating Conditions

Table 8 Recommended Operating Conditions

Operating Condition	Symbol	Min	Typ	Max	Unit
Operating Temperature		-40	20	125	°C
Supply voltage	VDD	3.0	3.3	3.6	V

4.7. Digital Terminal Characteristics

Table 9 Digital Terminal Characteristics

Terminals	Symbol	Min	Typ	Max	Unit
Input logic level low	V_{IL}	-0.3		0.25VDD	V
Input logic level high	V_{IH}	0.75VDD		VDD+0.3	V
Output logic level low	V_{OL}	N		0.1VDD	V
Output logic level high	V_{OH}	0.8VDD		N	V

Note: Test conditions: VDD = 3.3V, Temperature = 20 ℃, if nothing special is stated.

5. RF Performance

Description	Min.	Typ.	Max	Unit
Input frequency	2400		2483.5	MHz
Input impedance		50		ohm
Input reflection			-10	dB
Output power of PA for 72.2Mbps	15.5	16.5	17.5	dBm
Output power of PA for 11b mode	19.5	20.5	21.5	dBm
Sensitivity				
CCK, 1Mbps		-98		dBm
CCK, 11Mbps		-91		dBm
6Mbps (1/2 BPSK)		-93		dBm
54Mbps (3/4 64-QAM)		-75		dBm
HT20, MCS7 (65Mbps, 72.2Mbps)		-72		dBm
Adjacent Channel Rejection				
OFDM, 6Mbps		37		dB
OFDM, 54Mbps		21		dB
HT20, MCS0		37		dB
HT20, MCS7		20		dB

Table 10 RF Performance

6. Power Consumption

Parameters	Min	Typical	Max	Unit
Tx802.11b, CCK 11Mbps, P OUT=+17dBm		170		mA
Tx 802.11g, OFDM 54Mbps, P OUT =+15dBm		140		mA
Tx 802.11n, MCS7, P OUT =+13dBm		120		mA
Rx 802.11b, 1024 bytes packet length , -80dBm		50		mA
Rx 802.11g, 1024 bytes packet length, -70dBm		56		mA
Rx 802.11n, 1024 bytes packet length, -65dBm		56		mA
Modem-Sleep①		15		mA
Light-Sleep②		0.9		mA
Deep-Sleep③		10		uA

Table 11 Power Consumption

❶Modem-Sleep requires the CPU to be working, as in PWM or I2S applications. According to 802.11 standards (like U-APSD), it saves power to shut down the Wi-Fi Modem circuit while maintaining a Wi-Fi connection with no data transmission. E.g. in DTIM3, to maintain a sleep 300mswake 3ms cycle to receive AP's Beacon packages, the current is about 15mA.

❷ During Light-Sleep, the CPU may be suspended in applications like Wi-Fi switch. Without data transmission, the Wi-Fi Modem circuit can be turned off and CPU suspended to save power according to the 802.11 standard (U-APSD). E.g. in DTIM3, to maintain a sleep 300ms-wake 3ms cycle to receive AP's Beacon packages, the current is about 0.9mA.

❸ Deep-Sleep does not require Wi-Fi connection to be maintained. For application with long time lags between data transmission, e.g. a temperature sensor that checks the temperature every 100s ,sleep 300s and waking up to connect to the AP (taking about 0.3~1s), the overall average current is less than 1mA.

7. Reflow Profile

Table 12 Instructions

T_S max to T_L (Ramp-up Rate)	3°C/second max
Preheat Temperature Min.(T_S Min.) Temperature Typical.(T_S Typ.) Temperature Min.(T_S Max.) Time(T_S)	 150°C 175°C 200°C 60~180 seconds
Ramp-up rate (T_L to T_P)	3°C/second max
Time Maintained Above: --Temperature(T_L)/Time(T_L)	 217°C/60~150 seconds
Peak Temperature(T_P)	260°C max. for 10 seconds
Target Peak Temperature (T_P Target)	260°C +0/-5°C
Time within 5°C of actual peak(t_P)	20~40 seconds
T_S max to T_L (Ramp-down Rate)	6°C/second max
Tune 25°C to Peak Temperature (t)	8 minutes max

8. Schematics

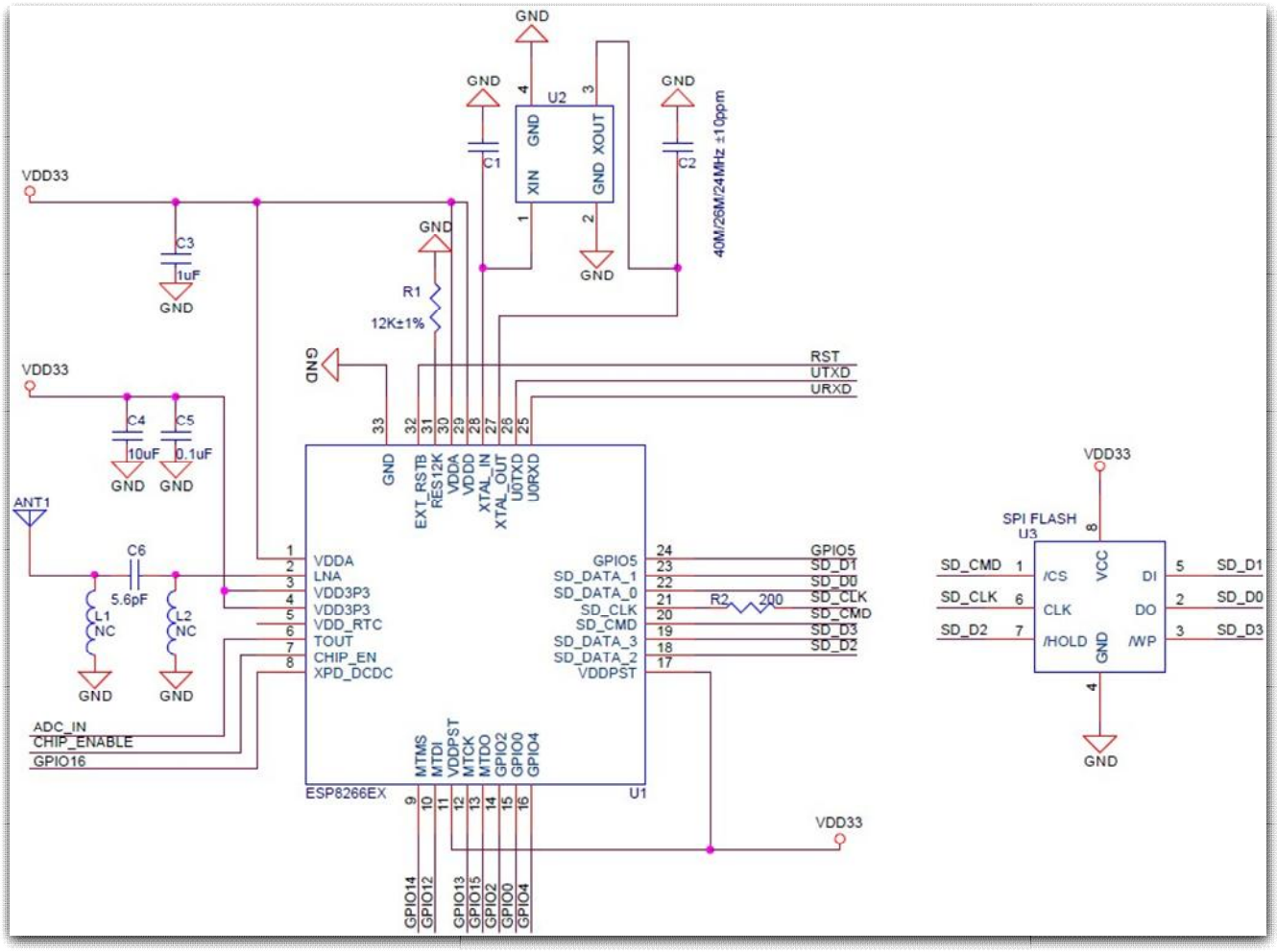
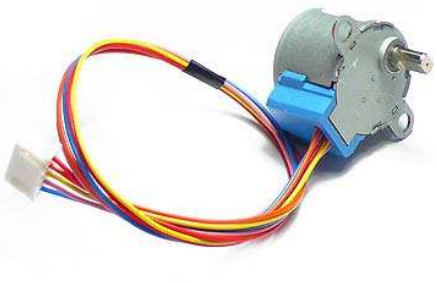
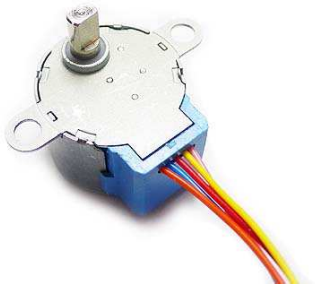


Figure 4 Schematics of Esp-01 WiFi Module

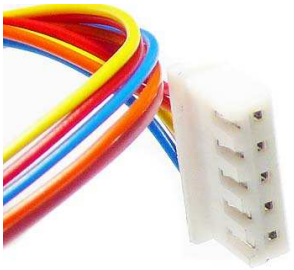
28BYJ-48 – 5V Stepper Motor

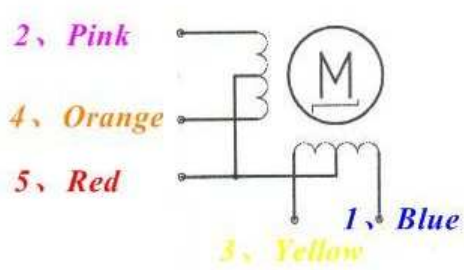
The 28BYJ-48 is a small stepper motor suitable for a large range of applications.

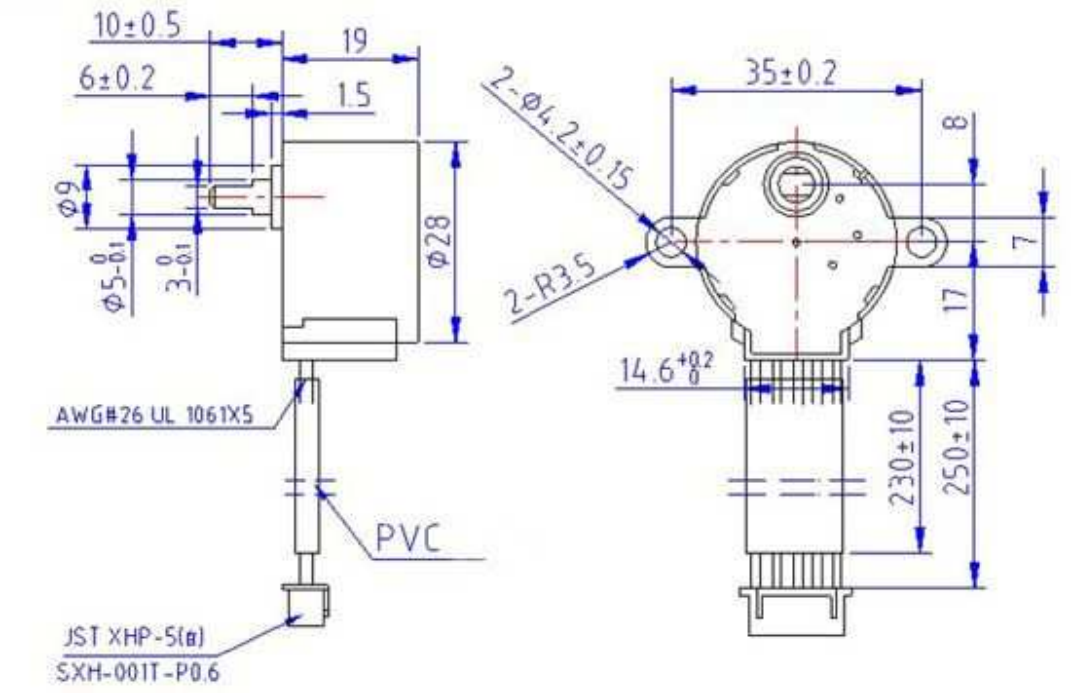




Rated voltage ：	5VDC
Number of Phase	4
Speed Variation Ratio	1/64
Stride Angle	5.625°/64
Frequency	100Hz
DC resistance	50Ω±7%(25℃)
Idle In-traction Frequency	> 600Hz
Idle Out-traction Frequency	> 1000Hz
In-traction Torque	>34.3mN.m(120Hz)
Self-positioning Torque	>34.3mN.m
Friction torque	600-1200 gf.cm
Pull in torque	300 gf.cm
Insulated resistance	>10MΩ(500V)
Insulated electricity power	600VAC/1mA/1s
Insulation grade	A
Rise in Temperature	<40K(120Hz)
Noise	<35dB(120Hz,No load,10cm)
Model	28BYJ-48 – 5V





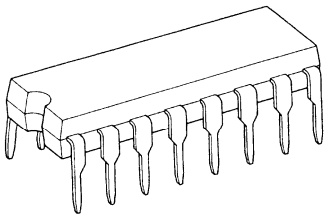


TOSHIBA Bipolar Digital Integrated Circuit Silicon Monolithic

ULN2003APG,ULN2003AFWG
ULN2004APG,ULN2004AFWG

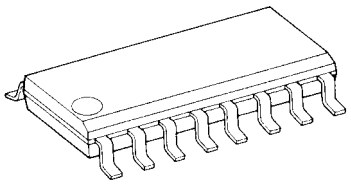
7-ch Darlington Sink Driver

ULN2003APG
ULN2004APG



DIP16-P-300-2.54A

ULN2003AFWG
ULN2004AFWG



SOL16-P-150-1.27A

Weight
DIP16−P-300-2.54A : 1.11 g (typ.)
SOL16−P-150-1.27A: 0.15 g (typ.)

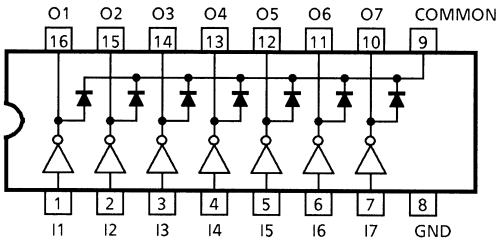
The ULN2003APG/AFWG Series are high−voltage, high−current darlington drivers comprised of seven NPN darlington pairs.
All units feature integral clamp diodes for switching inductive loads.
Applications include relay, hammer, lamp and display (LED) drivers.

Features

- Output current (single output): 500 mA max
- High sustaining voltage output: 50 V min
- Output clamp diodes
- Inputs compatible with various types of logic
- Package Type-APG: DIP-16pin
- Package Type-AFWG: SOL-16pin

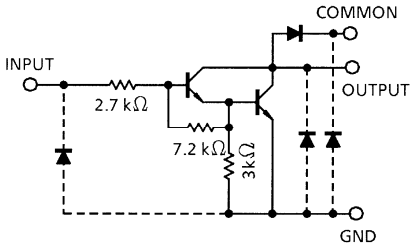
Type	Input Base Resistor	Designation
ULN2003APG/AFWG	2.7 kΩ	TTL, 5 V CMOS
ULN2004APG/AFWG	10.5 kΩ	6 to 15 V PMOS, CMOS

Pin Connection (top view)

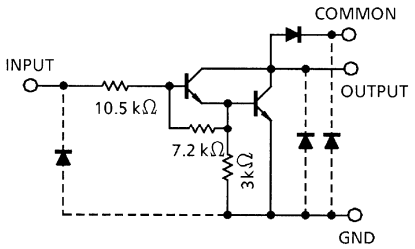


Schematics (each driver)

ULN2003APG/AFWG



ULN2004APG/AFWG



Note: The input and output parasitic diodes cannot be used as clamp diodes.

Absolute Maximum Ratings (Ta = 25°C)

Characteristic		Symbol	Rating	Unit
Output sustaining voltage		$V_{CE\ (SUS)}$	−0.5 to 50	V
Output current		I_{OUT}	500	mA/ch
Input voltage		V_{IN}	−0.5 to 30	V
Clamp diode reverse voltage		V_R	50	V
Clamp diode forward current		I_F	500	mA
Power dissipation	APG	P_D	1.47	W
	AFWG		1.25 (Note)	
Operating temperature		T_{opr}	−40 to 85	°C
Storage temperature		T_{stg}	−55 to 150	°C

Note: On PCB (Test Board: JEDEC 2s2p)

Recommended Operating Conditions (Ta = −40 to 85°C)

Characteristic		Symbol	Test Condition		Min	Typ.	Max	Unit
Output sustaining voltage		$V_{CE\ (SUS)}$	—		0	—	50	V
Output current	APG	I_{OUT}	t_{pw} = 25 ms 7 Circuits Ta = 85°C T_j = 120°C	Duty = 10%	0	—	350	mA/ch
				Duty = 50%	0	—	100	
	AFWG			Duty = 10%	0	—	300	
				Duty = 50%	0	—	90	
Input voltage		V_{IN}	—		0	—	24	V
Input voltage (output on)	ULN2003A	$V_{IN\ (ON)}$	I_{OUT} = 400 mA h_{FE} = 800		2.8	—	24	V
	ULN2004A				6.2	—	24	
Input voltage (output off)	ULN2003A	$V_{IN\ (OFF)}$	—		0	—	0.7	V
	ULN2004A		—		0	—	1.0	
Clamp diode reverse voltage		V_R	—		—	—	50	V
Clamp diode forward current		I_F	—		—	—	350	mA
Power dissipation	APG	P_D	Ta = 85°C		—	—	0.76	W
	AFWG		Ta = 85°C	(Note)	—	—	0.65	

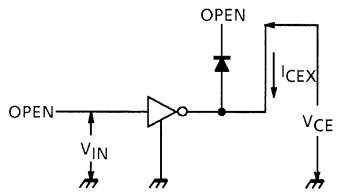
Note: On PCB (Test Board: JEDEC 2s2p)

Electrical Characteristics (Ta = 25°C unless otherwise noted)

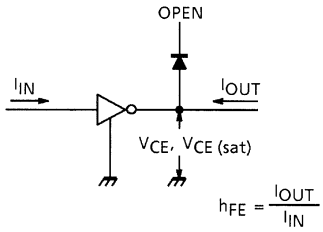
Characteristic		Symbol	Test Circuit	Test Condition		Min	Typ.	Max	Unit
Output leakage current		I_{CEX}	1	V_{CE} = 50 V, Ta = 25°C		—	—	50	μA
				V_{CE} = 50 V, Ta = 85°C		—	—	100	
Collector−emitter saturation voltage		$V_{CE\ (sat)}$	2	I_{OUT} = 350 mA, I_{IN} = 500 μA		—	1.3	1.6	V
				I_{OUT} = 200 mA, I_{IN} = 350 μA		—	1.1	1.3	
				I_{OUT} = 100 mA, I_{IN} = 250 μA		—	0.9	1.1	
DC Current transfer ratio		h_{FE}	2	V_{CE} = 2 V, I_{OUT} = 350 mA		1000	—	—	—
Input current (output on)	ULN2003A	$I_{IN\ (ON)}$	3	V_{IN} = 2.4 V, I_{OUT} = 350 mA		—	0.4	0.7	mA
	ULN2004A			V_{IN} = 9.5 V, I_{OUT} = 350 mA		—	0.8	1.2	
Input current (output off)		$I_{IN\ (OFF)}$	4	I_{OUT} = 500 μA, Ta = 85°C		50	65	—	μA
Input voltage (output on)	ULN2003A	$V_{IN\ (ON)}$	5	V_{CE} = 2 V h_{FE} = 800	I_{OUT} = 350 mA	—	—	2.6	V
					I_{OUT} = 200 mA	—	—	2.0	
	ULN2004A				I_{OUT} = 350 mA	—	—	4.7	
					I_{OUT} = 200 mA	—	—	4.4	
Clamp diode reverse current		I_R	6	V_R = 50 V, Ta = 25°C		—	—	50	μA
				V_R = 50 V, Ta = 85°C		—	—	100	
Clamp diode forward voltage		V_F	7	I_F = 350 mA		—	—	2.0	V
Input capacitance		C_{IN}	—	—		—	15	—	pF
Turn−on delay		t_{ON}	8	V_{OUT} = 50 V, R_L = 125 Ω C_L = 15 pF		—	0.1	—	μs
Turn−off delay		t_{OFF}	8	V_{OUT} = 50 V, R_L = 125 Ω C_L = 15 pF		—	0.2	—	

Test Circuit

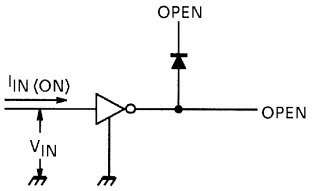
1. I_{CEX}



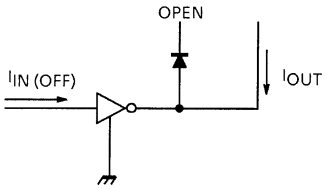
2. $V_{CE\ (sat)}$, h_{FE}



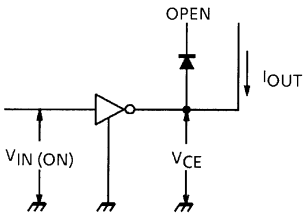
3. $I_{IN\ (ON)}$



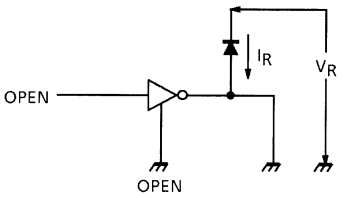
4. $I_{IN\ (OFF)}$



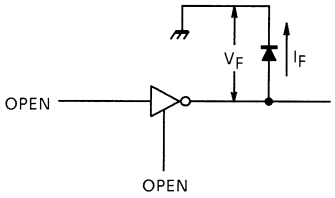
5. $V_{IN\ (ON)}$



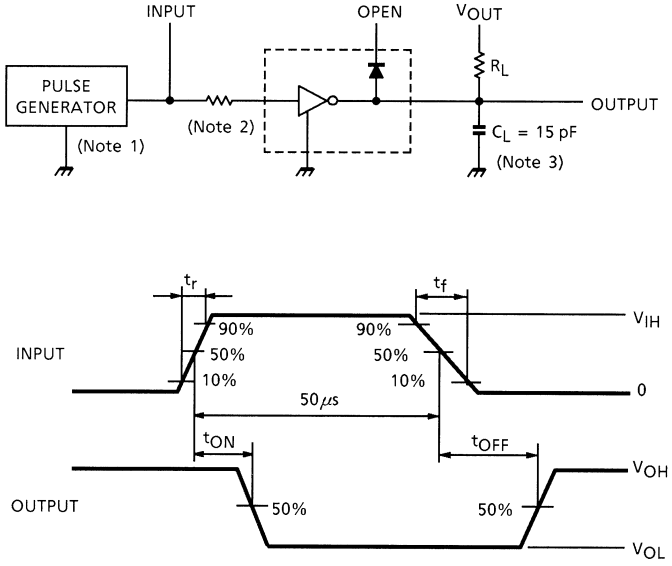
6. I_R



7. V_F



8. t_{ON}, t_{OFF}



Note 1: Pulse width 50 μs, duty cycle 10%
Output impedance 50 Ω, $t_r \leq 5$ ns, $t_f \leq 10$ ns

Note 2: See below

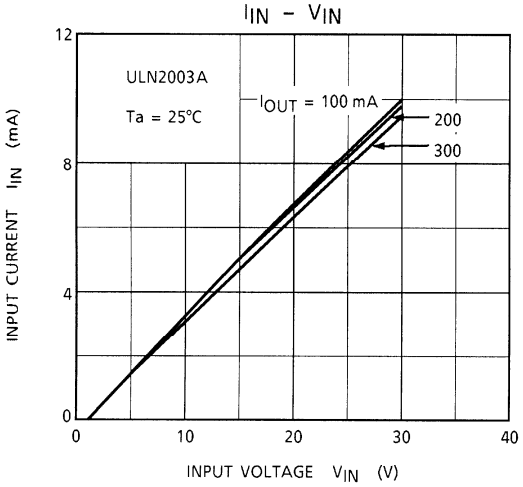
Input Condition

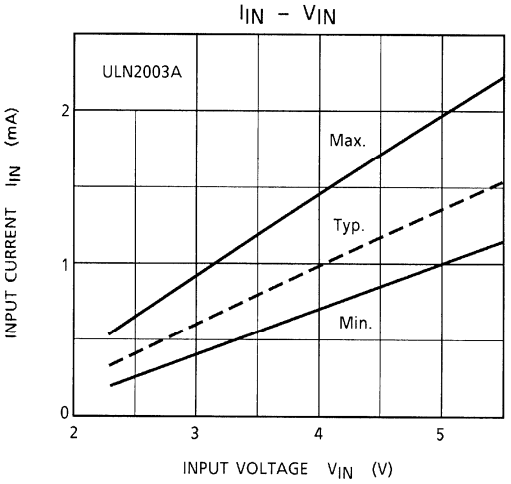
Type Number	R1	V_{IH}
ULN2003A	0	3 V
ULN2004A	0	8 V

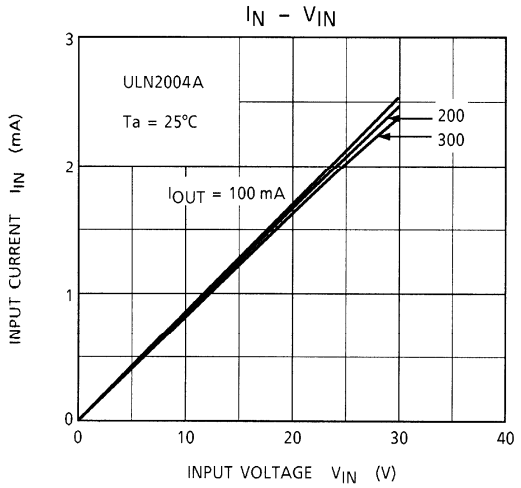
Note 3: C_L includes probe and jig capacitance.

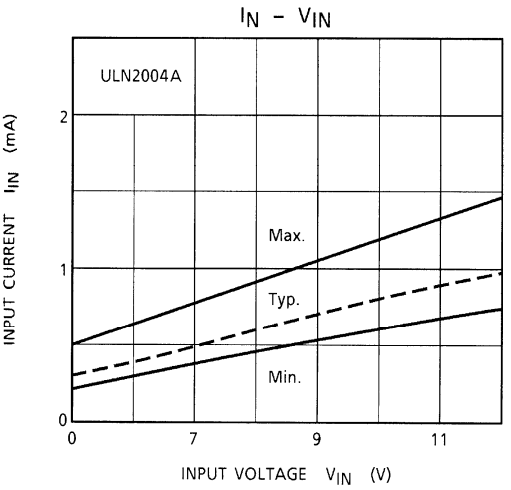
Precautions for Using

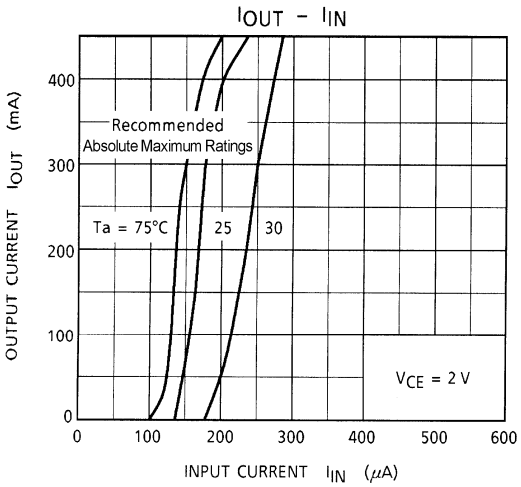
This IC does not include built-in protection circuits for excess current or overvoltage.
If this IC is subjected to excess current or overvoltage, it may be destroyed.
Hence, the utmost care must be taken when systems which incorporate this IC are designed.
Utmost care is necessary in the design of the output line, COMMON and GND line since IC may be destroyed due to short–circuit between outputs, air contamination fault, or fault by improper grounding.

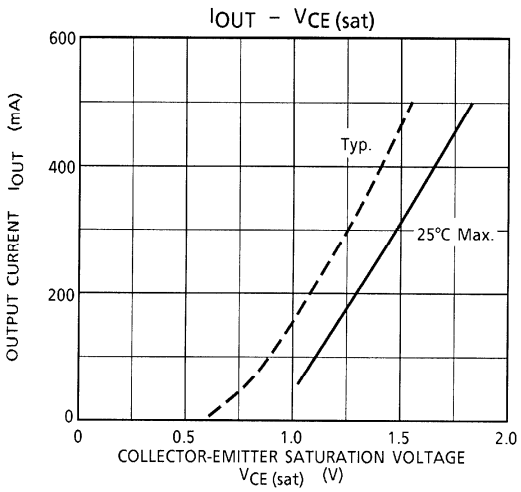

I_{IN} – V_{IN}
ULN2003A
Ta = 25°C
I_{OUT} = 100 mA
200
300
INPUT CURRENT I_{IN} (mA)
INPUT VOLTAGE V_{IN} (V)
0
4
8
12
0
10
20
30
40

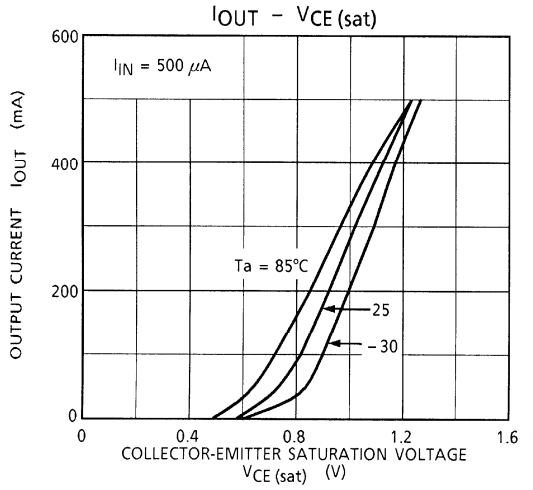

I_{IN} – V_{IN}
ULN2003A
Max.
Typ.
Min.
INPUT CURRENT I_{IN} (mA)
INPUT VOLTAGE V_{IN} (V)
0
1
2
2
3
4
5

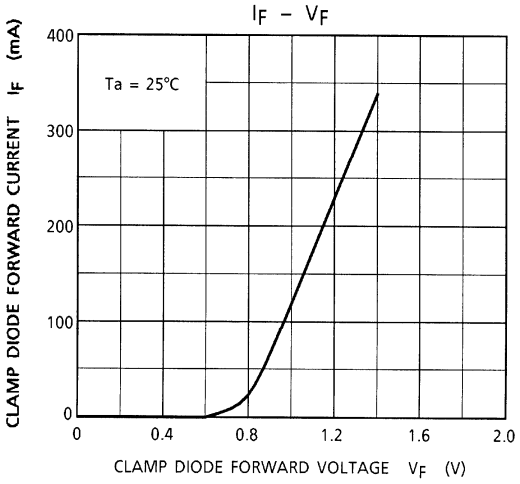

I_N – V_{IN}
ULN2004A
Ta = 25°C
200
300
I_{OUT} = 100 mA
INPUT CURRENT I_{IN} (mA)
INPUT VOLTAGE V_{IN} (V)
0
1
2
3
0
10
20
30
40

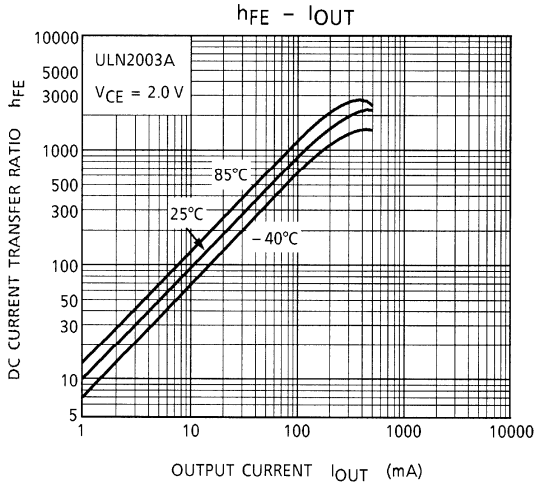

I_N – V_{IN}
ULN2004A
Max.
Typ.
Min.
INPUT CURRENT I_{IN} (mA)
INPUT VOLTAGE V_{IN} (V)
0
1
2
0
7
9
11

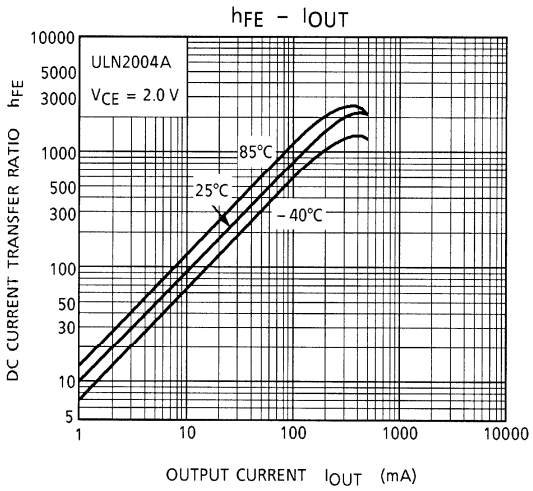

I_{OUT} – I_{IN}
Recommended
Absolute Maximum Ratings
Ta = 75°C
25
30
V_{CE} = 2 V
OUTPUT CURRENT I_{OUT} (mA)
INPUT CURRENT I_{IN} (μA)
0
100
200
300
400
0
100
200
300
400
500
600

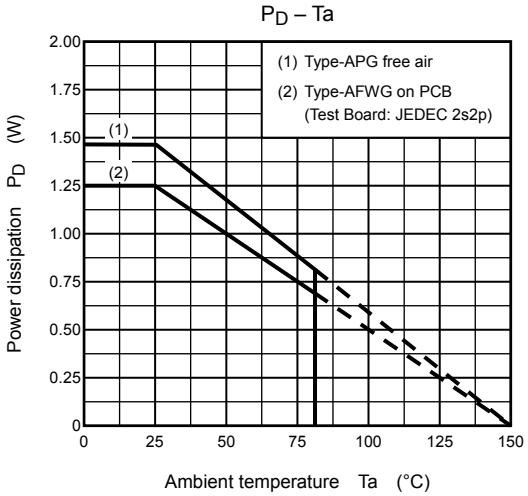

I_{OUT} – $V_{CE\,(sat)}$
Typ.
25°C Max.
OUTPUT CURRENT I_{OUT} (mA)
COLLECTOR-EMITTER SATURATION VOLTAGE
$V_{CE\,(sat)}$ (V)
0
200
400
600
0
0.5
1.0
1.5
2.0


I_{OUT} – $V_{CE\,(sat)}$
I_{IN} = 500 μA
Ta = 85°C
25
– 30
OUTPUT CURRENT I_{OUT} (mA)
COLLECTOR-EMITTER SATURATION VOLTAGE
$V_{CE\,(sat)}$ (V)


I_F – V_F
Ta = 25°C
CLAMP DIODE FORWARD CURRENT I_F (mA)
CLAMP DIODE FORWARD VOLTAGE V_F (V)

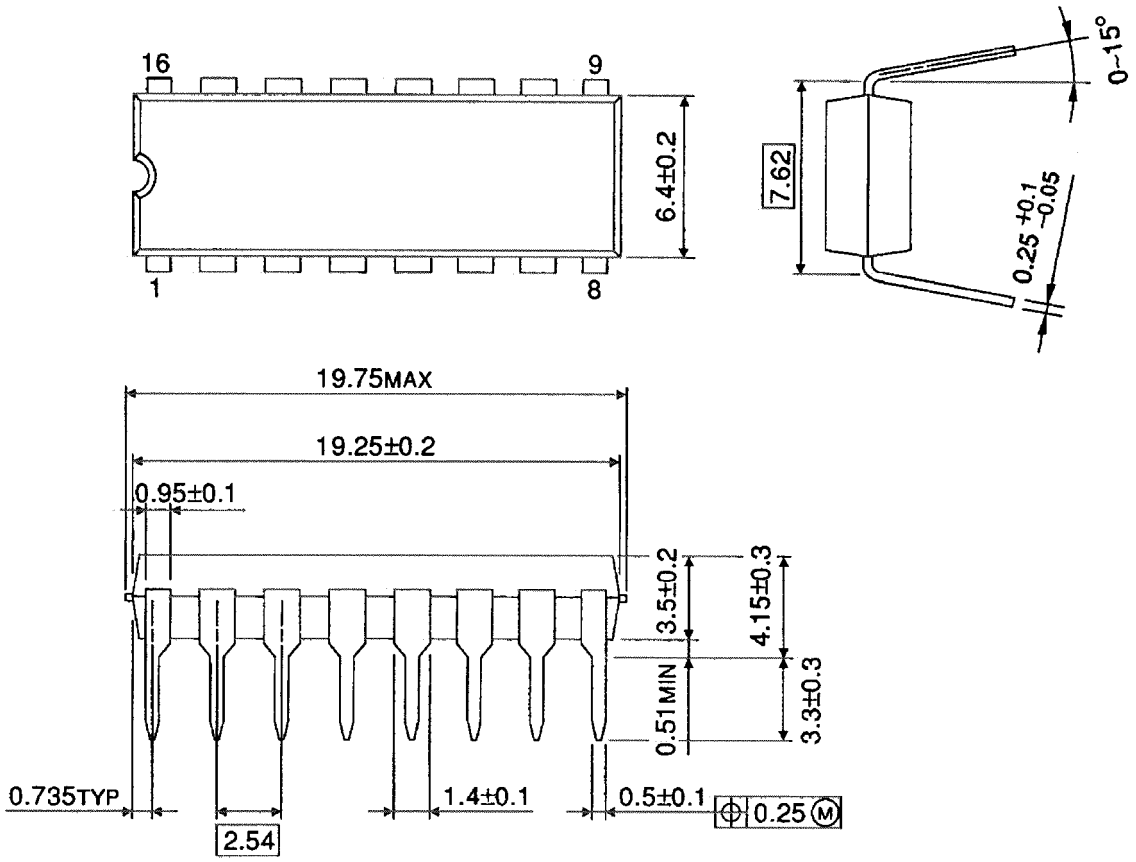

h_{FE} – I_{OUT}
ULN2003A
V_{CE} = 2.0 V
85°C
25°C
– 40°C
DC CURRENT TRANSFER RATIO h_{FE}
OUTPUT CURRENT I_{OUT} (mA)


h_{FE} – I_{OUT}
ULN2004A
V_{CE} = 2.0 V
85°C
25°C
– 40°C
DC CURRENT TRANSFER RATIO h_{FE}
OUTPUT CURRENT I_{OUT} (mA)


P_D – Ta
(1) Type-APG free air
(2) Type-AFWG on PCB
(Test Board: JEDEC 2s2p)
(1)
(2)
Power dissipation P_D (W)
Ambient temperature Ta (°C)

Package Dimensions

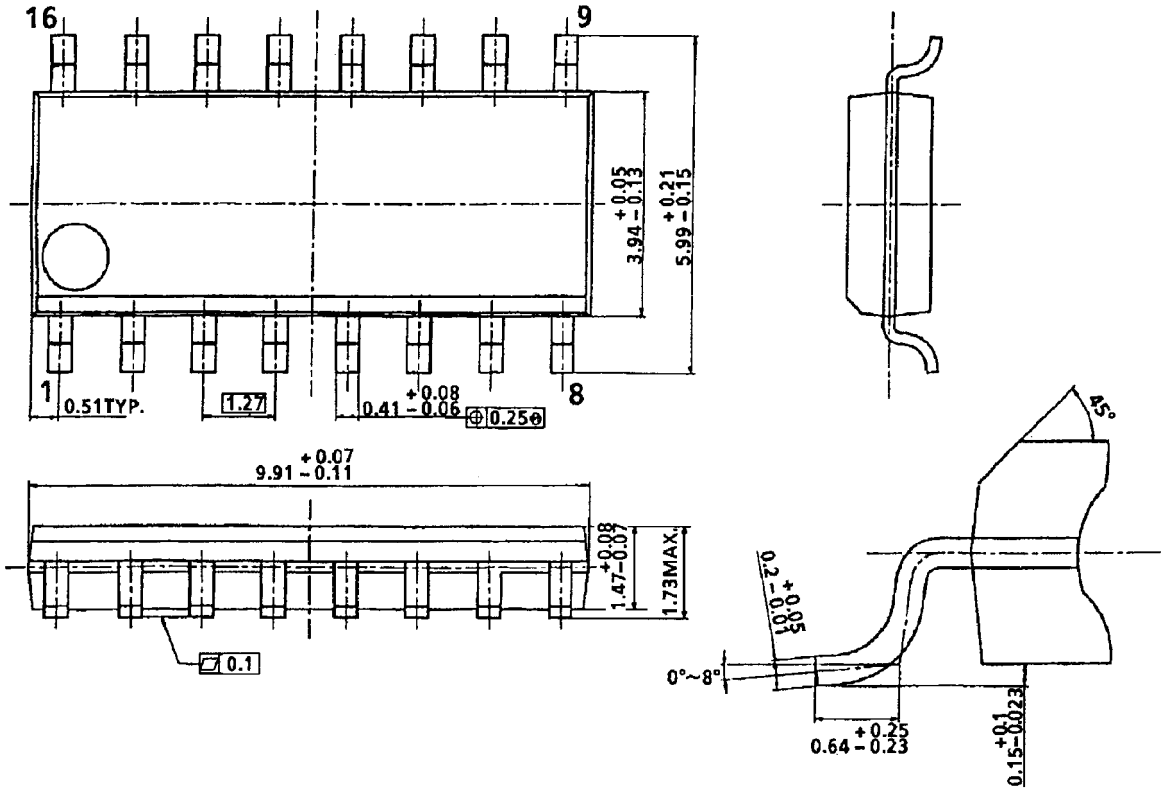
DIP16-P-300-2.54A Unit : mm



Weight: 1.11 g (typ.)

Package Dimensions

SOL16-P-150-1.27A Unit : mm



Weight: 0.15 g (typ.)

Notes on Contents

1. Equivalent Circuits

The equivalent circuit diagrams may be simplified or some parts of them may be omitted for explanatory purposes.

2. Test Circuits

Components in the test circuits are used only to obtain and confirm the device characteristics. These components and circuits are not guaranteed to prevent malfunction or failure from occurring in the application equipment.

IC Usage Considerations

Notes on Handling of ICs

(1) The absolute maximum ratings of a semiconductor device are a set of ratings that must not be exceeded, even for a moment. Do not exceed any of these ratings.
Exceeding the rating(s) may cause the device breakdown, damage or deterioration, and may result injury by explosion or combustion.

(2) Use an appropriate power supply fuse to ensure that a large current does not continuously flow in case of over current and/or IC failure. The IC will fully break down when used under conditions that exceed its absolute maximum ratings, when the wiring is routed improperly or when an abnormal pulse noise occurs from the wiring or load, causing a large current to continuously flow and the breakdown can lead smoke or ignition. To minimize the effects of the flow of a large current in case of breakdown, appropriate settings, such as fuse capacity, fusing time and insertion circuit location, are required.

(3) If your design includes an inductive load such as a motor coil, incorporate a protection circuit into the design to prevent device malfunction or breakdown caused by the current resulting from the inrush current at power ON or the negative current resulting from the back electromotive force at power OFF. IC breakdown may cause injury, smoke or ignition.
Use a stable power supply with ICs with built-in protection functions. If the power supply is unstable, the protection function may not operate, causing IC breakdown. IC breakdown may cause injury, smoke or ignition.

(4) Do not insert devices in the wrong orientation or incorrectly.
Make sure that the positive and negative terminals of power supplies are connected properly. Otherwise, the current or power consumption may exceed the absolute maximum rating, and exceeding the rating(s) may cause the device breakdown, damage or deterioration, and may result injury by explosion or combustion.
In addition, do not use any device that is applied the current with inserting in the wrong orientation or incorrectly even just one time.

(5) Carefully select external components (such as inputs and negative feedback capacitors) and load components (such as speakers), for example, power amp and regulator.
If there is a large amount of leakage current such as input or negative feedback condenser, the IC output DC voltage will increase. If this output voltage is connected to a speaker with low input withstand voltage, overcurrent or IC failure can cause smoke or ignition. (The over current can cause smoke or ignition from the IC itself.) In particular, please pay attention when using a Bridge Tied Load (BTL) connection type IC that inputs output DC voltage to a speaker directly.

Points to Remember on Handling of ICs

(1) Heat Radiation Design
In using an IC with large current flow such as power amp, regulator or driver, please design the device so that heat is appropriately radiated, not to exceed the specified junction temperature (Tj) at any time and condition. These ICs generate heat even during normal use. An inadequate IC heat radiation design can lead to decrease in IC life, deterioration of IC characteristics or IC breakdown. In addition, please design the device taking into considerate the effect of IC heat radiation with peripheral components.

(2) Back-EMF
When a motor rotates in the reverse direction, stops or slows down abruptly, a current flow back to the motor's power supply due to the effect of back-EMF. If the current sink capability of the power supply is small, the device's motor power supply and output pins might be exposed to conditions beyond absolute maximum ratings. To avoid this problem, take the effect of back-EMF into consideration in system design.

About solderability, following conditions were confirmed

- Solderability

(1) Use of Sn-37Pb solder Bath
- solder bath temperature = 230°C
- dipping time = 5 seconds
- the number of times = once
- use of R-type flux

(2) Use of Sn-3.0Ag-0.5Cu solder Bath
- solder bath temperature = 245°C
- dipping time = 5 seconds
- the number of times = once
- use of R-type flux

RESTRICTIONS ON PRODUCT USE

- Toshiba Corporation, and its subsidiaries and affiliates (collectively “TOSHIBA”), reserve the right to make changes to the information in this document, and related hardware, software and systems (collectively “Product”) without notice.
- This document and any information herein may not be reproduced without prior written permission from TOSHIBA. Even with TOSHIBA’s written permission, reproduction is permissible only if reproduction is without alteration/omission.
- Though TOSHIBA works continually to improve Product’s quality and reliability, Product can malfunction or fail. Customers are responsible for complying with safety standards and for providing adequate designs and safeguards for their hardware, software and systems which minimize risk and avoid situations in which a malfunction or failure of Product could cause loss of human life, bodily injury or damage to property, including data loss or corruption. Before customers use the Product, create designs including the Product, or incorporate the Product into their own applications, customers must also refer to and comply with (a) the latest versions of all relevant TOSHIBA information, including without limitation, this document, the specifications, the data sheets and application notes for Product and the precautions and conditions set forth in the “TOSHIBA Semiconductor Reliability Handbook” and (b) the instructions for the application with which the Product will be used with or for. Customers are solely responsible for all aspects of their own product design or applications, including but not limited to (a) determining the appropriateness of the use of this Product in such design or applications; (b) evaluating and determining the applicability of any information contained in this document, or in charts, diagrams, programs, algorithms, sample application circuits, or any other referenced documents; and (c) validating all operating parameters for such designs and applications. **TOSHIBA ASSUMES NO LIABILITY FOR CUSTOMERS’ PRODUCT DESIGN OR APPLICATIONS.**
- Product is intended for use in general electronics applications (e.g., computers, personal equipment, office equipment, measuring equipment, industrial robots and home electronics appliances) or for specific applications as expressly stated in this document. Product is neither intended nor warranted for use in equipment or systems that require extraordinarily high levels of quality and/or reliability and/or a malfunction or failure of which may cause loss of human life, bodily injury, serious property damage or serious public impact (“Unintended Use”). Unintended Use includes, without limitation, equipment used in nuclear facilities, equipment used in the aerospace industry, medical equipment, equipment used for automobiles, trains, ships and other transportation, traffic signaling equipment, equipment used to control combustions or explosions, safety devices, elevators and escalators, devices related to electric power, and equipment used in finance-related fields. Do not use Product for Unintended Use unless specifically permitted in this document.
- Do not disassemble, analyze, reverse-engineer, alter, modify, translate or copy Product, whether in whole or in part.
- Product shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable laws or regulations.
- The information contained herein is presented only as guidance for Product use. No responsibility is assumed by TOSHIBA for any infringement of patents or any other intellectual property rights of third parties that may result from the use of Product. No license to any intellectual property right is granted by this document, whether express or implied, by estoppel or otherwise.
- **ABSENT A WRITTEN SIGNED AGREEMENT, EXCEPT AS PROVIDED IN THE RELEVANT TERMS AND CONDITIONS OF SALE FOR PRODUCT, AND TO THE MAXIMUM EXTENT ALLOWABLE BY LAW, TOSHIBA (1) ASSUMES NO LIABILITY WHATSOEVER, INCLUDING WITHOUT LIMITATION, INDIRECT, CONSEQUENTIAL, SPECIAL, OR INCIDENTAL DAMAGES OR LOSS, INCLUDING WITHOUT LIMITATION, LOSS OF PROFITS, LOSS OF OPPORTUNITIES, BUSINESS INTERRUPTION AND LOSS OF DATA, AND (2) DISCLAIMS ANY AND ALL EXPRESS OR IMPLIED WARRANTIES AND CONDITIONS RELATED TO SALE, USE OF PRODUCT, OR INFORMATION, INCLUDING WARRANTIES OR CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, ACCURACY OF INFORMATION, OR NONINFRINGEMENT.**
- Do not use or otherwise make available Product or related software or technology for any military purposes, including without limitation, for the design, development, use, stockpiling or manufacturing of nuclear, chemical, or biological weapons or missile technology products (mass destruction weapons). Product and related software and technology may be controlled under the Japanese Foreign Exchange and Foreign Trade Law and the U.S. Export Administration Regulations. Export and re-export of Product or related software or technology are strictly prohibited except in compliance with all applicable export laws and regulations.
- Please contact your TOSHIBA sales representative for details as to environmental matters such as the RoHS compatibility of Product. Please use Product in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. TOSHIBA assumes no liability for damages or losses occurring as a result of noncompliance with applicable laws and regulations.

Mouser Electronics

Authorized Distributor

Click to View Pricing, Inventory, Delivery & Lifecycle Information:

Toshiba:

ULN2003APG(5,M)

MP1584

3A, 1.5MHz, 28V Step-Down Converter

The Future of Analog IC Technology

DESCRIPTION

The MP1584 is a high frequency step-down switching regulator with an integrated internal high-side high voltage power MOSFET. It provides 3A output with current mode control for fast loop response and easy compensation.

The wide 4.5V to 28V input range accommodates a variety of step-down applications, including those in an automotive input environment. A 100μA operational quiescent current allows use in battery-powered applications.

High power conversion efficiency over a wide load range is achieved by scaling down the switching frequency at light load condition to reduce the switching and gate driving losses.

The frequency foldback helps prevent inductor current runaway during startup and thermal shutdown provides reliable, fault tolerant operation.

By switching at 1.5MHz, the MP1584 is able to prevent EMI (Electromagnetic Interference) noise problems, such as those found in AM radio and ADSL applications.

The MP1584 is available in a thermally enhanced SOIC8E package.

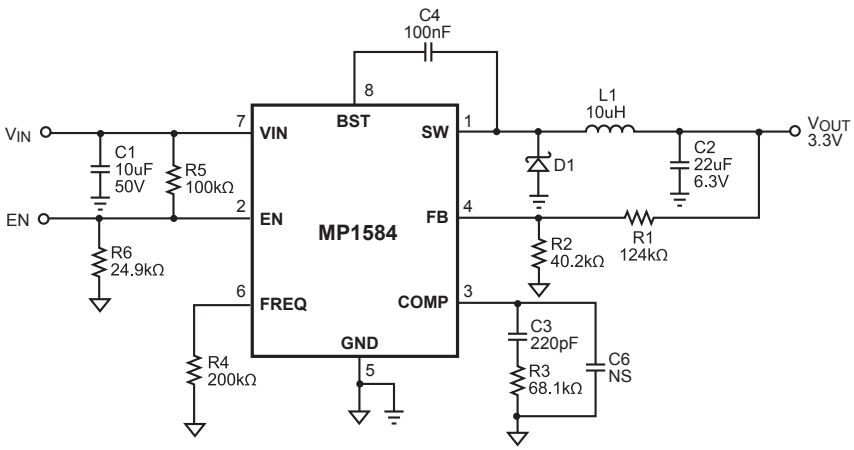
FEATURES

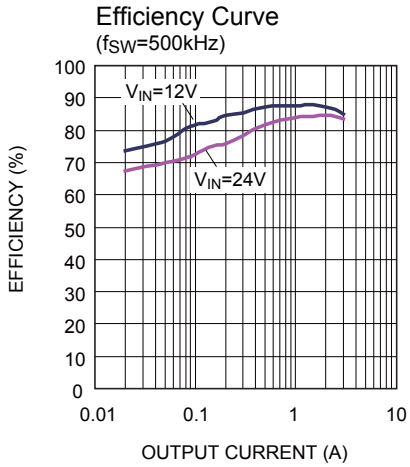
- Wide 4.5V to 28V Operating Input Range
- Programmable Switching Frequency from 100kHz to 1.5MHz
- High-Efficiency Pulse Skipping Mode for Light Load
- Ceramic Capacitor Stable
- Internal Soft-Start
- Internally Set Current Limit without a Current Sensing Resistor
- Available in SOIC8E Package.

APPLICATIONS

- High Voltage Power Conversion
- Automotive Systems
- Industrial Power Systems
- Distributed Power Systems
- Battery Powered Systems

TYPICAL APPLICATION



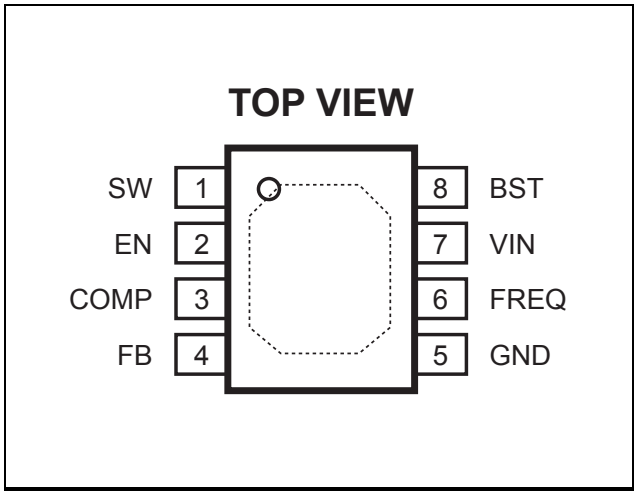


ORDERING INFORMATION

Part Number*	Package	Top Marking	Free Air Temperature (T_A)
MP1584EN	SOIC8E	MP1584EN	–20°C to +85°C

* For Tape & Reel, add suffix –Z (e.g. MP1584EN–Z);
For RoHS Compliant Packaging, add suffix –LF. (e.g. MP1584EN–LF–Z)

PACKAGE REFERENCE



ABSOLUTE MAXIMUM RATINGS (1)

Supply Voltage (V_{IN}).....–0.3V to +30V
Switch Voltage (V_{SW})..... –0.3V to V_{IN} + 0.3V
BST to SW .....–0.3V to +6V
All Other Pins.....–0.3V to +6V
Continuous Power Dissipation (T_A = +25°C)(2)
..... 2.5W
Junction Temperature.....150°C
Lead Temperature .....260°C
Storage Temperature..... –65°C to +150°C

Recommended Operating Conditions (3)

Supply Voltage V_{IN} .....4.5V to 28V
Output Voltage V_{OUT}.....0.8V to 25V
Operating Junct. Temp (T_J) .....–20°C to +125°C

Thermal Resistance (4)

	θ_{JA}	θ_{JC}
SOIC8E	50	10... °C/W

Notes:
1) Exceeding these ratings may damage the device.
2) The maximum allowable power dissipation is a function of the maximum junction temperature T_J(MAX), the junction-to-ambient thermal resistance θ_{JA}, and the ambient temperature T_A. The maximum allowable continuous power dissipation at any ambient temperature is calculated by P_D(MAX)=(T_J(MAX)-T_A)/ θ_{JA}. Exceeding the maximum allowable power dissipation will cause excessive die temperature, and the regulator will go into thermal shutdown. Internal thermal shutdown circuitry protects the device from permanent damage.
3) The device is not guaranteed to function outside of its operating conditions.
4) Measured on JESD51-7, 4-layer PCB.

ELECTRICAL CHARACTERISTICS

V_{IN} = 12V, V_{EN} = 2.5V, V_{COMP} = 1.4V, T_A= +25°C, unless otherwise noted.

Parameter	Symbol	Condition	Min	Typ	Max	Units
Feedback Voltage	V_{FB}	4.5V < V_{IN} < 28V	0.776	0.8	0.824	V
Upper Switch On Resistance	$R_{DS(ON)}$	V_{BST} – V_{SW} = 5V		150		mΩ
Upper Switch Leakage		V_{EN} = 0V, V_{SW} = 0V, V_{IN} = 28V		1		μA
Current Limit			4.0	4.7		A
COMP to Current Sense Transconductance	G_{CS}			9		A/V
Error Amp Voltage Gain (5)				200		V/V
Error Amp Transconductance		I_{COMP} = ±3μA	40	60	80	μA/V
Error Amp Min Source current		V_{FB} = 0.7V		5		μA
Error Amp Min Sink current		V_{FB} = 0.9V		–5		μA
VIN UVLO Threshold			2.7	3.0	3.3	V
VIN UVLO Hysteresis				0.35		V
Soft-Start Time (5)		0V < V_{FB} < 0.8V		1.5		ms
Oscillator Frequency		R_{FREQ} = 100kΩ		900		kHz
Shutdown Supply Current		V_{EN} = 0V		12	20	μA
Quiescent Supply Current		No load, V_{FB} = 0.9V		100	125	μA
Thermal Shutdown				150		°C
Thermal Shutdown Hysteresis				15		°C
Minimum Off Time (5)				100		ns
Minimum On Time (5)				100		ns
EN Up Threshold			1.35	1.5	1.65	V
EN Hysteresis				300		mV

Note:

5) Guaranteed by design.

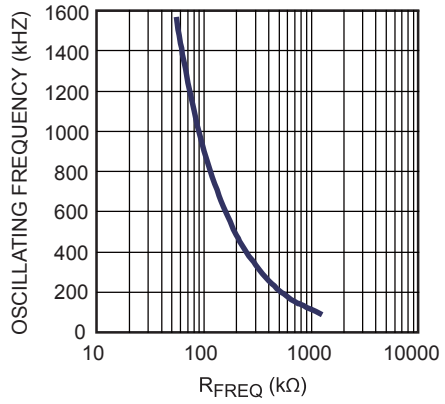
PIN FUNCTIONS

SOIC Pin #	Name	Description
1	SW	Switch Node. This is the output from the high-side switch. A low forward drop Schottky diode to ground is required. The diode must be close to the SW pins to reduce switching spikes.
2	EN	Enable Input. Pulling this pin below the specified threshold shuts the chip down. Pulling it up above the specified threshold or leaving it floating enables the chip.
3	COMP	Compensation. This node is the output of the error amplifier. Control loop frequency compensation is applied to this pin.
4	FB	Feedback. This is the input to the error amplifier. The output voltage is set by a resistive divider connected between the output and GND which scales down V_{OUT} equal to the internal +0.8V reference.
5	GND Exposed Pad	Ground. It should be connected as close as possible to the output capacitor to shorten the high current switch paths. Connect exposed pad to GND plane for optimal thermal performance.
6	FREQ	Switching Frequency Program Input. Connect a resistor from this pin to ground to set the switching frequency.
7	VIN	Input Supply. This supplies power to all the internal control circuitry, both BS regulators and the high-side switch. A decoupling capacitor to ground must be placed close to this pin to minimize switching spikes.
8	BST	Bootstrap. This is the positive power supply for the internal floating high-side MOSFET driver. Connect a bypass capacitor between this pin and SW pin.

TYPICAL PERFORMANCE CHARACTERISTICS

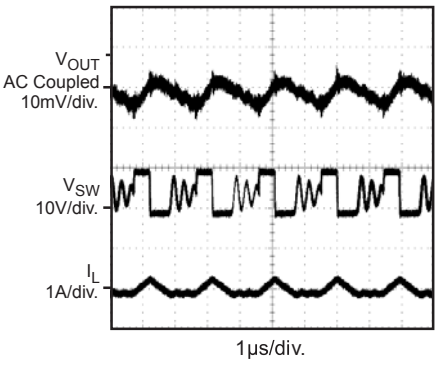
V_{IN} = 12V, V_{OUT}=5V, C1 = 10µF, C2 = 22µF, L1= 10µH, T_A = +25°C, unless otherwise noted.

Oscillating Frequency vs. R_{freq}



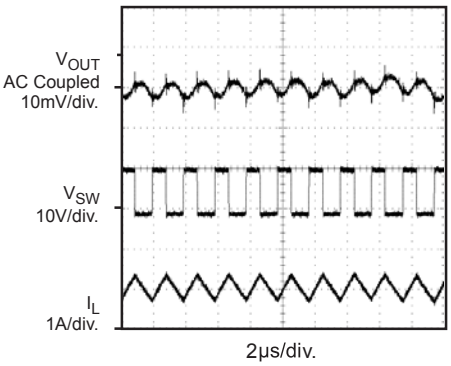
Steady State

I_{OUT}=0.1A, f_{SW}=500kHz



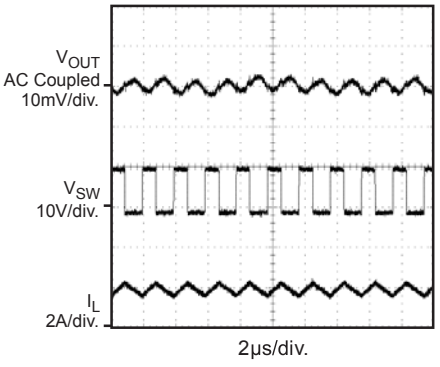
Steady State

I_{OUT}=1A, f_{SW}=500kHz



Steady State

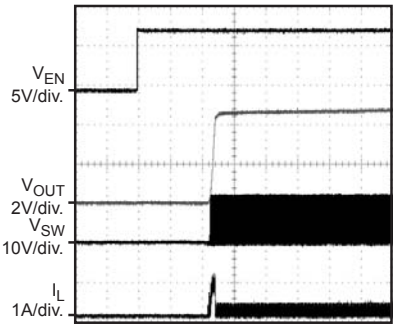
I_{OUT}=2A, f_{SW}=500kHz



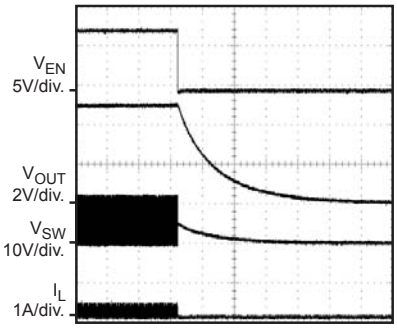
TYPICAL PERFORMANCE CHARACTERISTICS *(continued)*

V_{IN} = 12V, C1 = 10µF, C2 = 22µF, L1 = 10µH, f_{SW}=500kHz, and T_A = +25°C, unless otherwise noted.

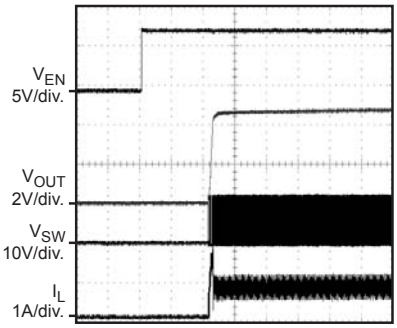
Startup
I_{OUT} = 0.1A



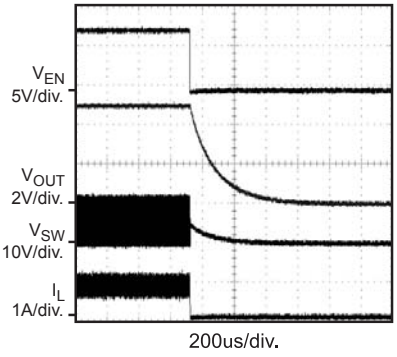
Shutdown
I_{OUT} = 0.1A



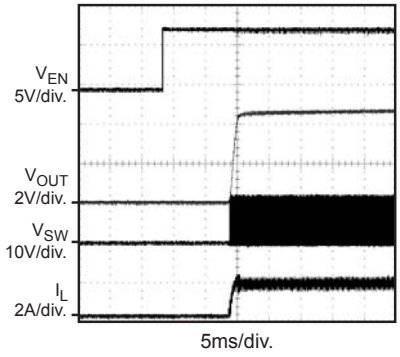
Startup
I_{OUT} = 1A



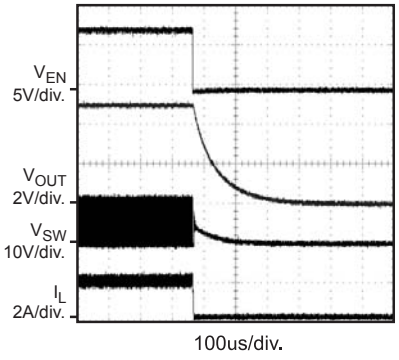
Shutdown
I_{OUT} = 1A



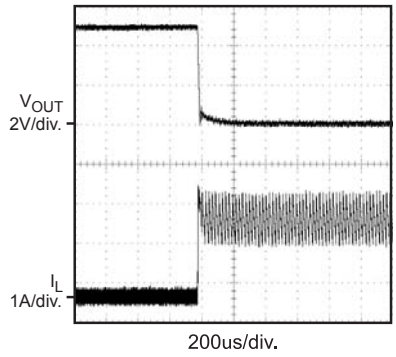
Startup
I_{OUT} = 2A



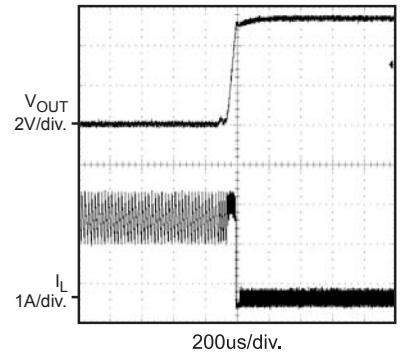
Shutdown
I_{OUT} = 2A



Short Circuit Entry
I_{OUT} = 0.1A to Short



Short Circuit Recovery
I_{OUT} = Short to 0.1A



BLOCK DIAGRAM

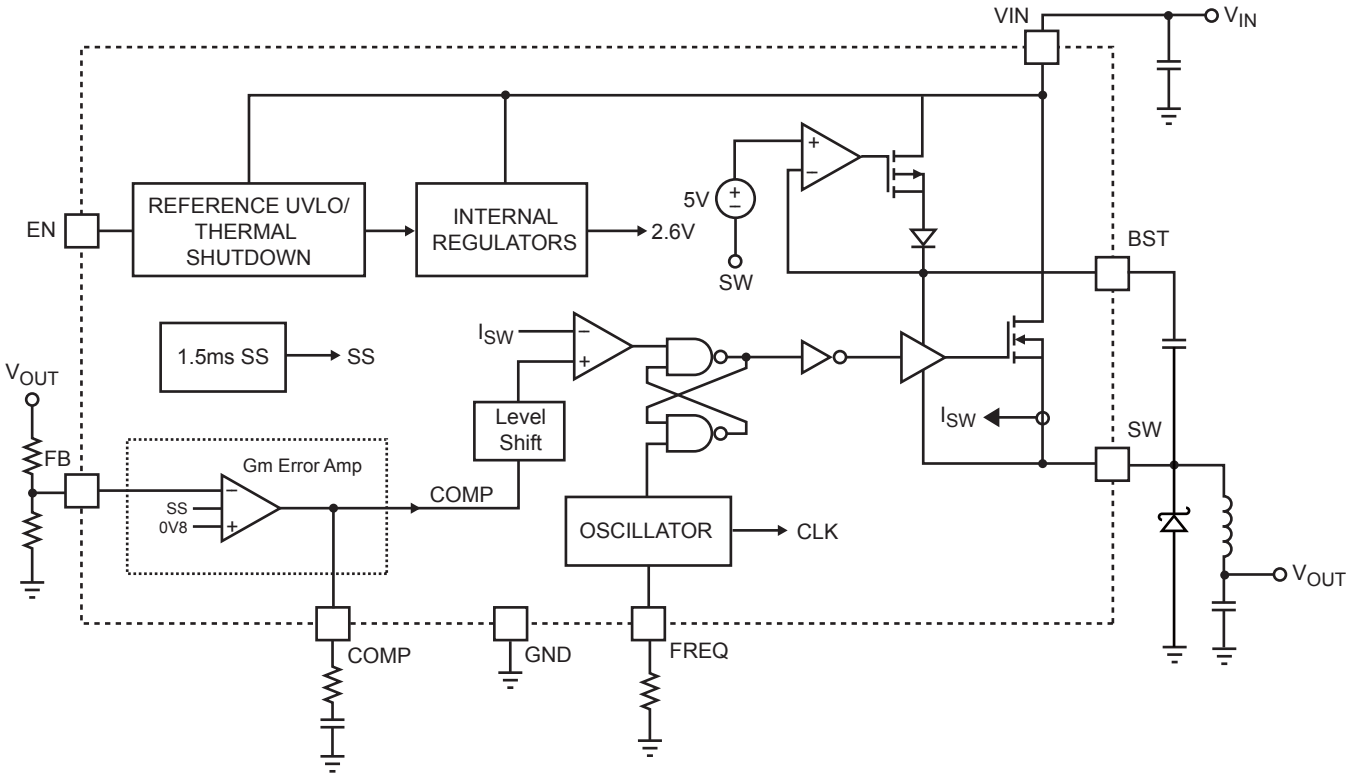


Figure 1—Functional Block Diagram

OPERATION

The MP1584 is a variable frequency, non-synchronous, step-down switching regulator with an integrated high-side high voltage power MOSFET. It provides a highly efficient solution with current mode control for fast loop response and easy compensation. It features a wide input voltage range, internal soft-start control and precision current limiting. Its very low operational quiescent current makes it suitable for battery powered applications.

PWM Control

At moderate to high output current, the MP1584 operates in a fixed frequency, peak current control mode to regulate the output voltage. A PWM cycle is initiated by the internal clock. The power MOSFET is turned on and remains on until its current reaches the value set by the COMP voltage. When the power switch is off, it remains off for at least 100ns before the next cycle starts. If, in one PWM period, the current in the power MOSFET does not reach the COMP set current value, the power MOSFET remains on, saving a turn-off operation.

Error Amplifier

The error amplifier compares the FB pin voltage with the internal reference (REF) and outputs a current proportional to the difference between the two. This output current is then used to charge the external compensation network to form the COMP voltage, which is used to control the power MOSFET current.

During operation, the minimum COMP voltage is clamped to 0.9V and its maximum is clamped to 2.0V. COMP is internally pulled down to GND in shutdown mode. COMP should not be pulled up beyond 2.6V.

Internal Regulator

Most of the internal circuitries are powered from the 2.6V internal regulator. This regulator takes the VIN input and operates in the full VIN range. When VIN is greater than 3.0V, the output of the regulator is in full regulation. When VIN is lower than 3.0V, the output decreases.

Enable Control

The MP1584 has a dedicated enable control pin (EN). With high enough input voltage, the chip can be enabled and disabled by EN which has positive logic. Its falling threshold is a precision 1.2V, and its rising threshold is 1.5V (300mV higher).

When floating, EN is pulled up to about 3.0V by an internal 1μA current source so it is enabled. To pull it down, 1μA current capability is needed.

When EN is pulled down below 1.2V, the chip is put into the lowest shutdown current mode. When EN is higher than zero but lower than its rising threshold, the chip is still in shutdown mode but the shutdown current increases slightly.

Under-Voltage Lockout (UVLO)

Under-voltage lockout (UVLO) is implemented to protect the chip from operating at insufficient supply voltage. The UVLO rising threshold is about 3.0V while its falling threshold is a consistent 2.6V.

Internal Soft-Start

The soft-start is implemented to prevent the converter output voltage from overshooting during startup. When the chip starts, the internal circuitry generates a soft-start voltage (SS) ramping up from 0V to 2.6V. When it is lower than the internal reference (REF), SS overrides REF so the error amplifier uses SS as the reference. When SS is higher than REF, REF regains control.

Thermal Shutdown

Thermal shutdown is implemented to prevent the chip from operating at exceedingly high temperatures. When the silicon die temperature is higher than its upper threshold, it shuts down the whole chip. When the temperature is lower than its lower threshold, the chip is enabled again.

Floating Driver and Bootstrap Charging

The floating power MOSFET driver is powered by an external bootstrap capacitor. This floating driver has its own UVLO protection. This UVLO's rising threshold is 2.2V with a threshold of 150mV.

The bootstrap capacitor is charged and regulated to about 5V by the dedicated internal bootstrap regulator. When the voltage between the BST and SW nodes is lower than its regulation, a PMOS pass transistor connected from VIN to BST is turned on. The charging current path is from VIN, BST and then to SW. External circuit should provide enough voltage headroom to facilitate the charging.

As long as VIN is sufficiently higher than SW, the bootstrap capacitor can be charged. When the power MOSFET is ON, VIN is about equal to SW so the bootstrap capacitor cannot be charged. When the external diode is on, the difference between VIN and SW is largest, thus making it the best period to charge. When there is no current in the inductor, SW equals the output voltage V_{OUT} so the difference between V_{IN} and V_{OUT} can be used to charge the bootstrap capacitor.

At higher duty cycle operation condition, the time period available to the bootstrap charging is less so the bootstrap capacitor may not be sufficiently charged.

In case the internal circuit does not have sufficient voltage and the bootstrap capacitor is not charged, extra external circuitry can be used to ensure the bootstrap voltage is in the normal operational region. Refer to *External Bootstrap Diode* in Application section.

The DC quiescent current of the floating driver is about 20μA. Make sure the bleeding current at the SW node is higher than this value, such that:

$$I_O + \frac{V_O}{(R1 + R2)} > 20\mu A$$

Current Comparator and Current Limit

The power MOSFET current is accurately sensed via a current sense MOSFET. It is then fed to the high speed current comparator for the current mode control purpose. The current comparator takes this sensed current as one of its inputs. When the power MOSFET is turned on, the comparator is first blanked till the end of the turn-on transition to avoid noise issues. The comparator then compares the power switch current with the COMP voltage. When the sensed current is higher than the COMP voltage, the comparator output is low, turning off the power MOSFET. The cycle-by-cycle maximum current of the internal power MOSFET is internally limited.

Startup and Shutdown

If both VIN and EN are higher than their appropriate thresholds, the chip starts. The reference block starts first, generating stable reference voltage and currents, and then the internal regulator is enabled. The regulator provides stable supply for the remaining circuitries.

While the internal supply rail is up, an internal timer holds the power MOSFET OFF for about 50μs to blank the startup glitches. When the internal soft-start block is enabled, it first holds its SS output low to ensure the remaining circuitries are ready and then slowly ramps up.

Three events can shut down the chip: EN low, VIN low and thermal shutdown. In the shutdown procedure, power MOSFET is turned off first to avoid any fault triggering. The COMP voltage and the internal supply rail are then pulled down.

Programmable Oscillator

The MP1584 oscillating frequency is set by an external resistor, R_{freq} from the FREQ pin to ground. The value of R_{freq} can be calculated from:

$$R_{freq}(k\Omega) = \frac{180000}{\left[f_s(kHz)\right]^{1.1}}$$

APPLICATION INFORMATION

COMPONENT SELECTION

Setting the Output Voltage

The output voltage is set using a resistive voltage divider from the output voltage to FB pin. The voltage divider divides the output voltage down to the feedback voltage by the ratio:

$$V_{FB} = V_{OUT} \frac{R2}{R1 + R2}$$

Thus the output voltage is:

$$V_{OUT} = V_{FB} \frac{(R1 + R2)}{R2}$$

About 20μA current from high side BS circuitry can be seen at the output when the MP1584 is at no load. In order to absorb this small amount of current, keep R2 under 40KΩ. A typical value for R2 can be 40.2kΩ. With this value, R1 can be determined by:

$$R1 = 50.25 \times (V_{OUT} - 0.8)(k\Omega)$$

For example, for a 3.3V output voltage, R2 is 40.2kΩ, and R1 is 127kΩ.

Inductor

The inductor is required to supply constant current to the output load while being driven by the switched input voltage. A larger value inductor will result in less ripple current that will result in lower output ripple voltage. However, the larger value inductor will have a larger physical size, higher series resistance, and/or lower saturation current.

A good rule for determining the inductance to use is to allow the peak-to-peak ripple current in the inductor to be approximately 30% of the maximum switch current limit. Also, make sure that the peak inductor current is below the maximum switch current limit. The inductance value can be calculated by:

$$L1 = \frac{V_{OUT}}{f_S \times \Delta I_L} \times \left(1 - \frac{V_{OUT}}{V_{IN}}\right)$$

Where V_{OUT} is the output voltage, V_{IN} is the input voltage, f_S is the switching frequency, and ΔI_L is the peak-to-peak inductor ripple current.

Choose an inductor that will not saturate under the maximum inductor peak current. The peak inductor current can be calculated by:

$$I_{LP} = I_{LOAD} + \frac{V_{OUT}}{2 \times f_S \times L1} \times \left(1 - \frac{V_{OUT}}{V_{IN}}\right)$$

Where I_{LOAD} is the load current.

Table 1 lists a number of suitable inductors from various manufacturers. The choice of which style inductor to use mainly depends on the price vs. size requirements and any EMI requirement.

Table 1—Inductor Selection Guide

Part Number	Inductance (µH)	Max DCR (Ω)	Current Rating (A)	Dimensions L x W x H (mm^3)
Wurth Electronics				
7447789003	3.3	0.024	3.42	7.3x7.3x3.2
744066100	10	0.035	3.6	10x10x3.8
744771115	15	0.025	3.75	12x12x6
744771122	22	0.031	3.37	12x12x6
TDK				
RLF7030T-3R3	3.3	0.02	4.1	7.3x6.8x3.2
RLF7030T-4R7	4.7	0.031	3.4	7.3x6.8x3.2
SLF10145T-100	10	0.0364	3	10.1x10.1x4.5
SLF12565T-220M3R5	22	0.0316	3.5	12.5x12.5x6.5
Toko				
FDV0630-3R3M	3.3	0.031	4.3	7.7x7x3
FDV0630-4R7M	4.7	0.049	3.3	7.7x7x3
919AS-100M	10	0.0265	4.3	10.3x10.3x4.5
919AS-160M	16	0.0492	3.3	10.3x10.3x4.5
919AS-220M	22	0.0776	3	10.3x10.3x4.5

Output Rectifier Diode

The output rectifier diode supplies the current to the inductor when the high-side switch is off. To reduce losses due to the diode forward voltage and recovery times, use a Schottky diode.

Choose a diode whose maximum reverse voltage rating is greater than the maximum input voltage, and whose current rating is greater than the maximum load current. Table 2 lists example Schottky diodes and manufacturers.

Table 2—Diode Selection Guide

Diodes	Voltage/ Current Rating	Manufacturer
B340A-13-F	40V, 3A	Diodes Inc.
CMSH3-40MA	40V, 3A	Central Semi

Input Capacitor

The input current to the step-down converter is discontinuous, therefore a capacitor is required to supply the AC current to the step-down converter while maintaining the DC input voltage. Use low ESR capacitors for the best performance. Ceramic capacitors are preferred, but tantalum or low-ESR electrolytic capacitors may also suffice.

For simplification, choose the input capacitor with RMS current rating greater than half of the maximum load current.

The input capacitor (C1) can be electrolytic, tantalum or ceramic. When using electrolytic or tantalum capacitors, a small, high quality ceramic capacitor, i.e. 0.1μF, should be placed as close to the IC as possible. When using ceramic capacitors, make sure that they have enough capacitance to provide sufficient charge to prevent excessive voltage ripple at input. The input voltage ripple caused by capacitance can be estimated by:

$$\Delta V_{IN} = \frac{I_{LOAD}}{f_S \times C1} \times \frac{V_{OUT}}{V_{IN}} \times \left(1 - \frac{V_{OUT}}{V_{IN}}\right)$$

Output Capacitor

The output capacitor (C2) is required to maintain the DC output voltage. Ceramic, tantalum, or low ESR electrolytic capacitors are recommended. Low ESR capacitors are preferred to keep the output voltage ripple low. The output voltage ripple can be estimated by:

$$\Delta V_{OUT} = \frac{V_{OUT}}{f_S \times L} \times \left(1 - \frac{V_{OUT}}{V_{IN}}\right) \times \left(R_{ESR} + \frac{1}{8 \times f_S \times C2}\right)$$

Where L is the inductor value and R_{ESR} is the equivalent series resistance (ESR) value of the output capacitor.

In the case of ceramic capacitors, the impedance at the switching frequency is dominated by the capacitance. The output voltage ripple is mainly caused by the capacitance. For simplification, the output voltage ripple can be estimated by:

$$\Delta V_{OUT} = \frac{V_{OUT}}{8 \times f_S^2 \times L \times C2} \times \left(1 - \frac{V_{OUT}}{V_{IN}}\right)$$

In the case of tantalum or electrolytic capacitors, the ESR dominates the impedance at the switching frequency. For simplification, the output ripple can be approximated to:

$$\Delta V_{OUT} = \frac{V_{OUT}}{f_S \times L} \times \left(1 - \frac{V_{OUT}}{V_{IN}}\right) \times R_{ESR}$$

The characteristics of the output capacitor also affect the stability of the regulation system. The MP1584 can be optimized for a wide range of capacitance and ESR values.

Compensation Components

MP1584 employs current mode control for easy compensation and fast transient response. The system stability and transient response are controlled through the COMP pin. COMP pin is the output of the internal error amplifier. A series capacitor-resistor combination sets a pole-zero combination to control the characteristics of the control system. The DC gain of the voltage feedback loop is given by:

$$A_{VDC} = R_{LOAD} \times G_{CS} \times A_{VEA} \times \frac{V_{FB}}{V_{OUT}}$$

Where A_{VEA} is the error amplifier voltage gain, 200V/V; G_{CS} is the current sense transconductance, 9A/V; R_{LOAD} is the load resistor value.

The system has two poles of importance. One is due to the compensation capacitor (C3), the output resistor of error amplifier. The other is due to the output capacitor and the load resistor. These poles are located at:

$$f_{P1} = \frac{G_{EA}}{2\pi \times C3 \times A_{VEA}}$$

$$f_{P2} = \frac{1}{2\pi \times C2 \times R_{LOAD}}$$

Where, G_{EA} is the error amplifier transconductance, 60μA/V.

The system has one zero of importance, due to the compensation capacitor (C3) and the compensation resistor (R3). This zero is located at:

$$f_{Z1} = \frac{1}{2\pi \times C3 \times R3}$$

The system may have another zero of importance, if the output capacitor has a large capacitance and/or a high ESR value. The zero, due to the ESR and capacitance of the output capacitor, is located at:

$$f_{ESR} = \frac{1}{2\pi \times C2 \times R_{ESR}}$$

In this case (as shown in Figure 2), a third pole set by the compensation capacitor (C6) and the compensation resistor (R3) is used to compensate the effect of the ESR zero on the loop gain. This pole is located at:

$$f_{P3} = \frac{1}{2\pi \times C6 \times R3}$$

The goal of compensation design is to shape the converter transfer function to get a desired loop gain. The system crossover frequency where the feedback loop has the unity gain is important. Lower crossover frequencies result in slower line and load transient responses, while higher crossover frequencies could cause system unstable. A good rule of thumb is to set the crossover frequency to approximately one-tenth of the switching frequency. The Table 3 lists the typical values of compensation components for some standard output voltages with various output capacitors and inductors. The values of the compensation components have been optimized for fast transient responses and good stability at given conditions.

Table 3—Compensation Values for Typical Output Voltage/Capacitor Combinations

V_{OUT} (V)	L (μH)	C2 (μF)	R3 (kΩ)	C3 (pF)	C6
1.8	4.7	47	105	100	None
2.5	4.7 - 6.8	22	54.9	220	None
3.3	6.8 -10	22	68.1	220	None
5	15 - 22	22	100	150	None
12	22 - 33	22	147	150	None

To optimize the compensation components for conditions not listed in Table 3, the following procedure can be used.

1. Choose the compensation resistor (R3) to set the desired crossover frequency. Determine the R3 value by the following equation:

$$R3 = \frac{2\pi \times C2 \times f_C}{G_{EA} \times G_{CS}} \times \frac{V_{OUT}}{V_{FB}}$$

Where f_C is the desired crossover frequency.

2. Choose the compensation capacitor (C3) to achieve the desired phase margin. For applications with typical inductor values, setting the compensation zero, f_{Z1}, below one forth of the crossover frequency provides sufficient phase margin. Determine the C3 value by the following equation:

$$C3 > \frac{4}{2\pi \times R3 \times f_C}$$

3. Determine if the second compensation capacitor (C6) is required. It is required if the ESR zero of the output capacitor is located at less than half of the switching frequency, or the following relationship is valid:

$$\frac{1}{2\pi \times C2 \times R_{ESR}} < \frac{f_S}{2}$$

If this is the case, then add the second compensation capacitor (C6) to set the pole f_{P3} at the location of the ESR zero. Determine the C6 value by the equation:

$$C6 = \frac{C2 \times R_{ESR}}{R3}$$

High Frequency Operation

The switching frequency of MP1584 can be programmed up to 1.5MHz with an external resistor.

With higher switching frequencies, the inductive reactance (X_L) of capacitor comes to dominate, so that the ESL of input/output capacitor determines the input/output ripple voltage at higher switching frequency. As a result of that, high frequency ceramic capacitor is strongly recommended as input decoupling capacitor and output filtering capacitor for such high frequency operation.

Layout becomes more important when the device switches at higher frequency. It is essential to place the input decoupling capacitor, catch diode and the MP1584 (Vin pin, SW pin and PGND) as close as possible, with traces that are very short and fairly wide. This can help to greatly reduce the voltage spike on SW node, and lower the EMI noise level as well.

Try to run the feedback trace as far from the inductor and noisy power traces as possible. It is often a good idea to run the feedback trace on the side of the PCB opposite of the inductor with a ground plane separating the two. The compensation components should be placed closed to the MP1584. Do not place the compensation components close to or under high dv/dt SW node, or inside the high di/dt power loop. If you have to do so, the proper ground plane must be in place to isolate those. Switching loss is expected to be increased at high switching frequency. To help to improve the thermal conduction, a grid of thermal vias can be created right under the exposed pad. It is recommended that they be small (15mil barrel diameter) so that the hole is essentially filled up during the plating process, thus aiding conduction to the other side. Too large a hole can cause ‘solder wicking’ problems during the reflow soldering process. The pitch (distance between the centers) of several such thermal vias in an area is typically 40mil.

External Bootstrap Diode

It is recommended that an external bootstrap diode be added when the input voltage is no greater than 5V or the 5V rail is available in the system. This helps improve the efficiency of the regulator. The bootstrap diode can be a low cost one such as IN4148 or BAT54.

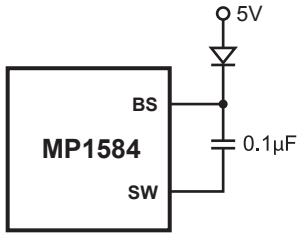


Figure 2—External Bootstrap Diode

This diode is also recommended for high duty cycle operation (when V_{OUT} /V_{IN} >65%) or low V_{IN} (<5Vin) applications.

At no load or light load, the converter may operate in pulse skipping mode in order to maintain the output voltage in regulation. Thus there is less time to refresh the BS voltage. In order to have enough gate voltage under such operating conditions, the difference of V_{IN} –V_{OUT} should be greater than 3V. For example, if the V_{OUT} is set to 3.3V, the V_{IN} needs to be higher than 3.3V+3V=6.3V to maintain enough BS voltage at no load or light load. To meet this requirement, EN pin can be used to program the input UVLO voltage to Vout+3V.

TYPICAL APPLICATION CIRCUITS

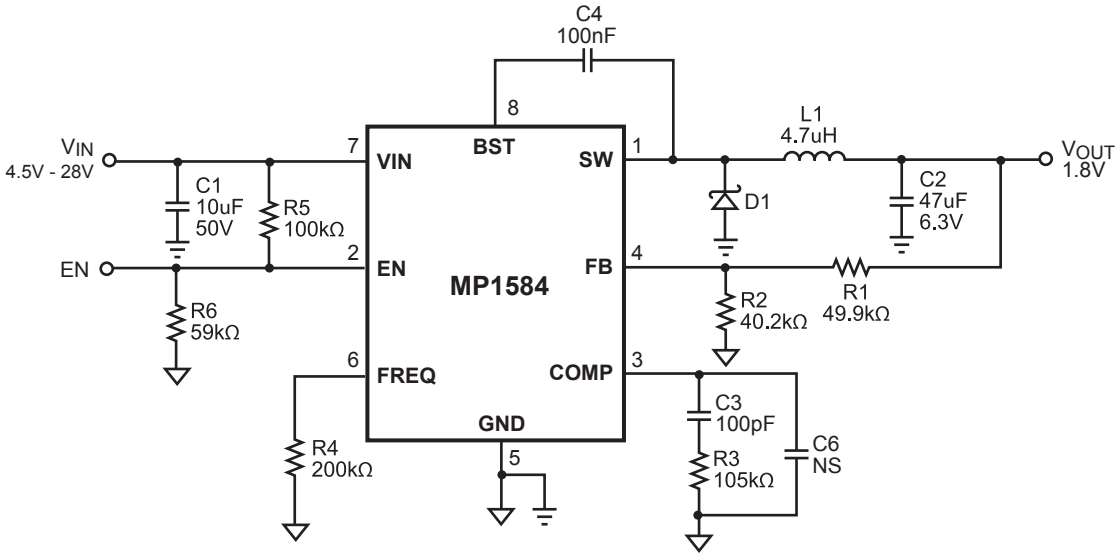


Figure 3—1.8V Output Typical Application Schematic

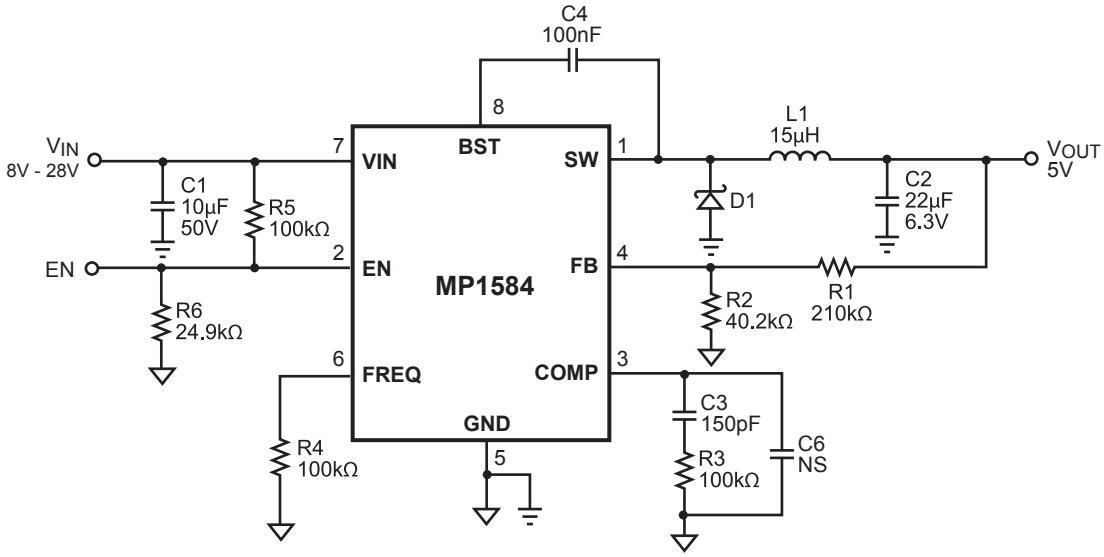


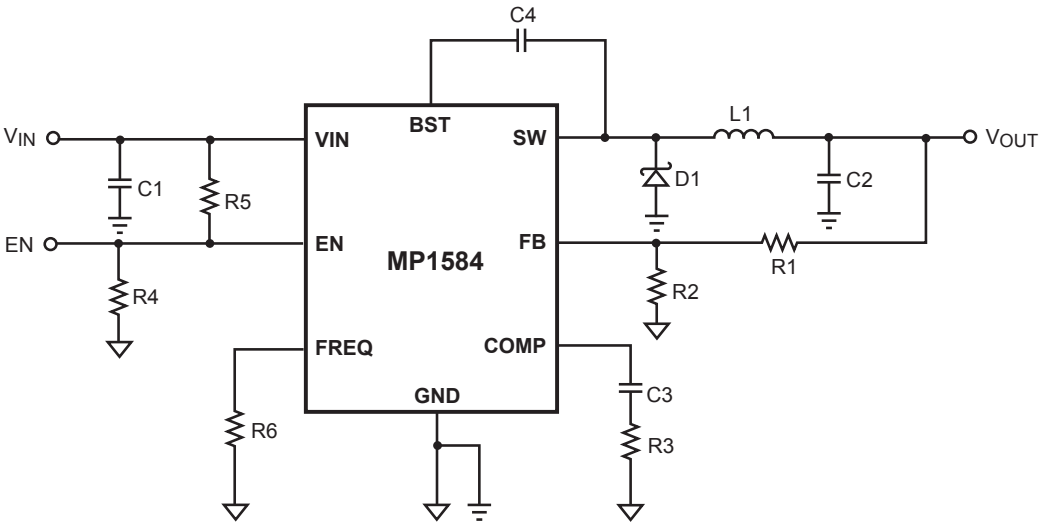
Figure 4—5V Output Typical Application Schematic

PCB LAYOUT GUIDE

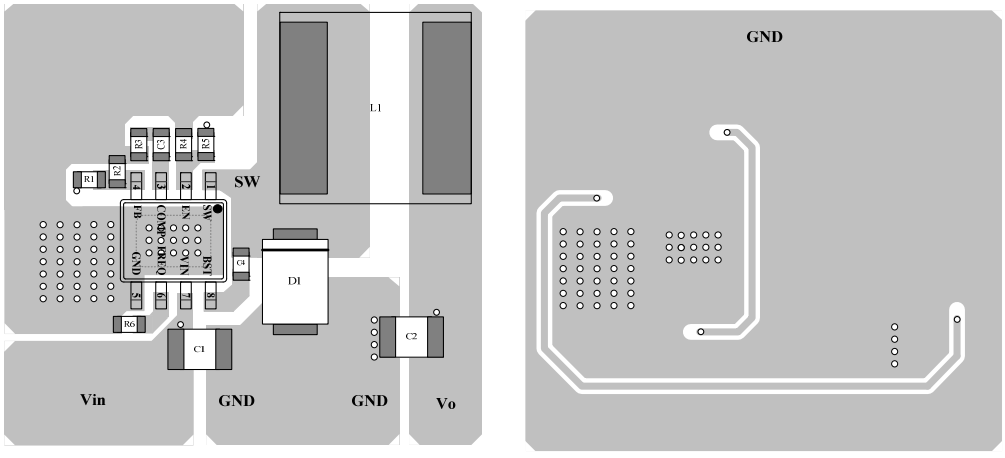
PCB layout is very important to achieve stable operation. It is highly recommended to duplicate EVB layout for optimum performance.

If change is necessary, please follow these guidelines and take Figure 5 for reference.

1) Keep the path of switching current short and minimize the loop area formed by Input cap, high-side MOSFET and external switching diode.
2) Bypass ceramic capacitors are suggested to be put close to the V_{IN} Pin.
3) Ensure all feedback connections are short and direct. Place the feedback resistors and compensation components as close to the chip as possible.
4) Route SW away from sensitive analog areas such as FB.
5) Connect IN, SW, and especially GND respectively to a large copper area to cool the chip to improve thermal performance and long-term reliability.



MP1584 Typical Application Circuit

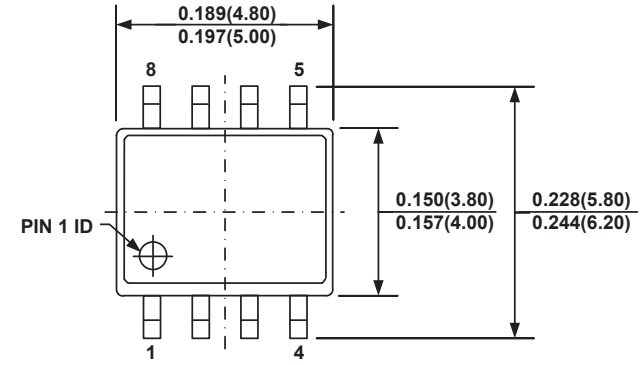


Top Layer **Bottom Layer**

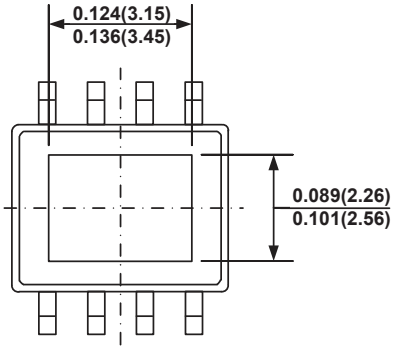
Figure 5—MP1584 Typical Application Circuit and PCB Layout Guide

PACKAGE INFORMATION

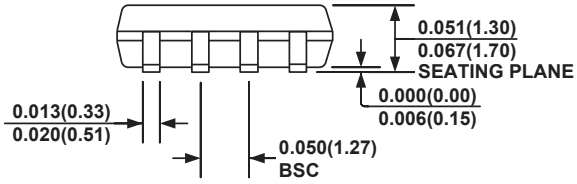
SOIC8E (EXPOSED PAD)



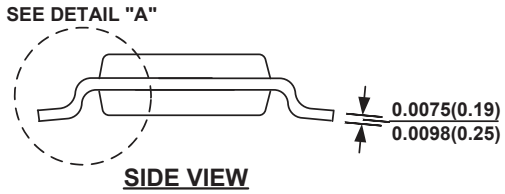
TOP VIEW



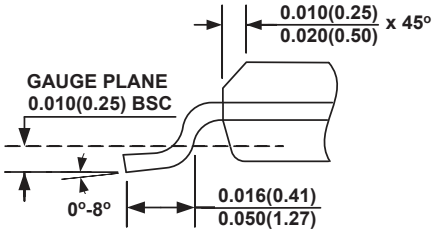
BOTTOM VIEW



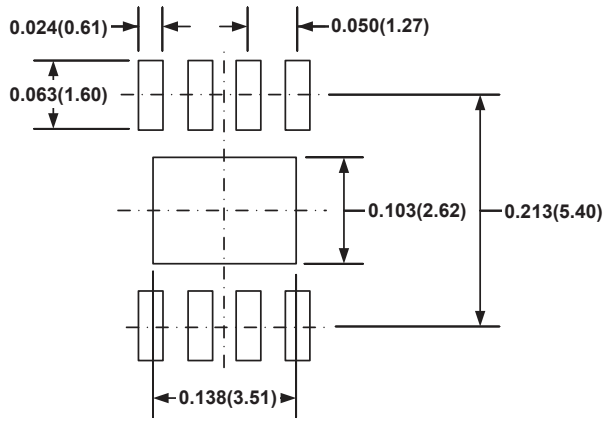
FRONT VIEW



SIDE VIEW



DETAIL "A"



RECOMMENDED LAND PATTERN

NOTE:

1) CONTROL DIMENSION IS IN INCHES. DIMENSION IN BRACKET IS IN MILLIMETERS.
2) PACKAGE LENGTH DOES NOT INCLUDE MOLD FLASH, PROTRUSIONS OR GATE BURRS.
3) PACKAGE WIDTH DOES NOT INCLUDE INTERLEAD FLASH OR PROTRUSIONS.
4) LEAD COPLANARITY (BOTTOM OF LEADS AFTER FORMING) SHALL BE 0.004" INCHES MAX.
5) DRAWING CONFORMS TO JEDEC MS-012, VARIATION BA.
6) DRAWING IS NOT TO SCALE.

MANTECH ELECTRONICS
International Components Distributor
A MOBICON COMPANY

TEL JHB : (011) 493-9307
CAPE : (021) 535-3150
KZN : (031) 309-7686

FAX : (011) 493-9319

sales@mantech.co.za

www.mantech.co.za

I2C interface for LCD

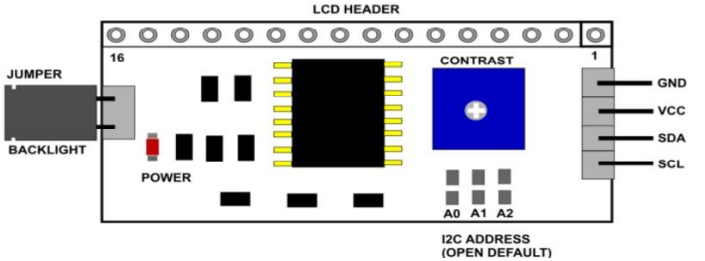
Discription:

This LCD2004 is a great I2C interface for 2x16 and 4x20 LCD displays. With the limited pin resources, your project may be out of resources using normal LCD shield. With this I2C interface LCD module, you only need 2 lines (I2C) to display the information. If you already has I2C devices in your project, this LCD module actually cost no more resources at all. Fantastic for Arduino based projects.

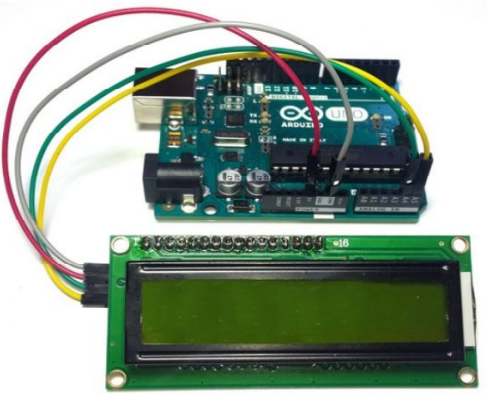
Specification:

Compatible with 16x2 and 20x4 LCD's
Default I2C Address = 0X27
Address selectable - Range 0x20 to 0x27

Board Layout:



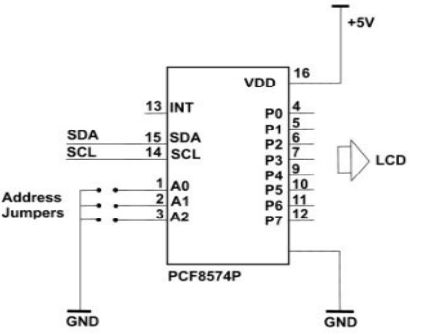
I2C Address Setup:

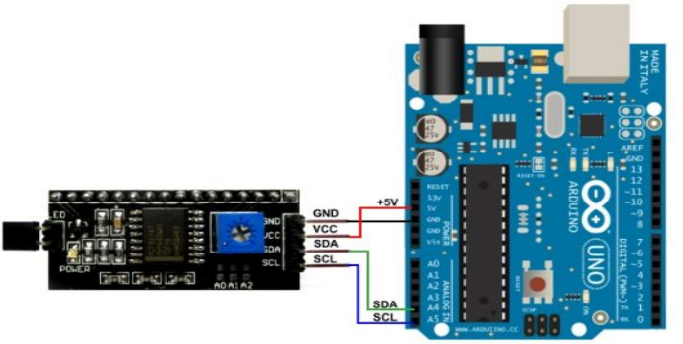


The LCD2004 board utilized the PCF8574 I/O expander. This nifty little chip provides eight bits of parallel I/O addressabl by a I2C bus address – 0x00 to 0x27. SainSmart tied all address leads to Vcc, so the LCD2004 board's I2C address is permanently fixed at hex 27. This is rather limiting since no additional LCD2004s can be added to the bus. Anyway, you simply address the board and write an eight bit value which is then presented on the output pins of the PCF8574, which, in this case, are connected to the HD44780 based LCD screen.

INPUTS			I2C SLAVE ADDRESS
A2	A1	A0	
L	L	L	0x20
L	L	H	0x21
L	H	L	0x22
L	H	H	0x23
H	L	L	0x24
H	L	H	0x25
H	H	L	0x26
H	H	H	0x27

H = Open Jumper L = Close Jumper






```
//Arduino Code
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x27,2,1,0,4,5,6,7,3, POSITIVE); // Initialize LCD Display at address 0x27
                                                       // unmodified backpack
void setup() {
  // activate LCD module
  lcd.begin (16,2); // for 16 x 2 LCD module
  lcd.setBacklightPin(3,POSITIVE);
  lcd.setBacklight(HIGH);
  }

void loop() {
  lcd.home (); // set cursor to 0,0
  lcd.print(" Hello, world!");
  lcd.setCursor (0,1);        // go to start of 2nd line
  lcd.print(millis());
  delay(1000);
  lcd.setBacklight(LOW);      // Backlight off
  delay(500);
  lcd.setBacklight(HIGH);     // Backlight on

  delay(1000);
}       // END
```

Check for more info at
https://arduino-info.wikispaces.com/LCD-Blue-I2C

Capítulo VII. Bibliografía

Para el desarrollo del presente Trabajo de Fin de Grado, ha sido necesario consultar fuentes de información con el propósito de documentar correctamente el proyecto.

[1] L. Llamas, *Robot con omni Wheel controlado por Arduino* [online], 2018. Disponible en:

https://www.luisllamas.es/robot-con-omni-wheel-controlado-por-arduino/

[2] *Historia de la impresión 3D* [online], 2018. Disponible en:

https://www.3dfils.com/de/blog/20_historia3d

[3] *Impresoras 3D ¿Qué son? ¿Cómo funcionan? Todo sobre impresión 3D* [online]. Disponible en:

https://tecnologia-informatica.com/impresoras-3d-que-son-como-funcionan-impresion-3d/

[4] *Materiales de impresión 3D (I): PLA (ácido poliláctico)* [online], 2015. Disponible en:

http://hxx.es/2015/03/12/materiales-de-impresion-3d-i-pla-acido-polilactico/

[5] *Cómo imprimir FilaFlex* [online]. Disponible en:

https://recreus.com/es/content/12-como-imprimir-con-filaflex

[6] *What is Arduino?* [online]. Disponible en:

https://www.arduino.cc/en/Guide/Introduction

[7] Arduino *Language Reference* [online]. Disponible en:

https://www.arduino.cc/reference/en/.

[8] *Arduino UNO REV3* [online]. Disponible en:

https://store.arduino.cc/arduino-uno-rev3

[9] *D1 An Arduino UNO Compatible wifi board based on ESP8266EX* [online], 2017. Disponible en:

https://wiki.wemos.cc/products:d1:d1

[10] *Arduino UNO MEGA 2560 REV3* [online]. Disponible en:

https://store.arduino.cc/mega-2560-r3

[11] L. Llamas, *Tipos de motores rotativos para proyectos de Arduino* [online], 2016. Disponible en:

https://www.luisllamas.es/tipos-motores-rotativos-proyectos-arduino/

[12] *Encoder Installation and Wiring Guide* [online], 2011. Disponible en:

http://encoder.com/core/files/encoder/uploads/files/installation-wiring-guide.pdf

[13] J. E. Crespo, *Motor Paso a Paso con Arduino* [online], 2016. Disponible en:

https://aprendiendoarduino.wordpress.com/category/motores/

[14] *ESP-8266 Modules* [online], 2018. Disponible en:

https://www.esp8266.com/wiki/doku.php?id=esp8266-module-family

[15] *Anexo comandos AT para GSM/GPRS y GPS* [online]. Disponible en:

https://www.prometec.net/comandos-at-gsm-gprs-gps/

[16] J. E. Crespo, *Comunicación Serie Arduino* [online], 2016. Disponible en:

https://aprendiendoarduino.wordpress.com/2016/07/02/comunicacion-serie-arduino/

[17] L. Llamas, *Comunicación de Arduino con puerto serie* [online], 2014. Disponible en:

https://www.luisllamas.es/arduino-puerto-serie/

[18] *Entrenador de comunicaciones digitales*, Manual de teoría EC-796. Promax. 1997.

[19] Walter G. H, Claudio A. C & Alexis M. B., "Análisis de máximo desempeño para WLAN operando a tasas fijas o adaptativas usando el estándar IEEE 802.11 a/b/g" [online] Ingeniare., vol. 15, no. 3, pp. 320-322, 2007. Disponible en:

https://scielo.conicyt.cl/pdf/ingeniare/v15n3/art12.pdf

[20] J. Salazar, "Redes Inalámbricas" [online] TechPedia., pp. 1-22, 2016. Disponible en:

https://upcommons.upc.edu/bitstream/handle/2117/100918/LM01_R_ES.pdf?sequence=1&isAllowed=y

[21] *Protocolos TCP/IP* [online]. Disponible en:

https://www.ibm.com/support/knowledgecenter/es/ssw_aix_72/com.ibm.aix.networkcomm/tcpip_protocols.htm

[22] *User Datagram Protocol* [online]. Disponible en:

https://www.ibm.com/support/knowledgecenter/es/ssw_aix_71/com.ibm.aix.networkcomm/protocols_userdatagram.htm

[23] *Ros Tutorials* [online], 2019. Disponible en:

http://wiki.ros.org/ROS/Tutorials

[24] *Rosserial* [online], 2018. Disponible en:

http://wiki.ros.org/rosserial

[25] *Adding Support for New Hardware* [online], 2011. Disponible en:

http://wiki.ros.org/rosserial_client/Tutorials/Adding%20Support%20for%20New%20Hardware

[26] *Ros_arduino_bridge* [online], 2012. Disponible en:

http://wiki.ros.org/ros_arduino_bridge

[27] *Ros_arduino_bridge (installation)* [online], 2017. Disponible en:

https://github.com/hbrobotics/ros_arduino_bridge

[28] *Librería ESP8266WiFi.h* [online], 2019. Disponible en:

https://github.com/esp8266/Arduino/blob/master/libraries/ESP8266WiFi/src/ESP8266WiFi.h

[29] *Librería LiquidCrystal_I2C.h* [online], 2017. Disponible en:

https://github.com/fdebrabander/Arduino-LiquidCrystal-I2C-library/blob/master/LiquidCrystal_I2C.h

[30] Rosserial_*arduino Tutorials* [online], 2014. Disponible en:

http://wiki.ros.org/rosserial_arduino/Tutorials

[31] A. Nuñez, *ROS serial for ESP8266 over WiFi* [online], 2017. Disponible en:

https://github.com/agnunez/espros

[32] *Configuring and Using a Linux-Supported Joystick with ROS* [online], 2016. Disponible en:

http://wiki.ros.org/joy/Tutorials/ConfiguringALinuxJoystick

[33] *Writing a Teleoperation Node for a Linux-Supported Joystick* [online], 2019. Disponible en:

http://wiki.ros.org/joy/Tutorials/WritingTeleopNode

[34] *Teleop_twist_joy* [online], 2015. Disponible en:

http://wiki.ros.org/teleop_twist_joy

[35] *Publishing Odometry Information over ROS (python)* [online], 2016. Disponible en:

https://gist.github.com/atotto/f2754f75bedb6ea56e3e0264ec405dcf