



## ESCUELA SUPERIOR DE INGENIERÍA

MÁSTER UNIVERSITARIO EN INVESTIGACIÓN EN  
INGENIERÍA DE SISTEMAS Y DE LA COMPUTACIÓN

**Multi Sensor Robot System (SensorRS), Vehículo  
robótico multisensorial de exploración controlado por  
wifi basado en Arduino y Raspberry Pi.**

Manuel López Urbina  
Director: Arturo Morgado Estévez

Cádiz, 22 de septiembre de 2018





## ESCUELA SUPERIOR DE INGENIERÍA

### MÁSTER UNIVERSITARIO EN INVESTIGACIÓN EN INGENIERÍA DE SISTEMAS Y DE LA COMPUTACIÓN

Multi Sensor Robot System (SensorRS), Vehículo robótico multisensorial de exploración controlado por wifi basado en Arduino y Raspberry Pi.

- Departamento: Ingeniería en Automática, Electrónica, Arquitectura y Redes de Computadores
- Director del proyecto: Arturo Morgado Estévez
- Autor del proyecto: Manuel López Urbina

Cádiz, 22 de septiembre de 2018

Fdo: Manuel López Urbina



## **0.1. Agradecimientos**

La culminación de este proyecto significa un paso importante en mi carrera, por lo que me gustaría dedicárselo a todas las personas que me han ayudado a dar un paso más.

En primer lugar me gustaría agradecerle a mi familia el apoyarme y ayudarme durante estos años, y el esfuerzo que han hecho para que pueda progresar en mi vida personal y profesional.

Mención especial para Natalia Luciano, mi pareja, la cual ha estado siempre apoyándome en mis objetivos y tanta paciencia y comprensión me ha mostrado tras tantos días, meses e incluso años de estudio y dedicación.

Agradecimientos a D. Arturo Morgado por su ayuda y dedicación durante la realización y dirección de este proyecto, así como su carácter amable y servicial que han hecho más ameno el trabajo realizado.

También me gustaría agradecérselo a mis compañeros, con los que tantos ratos inolvidables he pasado, y que tanto me han ayudado.

Por último quiero dedicarle este proyecto a todos los estudiantes de informática, en especial a todos los amantes del fascinante mundo de la robótica, a los que espero que mi trabajo les sea de utilidad.



## 0.2. Licencia

Multi Sensor Robot System, en adelante SensorRs, es una aplicación software y hardware libre: usted puede redistribuirlo y/o modificarlo bajo los términos de la Licencia Pública General de GNU según lo publicado por la Free Software Foundation, ya sea la versión 3 de la Licencia, o (a su elección) cualquier versión posterior.

Este programa se distribuye con la esperanza de que sea útil, pero SIN NINGUNA GARANTÍA, incluso sin la garantía implícita de COMERCIALIZACIÓN o IDONEIDAD PARA UN PROPÓSITO PARTICULAR. Ver el GNU General Public License para más detalles.

Este documento ha sido liberado bajo Licencia GFDL 1.3 (GNU Free Documentation License). Se incluyen los términos de la licencia en inglés al final del mismo. Si no es así, consulte [GNU Licenses](#).

Copyright © 2018 Manuel López Urbina.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".



## Resumen

Multi Sensor Robot System (SensorRS), Vehículo robótico multisensorial de exploración controlado por wifi basado en Arduino y Raspberry Pi, es un proyecto robótico elaborado principalmente con una placa Raspberry Pi interconectada por puerto serie con una placa Arduino.

La placa Arduino se ha utilizado para la conexión de una serie de sensores cuya finalidad es la obtención de variables del entorno como temperatura, humedad, iluminación, etc. Estos parámetros son obtenidos mediante la programación del microcontrolador que la placa Arduino incorpora, encargándose, posteriormente, de transmitir información a la placa Raspberry Pi para su procesamiento y envío al servidor web donde se realiza el seguimiento y control del vehículo y los parámetros obtenidos.

La intención principal de este trabajo es la de permitir que usuarios puedan disponer de un vehículo robótico a coste muy reducido y de componentes de fácil adquisición y de reducido coste. Este vehículo podrá ser utilizado para la realización de labores de exploración de zonas en las que o bien no puede acceder una persona debido al reducido tamaño o difícil acceso de las áreas a explorar o bien porque alguna situación peligrosa lo impida. En resumidas cuentas un vehículo robótico con un amplio sistema de telemetría incorporado.

Dicho sistema es configurado en la aplicación web RobotUI para permitir sus control por parte de otros usuarios. RobotUI incorpora un asistente mediante el cual, cualquier usuario sin conocimientos previos de programación, pueda elaborar una interfaz para el control de su dispositivo robótico personalizada y adaptada a sus necesidades de control y a las características del dispositivo en cuestión.

Dicha interfaz permitirá realizar el control de los mencionados dispositivos además de permitir que otros usuarios entren en las salas o canales donde podrán visualizar el manejo que realiza un usuario de su robot en tiempo real donde obtendrán en todo momento las imágenes captadas junto con los diferentes comandos accionados por el usuario que está realizando el control.

Con todo ello, lo que se busca es crear un sistema robótico controlable por el usuario y que transmita información en tiempo real destinado a la exploración de áreas peligrosas y de coste reducido.

**Palabras clave:** Internet, aplicación web, robótica, robots, interfaz de usuario, streaming de vídeo, control remoto, tiempo real, Raspberry Pi, Arduino, sensores, comunicación serie, telemetría.



# Índice general

0.1. Agradecimientos . . . . .	I
0.2. Licencia . . . . .	III
<b>Índice general</b>	I
<b>Índice de figuras</b>	v
<b>1. Introducción</b>	1
1.1. Objetivos . . . . .	5
1.2. Alcance . . . . .	5
1.3. Acerca de este documento . . . . .	7
<b>2. Conceptos básicos</b>	9
2.1. Telemetría . . . . .	9
2.1.1. Aplicaciones . . . . .	10
2.2. Sensor . . . . .	10
2.2.1. clasificación . . . . .	10
2.2.2. Atendiendo a los errores de medición . . . . .	15
2.2.3. Características de los sensores . . . . .	15
2.3. Transmisión y comunicación . . . . .	18
2.4. Socket . . . . .	18
2.5. WebSocket . . . . .	19
2.6. La arquitectura TCP/IP y el modelo OSI . . . . .	19
2.7. Streaming . . . . .	20
2.8. Comunicación serie . . . . .	21
2.8.1. Características . . . . .	21
2.9. PWM . . . . .	22
<b>3. Estado del arte y herramientas utilizadas</b>	25
3.1. Estado del arte . . . . .	25
3.1.1. Introducción . . . . .	25
3.1.2. Referencias . . . . .	26
3.2. Tecnologías software utilizadas . . . . .	29
3.2.1. L <sup>A</sup> T <sub>E</sub> X . . . . .	29
3.2.2. WebStorm . . . . .	29
3.2.3. Fritzing . . . . .	29
3.2.4. Arduino IDE . . . . .	30
3.2.5. Github . . . . .	30

3.2.6. Git . . . . .	30
3.2.7. Amazon Web Services (AWS) . . . . .	31
3.2.8. Node js . . . . .	31
3.2.9. Npm . . . . .	32
3.2.10. SocketIO . . . . .	32
3.2.11. FFmpeg . . . . .	33
3.2.12. Bootstrap . . . . .	34
3.2.13. JQuery . . . . .	34
3.3. Tecnologías hardware y materiales utilizados . . . . .	34
3.3.1. Raspberry Pi Model B . . . . .	35
3.3.2. Arduino . . . . .	36
3.3.3. Arduino Sensor Kit . . . . .	38
3.3.4. Controlador de motores doble puente H - L298N . . . . .	39
3.3.5. Batería LiPo . . . . .	40
3.3.6. Alimentación USB/Protoboard . . . . .	41
3.3.7. Tarjeta de expansión con batería de Litio para Raspberry Pi . . . . .	41
3.3.8. Cámara USB de alta definición . . . . .	42
3.3.9. Gamepad . . . . .	43
<b>4. Requisitos . . . . .</b>	<b>45</b>
4.1. Requerimientos hardware . . . . .	45
4.1.1. Análisis y selección de componentes electrónicos . . . . .	46
4.2. Requerimientos software . . . . .	48
4.3. Especificación . . . . .	49
4.3.1. Requisitos funcionales . . . . .	49
4.3.2. Requisitos no funcionales . . . . .	50
<b>5. Construcción del robot . . . . .</b>	<b>51</b>
5.1. Montaje . . . . .	51
5.1.1. Chasis . . . . .	51
5.1.2. Tracción y dirección . . . . .	51
5.1.3. Interconexión de elementos . . . . .	53
5.1.4. Driver motores . . . . .	54
5.1.5. Motores . . . . .	56
5.2. Alimentación . . . . .	58
5.2.1. Interconexión entre módulos y fijación . . . . .	63
5.2.2. Conexionado general . . . . .	63
5.3. Iluminación . . . . .	67
5.3.1. Leds . . . . .	67
5.3.2. Puntero láser . . . . .	68
5.4. Sensores . . . . .	69
5.4.1. Sensores ultrasonidos . . . . .	69
5.4.2. Sensor de temperatura y humedad DHT11 . . . . .	72
5.4.3. Sensor de gases . . . . .	74
5.4.4. Sensor de sonido . . . . .	76
5.4.5. Fotoresistor . . . . .	78

5.4.6. Sensor de llama . . . . .	79
5.4.7. Sensor de proximidad . . . . .	82
5.4.8. Buzzer o zumbador . . . . .	86
<b>6. Desarrollo software</b>	<b>89</b>
6.1. Metodología de desarrollo . . . . .	89
6.2. Software de control . . . . .	91
6.3. Comunicación interna . . . . .	92
6.3.1. Entrada/Salida Raspberry Pi . . . . .	93
6.3.2. Entrada/Salida Arduino . . . . .	96
6.4. Comunicación externa . . . . .	98
<b>7. Pruebas</b>	<b>109</b>
7.1. Plan de pruebas . . . . .	109
<b>8. Organización temporal</b>	<b>111</b>
8.1. Planificación temporal de tareas . . . . .	114
8.1.1. Hito 1: Planificación y análisis . . . . .	114
8.1.2. Hito 2: Definición de requisitos . . . . .	114
8.1.3. Hito 3: Montaje del vehículo . . . . .	114
8.1.4. Hito 4: Programación del vehículo SensorRS . . . . .	115
8.1.5. Hito 5: Mejoras en la aplicación de control RobotUI . . . . .	115
8.1.6. Hito 6: Documentación . . . . .	115
8.2. Diagrama de Gantt . . . . .	115
<b>9. Guía de Usuario</b>	<b>117</b>
9.0.1. Objetivo de esta guía . . . . .	118
9.0.2. Dirigido a . . . . .	118
9.0.3. Obtener SensorRS . . . . .	118
9.1. Uso de SensorRS . . . . .	118
9.1.1. Configuración . . . . .	118
9.1.2. Programa tu robot . . . . .	119
9.2. Control . . . . .	123
<b>10. Presupuesto</b>	<b>125</b>
<b>11. Conclusiones</b>	<b>127</b>
11.1. Mejoras futuras . . . . .	128
<b>Anexos</b>	<b>129</b>
.1. Arduino, entorno de desarrollo . . . . .	129
.1.1. Descarga e instalación . . . . .	130
.2. Fritzing . . . . .	130
.2.1. Descarga e instalación . . . . .	131
.3. Instalación de Node.js . . . . .	131
.4. Instalación del control de versiones Git . . . . .	132
.4.1. Configuración . . . . .	132

<b>Bibliografía</b>	<b>135</b>
<b>GNU Documentation Free License</b>	<b>139</b>

# Índice de figuras

1.1.	Logo SensorRS <sup>1</sup> . . . . .	3
1.2.	Vehículo SensorRS <sup>2</sup> . . . . .	4
1.3.	Diagrama representativo de la estructura de SensorRS, sus canales de comunicación y sus niveles de jerarquía. . . . .	6
1.4.	Imagen de los diferentes elementos que integran SensorRS. . . . .	7
2.1.	Esquema básico de comunicación en un sistema de telemetría. . . . .	10
2.2.	Esquema de clasificación de los sensores según criterio. . . . .	11
2.3.	Ejemplos de señal digital y analógica. . . . .	12
2.4.	Esquema clasificatorio de las características estáticas y dinámicas de los sensores. . . . .	16
2.5.	Representación de los modelos de capas o niveles OSI y TCP/IP. . . . .	20
2.6.	Diagrama representativo de la comunicación serie frente a la comunicación en paralelo. . . . .	21
2.7.	Representación de diferentes ciclos de trabajo de una señal PWM. . . . .	23
3.1.	Comparativa del consumo de ancho de banda entre HTTP y WebSockets. . . . .	27
3.2.	Gráfico representativo de las etapas en una petición HTTP frente a una petición por Websocket [31]. . . . .	27
3.3.	Diagrama de bloques del sistema referenciado en [31]. . . . .	28
3.4.	Formato de mensaje para la comunicación entre Arduino y Raspberry Pi [14]. . . . .	28
3.6.	Imagen de una Raspberry Pi 3 Model B . . . . .	36
3.7.	Placa Arduino MEGA 2560 . . . . .	37
3.8.	Características Arduino MEGA 2560 . . . . .	38
3.9.	Pack de sensores para Arduino . . . . .	39
3.10.	Imagen de la controladora de motores doble puente H - L298N utilizada. . . . .	40
3.11.	Imagen de la batería LiPo utilizada. . . . .	41
3.12.	Adaptador de entrada USB a protoboard utilizado. . . . .	41
3.13.	Imagen de la tarjeta de expansión con batería de Litio utilizado. . . . .	42
3.14.	Imagen de la cámara USB utilizada. . . . .	43
3.15.	Vista del gamepad utilizado. . . . .	43
4.1.	Esquema GPIO de una Raspberry Pi Model B+. . . . .	47
4.2.	Placa Arduino Mega donde se visualiza la disposición de sus pines E/S. . . . .	48
5.1.	Imagen de un servomotor. . . . .	52

5.2.	Pines de entrada/salida del módulo L298N empleado. . . . .	55
5.3.	Diagrama de funcionamiento del puente H. . . . .	56
5.4.	Sistema de accionamiento de la dirección. . . . .	57
5.5.	Vista del motor de tracción. . . . .	57
5.6.	Conexionado del conjunto motores, driver y Arduino. . . . .	58
5.7.	Conjunto Raspberry Pi y módulo de expansión de alimentación. . . . .	59
5.8.	Batería LiPo que alimenta los motores. . . . .	60
5.9.	Alojamiento de la batería LiPo. . . . .	60
5.10.	Adaptador de entrada USB a protoboard. . . . .	61
5.11.	Esquemático del adaptador de entrada USB a protoboard. . . . .	61
5.12.	Vista del conector J1. . . . .	62
5.13.	Vista del conector J2 y J3. . . . .	62
5.14.	Vista del conector J4 y J5. . . . .	63
5.15.	Cables de interconexión para protoboard. . . . .	63
5.16.	Vista del conexionado general del robot. . . . .	64
5.17.	Esquemático del conexionado general del robot. . . . .	66
5.18.	Vehículo SensorRS con la iluminación activada. . . . .	67
5.19.	Conexionado de los leds de iluminación. . . . .	68
5.20.	Vehículo SensorRS con el puntero láser situado en un lateral. . . . .	69
5.21.	Sensores ultrasonidos empleados en el proyecto. . . . .	70
5.22.	Pulsos para la correcta lectura del sensor ultrasonidos. . . . .	70
5.23.	Directividad del sensor HC-SR04. . . . .	71
5.24.	Conexionado del sensor HC-SR04 con Arduino. . . . .	72
5.25.	Sensores ultrasónicos situados en la parte trasera del vehículo. . . . .	72
5.26.	Sensor de temperatura DHT11. . . . .	73
5.27.	Disposición de los bits de datos del DHT11. . . . .	73
5.28.	Conexionado de DHT11 con Arduino. . . . .	74
5.29.	Sensor MQ-2. . . . .	75
5.30.	Vista del potenciómetro del sensor MQ-2. . . . .	76
5.31.	Sensor MQ-2 utilizando el pin digital del microcontrolador. . . . .	76
5.32.	Amplificación de la señal de audio recibida. . . . .	77
5.33.	Vista del sensor de audio KY-038 utilizado. . . . .	77
5.34.	Sensor KY-038 utilizando el pin digital del microcontrolador. . . . .	78
5.35.	Vista del fotoresistor utilizado. . . . .	78
5.36.	Vista del fotoresistor utilizado. . . . .	79
5.37.	Espectro visible por el ojo humano. . . . .	80
5.38.	Vista del sensor de llama YG1006 utilizado. . . . .	80
5.39.	Gráfico de sensibilidad espectral del sensor YG1006. . . . .	81
5.40.	Vista del conexionado del sensor de llama YG1006. . . . .	82
5.41.	Sensor de llama instalado en la parte delantera del vehículo. . . . .	82
5.42.	Sensor GP2D12. . . . .	83
5.43.	Curva de tensión de salida según la distancia al obstáculo. . . . .	83
5.44.	Conexiones del GP2D12. . . . .	84
5.45.	Conexionado del GP2D12. . . . .	85
5.46.	Sensor de proximidad Sharp GP2D12 instalado en la parte frontal del vehículo. . . . .	85

5.47. Representación de la variación de volumen del material piezoeléctrico al paso de la corriente. . . . .	86
5.48. Conexionado del buzzer. . . . .	87
6.1. Subsistemas existentes en el proyecto junto con el modelo de ciclo de vida utilizado para su desarrollo. . . . .	90
6.2. Diagrama de casos de uso para la interacción con el robot. . . . .	90
6.3. Autómata representativo de los diferentes estados del robot. . . . .	91
6.4. Diferentes flujos de comunicación abiertos por SensorRS. . . . .	93
6.5. Canales de comunicación abiertos externos a SensorRS. . . . .	99
7.1. Vehículo SensorRS en funcionamiento por diferentes tipos de pista. . . .	110
8.1. Panel de actividades - Trello . . . . .	111
8.2. Descomposición de las tareas implicadas en la construcción del vehículo robótico SensorRS (Primera Parte). . . . .	113
8.3. Diagrama de Gantt. Desarrollo del proyecto. . . . .	116
9.1. Página principal RobotUI. . . . .	119
9.2. Panel informativo donde se aprecia el identificador del robot SensorRS en la aplicación. . . . .	120
9.3. Robot a la espera de conexión entrante. . . . .	123
9.4. Vista del gamepad utilizado. . . . .	124
1. Entorno de desarrollo de Arduino. . . . .	129
2. Enlace para la descarga del software de Arduino. . . . .	130
3. Enlace de acceso al IDE de Arduino en su versión web. . . . .	130
4. Sistemas operativos disponibles para Fritzing. . . . .	131



# Capítulo 1

## Introducción

La robótica es una rama de la ingeniería, la cual se ocupa del diseño, construcción, operación y uso de robots<sup>1</sup>, así como sistemas informáticos para su control, retroalimentación sensorial y procesamiento de información. Entre las diversas disciplinas aplicadas a la robótica podemos encontrar: la mecánica, la electrónica, la informática, la inteligencia artificial, la ingeniería de control y la física, entre otras muchas. De lo cual podemos considerar la robótica como una ciencia multidisciplinaria.

Un robot es, por tanto, una máquina capaz de interactuar con su entorno. Si es móvil, a menos que se mueva en un espacio absolutamente acotado y preparado para él, deberá ser capaz de adaptar sus movimientos y sus acciones de interacción en base a las características físicas de los ambientes con los que se encuentre y los objetos que hay en ellos.

Para lograr esta capacidad de adaptación, lo primero que necesitan los robots es tener conocimiento del entorno, resultando ésto absolutamente imprescindible. Para conocer el entorno los seres vivos recibimos la información mediante los sentidos. Los robots, en infinidad de ocasiones buscando su inspiración en la naturaleza, deben disponer de sistemas que les permitan saber dónde se encuentran, cómo es el lugar en el que están, a qué condiciones físicas se enfrentan, dónde están los objetos con los que deben interactuar, sus parámetros físicos, etc.

Para esto se utilizan diversos tipos de sensores (o captadores), con un rango de complejidad y sofisticación que varía desde algunos bastante simples a otros con altos niveles de sofisticación de hardware y más aún de complejidad de programación.

Un sensor<sup>2</sup> consta de algún elemento sensible a una magnitud física, como por ejemplo la intensidad o color de la luz, temperatura, presión, magnetismo, humedad, y debe ser capaz, por su propias características, o por medio de dispositivos intermedios, de transformar esa magnitud física en un cambio eléctrico que se pueda alimentar en un

---

<sup>1</sup>Robot: Máquina automática programable capaz de realizar determinadas operaciones de manera autónoma y sustituir a los seres humanos en algunas tareas, en especial las pesadas, repetitivas o peligrosas; puede estar dotada de sensores, que le permiten adaptarse a nuevas situaciones.

<sup>2</sup>En el capítulo de conceptos básicos, [2.2](#) puede acceder a información más detallada sobre los sensores, descripción, características, etc.

circuito para que la utilice directamente, o bien, pasando por una etapa previa que la condicione (amplificando, filtrando, etc.). Todo ello para que una vez procesada la señal por el robot éste actúe en consecuencia al parámetro recibido.

En la actualidad, estos sensores se encuentran presentes en robots de todo tipo. Éstos robots son muy utilizados en plantas de fabricación, montaje y embalaje, en transporte, en exploraciones en la Tierra y en el espacio, cirugía, armamento, investigación en laboratorios y en la producción en masa de bienes industriales o de consumo. También resultan de gran utilidad en la resolución de tareas peligrosas, existiendo multitud de aplicaciones como, limpieza de residuos tóxicos, minería, búsqueda, rescate de personas y localización de minas terrestres o accidentes nucleares.

Por tanto, centrándonos en el área de resolución de tareas peligrosas para las personas, un robot debe cumplir una serie de requerimientos para que cumpla adecuadamente con una serie de cometidos:

- La movilidad y destreza para maniobrar en zonas con multitud de obstáculos tales como escombros, típicos de las zonas de desastre o catastróficas.
- Capacidad de manipulación y utilización de diverso surtido de herramientas diseñadas para los seres humanos.
- Capacidad para ser controlado por los seres humanos que hayan tenido poca o ninguna formación robótica.
- Autonomía ya sea total o parcial en el nivel de tarea de toma de decisiones sobre la base de los comandos del operador y entradas de los sensores.

De todo lo anterior se extrae, por tanto, la necesidad de elaborar un sistema robótico operado por una persona y que permita el análisis del entorno, junto con la posibilidad de controlarlo mediante una aplicación software y una conexión a internet. O dicho de otro modo, un sistema de **telemetría** definida como una tecnología que permite recabar información de un entorno sin necesidad de estar en él presente <sup>3</sup>.

Dado que la información que se desea medir para posteriormente transmitirla puede ser muy variada debido a sus múltiples orígenes, ya sea procedente de una cámara de vídeo, de alguno de los distintos tipos de sensores existentes o de cualquier otra fuente de datos que podamos imaginar, hacen que desde el punto de vista de la comunicación, la información que se desea transmitir sea irrelevante, por lo que una vez establecida la forma en la que se envían y se reciben los datos, podremos añadir, reemplazar o modificar los sensores para adaptar el proyecto tipo desarrollado a un entorno concreto o diferente al inicialmente pensado sin necesidad de redefinir los procesos de comunicación ya creados.

---

<sup>3</sup>En la sección 2.1 referente al capítulo de conceptos básicos puede acceder a un apartado específico destinado a la telemetría.

## CAPÍTULO 1. INTRODUCCIÓN

---

Así que dadas las motivaciones existentes y, junto que la programación web y la robótica son temas que causan en mi un especial interés, hicieron que me lanzara a la elaboración de este proyecto que unifica ambos campos anteriormente citados. Más si cabe que el presente trabajo sirve de complemento a la aplicación RobotUI anteriormente desarrollada por mí.

Así surgió *SensorRS*, *Multi-sensor Robot System*.



Figura 1.1: Logo SensorRS <sup>4</sup>.

*Multi Sensor Robot System (SensorRS), Vehículo robótico multisensorial de exploración controlado por WiFi basado en Arduino y Raspberry Pi*, nombre del sistema resultante, será una combinación de un elemento software (aplicación web RobotUI) y hardware (SensorRC, Multisensor Robot System) surgido como un sistema para sacar el máximo aprovechamiento de la aplicación RobotUI para la que se elaborará un dispositivo robótico de propósito general cuya finalidad es análisis y exploración del entorno gracias a los sensores incorporados.

SensorRS se compone de un vehículo controlado vía WiFi el cual responde a una serie de señales, *comandos*<sup>5</sup>, a los que responde realizando determinadas acciones. Por otra parte también dispone de una cámara para la captura de imágenes.

La interfaz de control generada en la aplicación web RobotUI<sup>6</sup>, se configurará de tal manera que permita el control del vehículo desarrollado en el presente proyecto. Esta aplicación permite dar de alta dispositivos robóticos para su control y retransmisión de su funcionamiento en vivo con otros usuarios.

---

<sup>4</sup>Logotipo de SensorRS, Multi-sensor Robot Sysmtem.

<sup>5</sup> Comando: instrucción u orden que el usuario proporciona a un sistema informático, desde la línea de comandos (como una shell) o desde una llamada de programación.

<sup>6</sup> Aplicación anteriormente desarrollada por Manuel López Urbina para el control de dispositivos robóticos de propósito general. Acceso al código fuente: <https://github.com/lopi87/SAILS-RobotUI>



(a) Vista frontal-superior



(b) Vista lateral



(c) Vista posterior-lateral

Figura 1.2: Vehículo SensorRS <sup>7</sup>.

---

<sup>7</sup>Vehículo desarrollado con la finalidad de integrarlo en la aplicación RobotUI. Su desarrollo y características quedan descritas en el capítulo 5.

## 1.1. Objetivos

Como hemos visto, se requiere de multitud de conocimientos a la hora de afrontar un proyecto robótico con ciertas garantías. Este proyecto trata de la elaboración de un vehículo robótico de fácil construcción mediante la utilización de componentes fácilmente adquiribles y de reducido coste.

Dicho robot irá destinado al área de la exploración, buscando, al menos, reducir la exposición de personas a zonas peligrosas ayudando a las labores de rastreo o incursión en lugares en principio inalcanzables o de dificultoso acceso.

En la presente memoria se abarcará la descripción del diseño, construcción y control de un Robot autómata teledirigido a base de una placa Arduino y Raspberry Pi dotado de multitud de sensores que ayuden a proporcionar información al usuario del entorno más próximo en el que el robot se encuentra.

El robot contendrá una arquitectura basada en microcontroladores<sup>8</sup> la cual permitirá crear un sistema fácilmente escalable donde se agregue nuevos sensores sin que ello suponga un elevado esfuerzo.

En definitiva, este proyecto intenta demostrar que sin la necesidad de grandes conocimientos en programación, junto con pequeñas nociones de electrónica básica, que cualquier persona pueda elaborar un proyecto robótico de similares características. Evitando que multitud de personas vean que no disponer de grandes conocimientos en la materia sean un impedimento a la hora de comenzar a desarrollar sus ideas.

## 1.2. Alcance

El sistema está basado en una Raspberry Pi que recibirá el gobernada vía wifi a través de la aplicación web RobotUI constituyendo éste el nivel superior del sistema. Como nivel inferior, disponemos de una placa Arduino la cual será gestionada por la Raspberry Pi y que será la encargada de controlar todos los sensores. La Figura 1.4 muestra un diagrama representativo de la estructura global del sistema y sus canales de comunicación.

---

<sup>8</sup>Un microcontrolador (abreviado C, UC o MCU) es un circuito integrado programable, capaz de ejecutar las órdenes grabadas en su memoria. Está compuesto de varios bloques funcionales, los cuales cumplen una tarea específica. Un microcontrolador incluye en su interior las tres principales unidades funcionales de una computadora: unidad central de procesamiento, memoria y periféricos de entrada/salida.

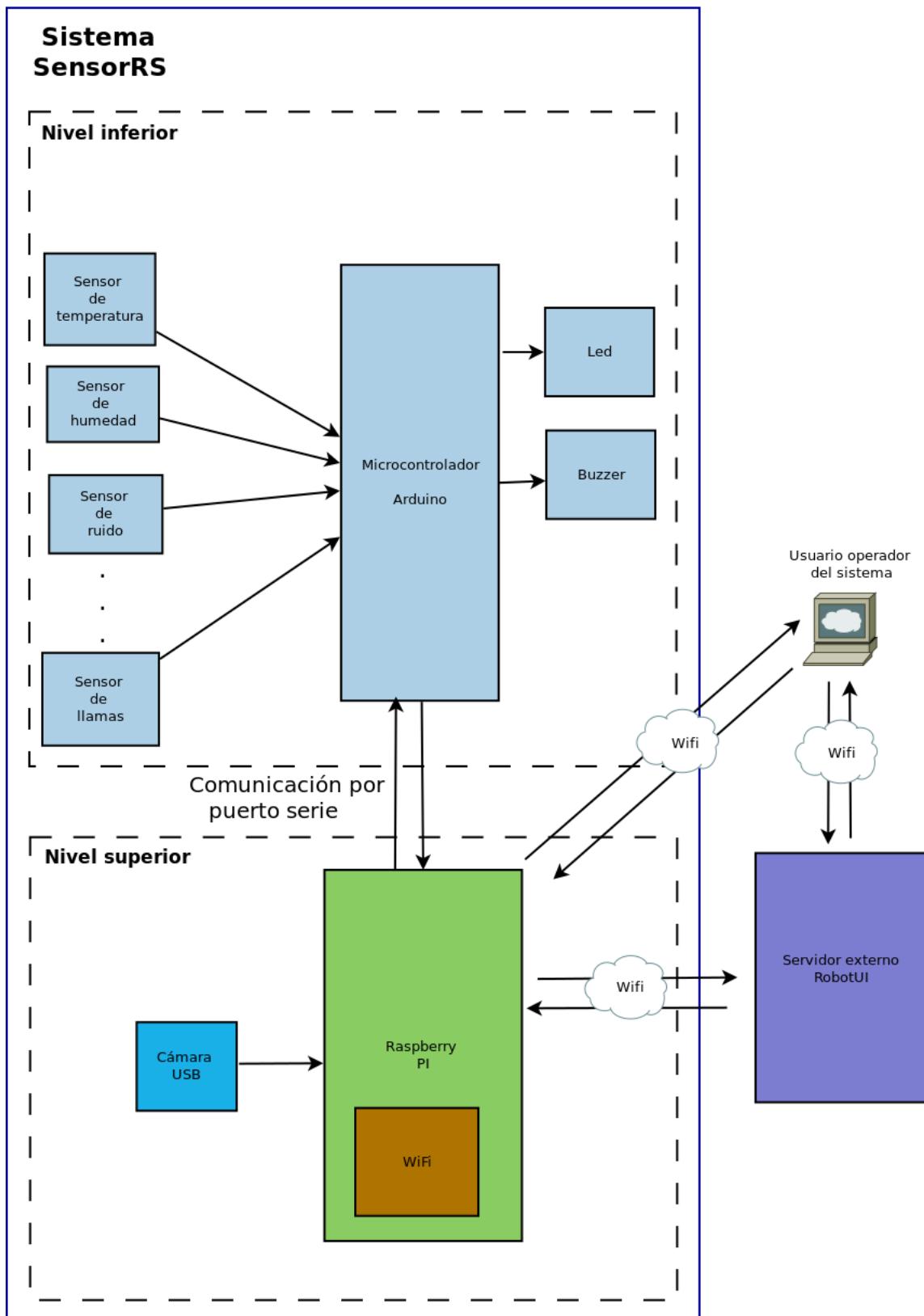


Figura 1.3: Diagrama representativo de la estructura de SensorRS, sus canales de comunicación y sus niveles de jerarquía.

Además el sistema se encargará de una transmisión de vídeo en tiempo real de las imágenes captadas a través de una cámara conectada a la Raspberry Pi.

SensorRS incorporará los elementos necesarios para la integración de diferentes sensores en la aplicación (Lectura, escritura de valores, ...) pudiendo ser éstos de tipos muy variados, temperatura, humedad, sonidos, de luminosidad, medición de distancias, sensor de fuego, de lluvia y de impacto. El sistema estará capacitado para soportar mayor número de sensores.

En el presente trabajo se deberán abarcar diferentes aspectos como la descripción del diseño, construcción y control de un robot teledirigido junto con una descripción de los diferentes procedimientos, bibliotecas y código elaborado para la interconexión e intercambio de información entre la placa Arduino y Raspberry Pi.

A continuación se muestra una imagen general del vehículo donde se aprecian los diferentes elementos conectados.

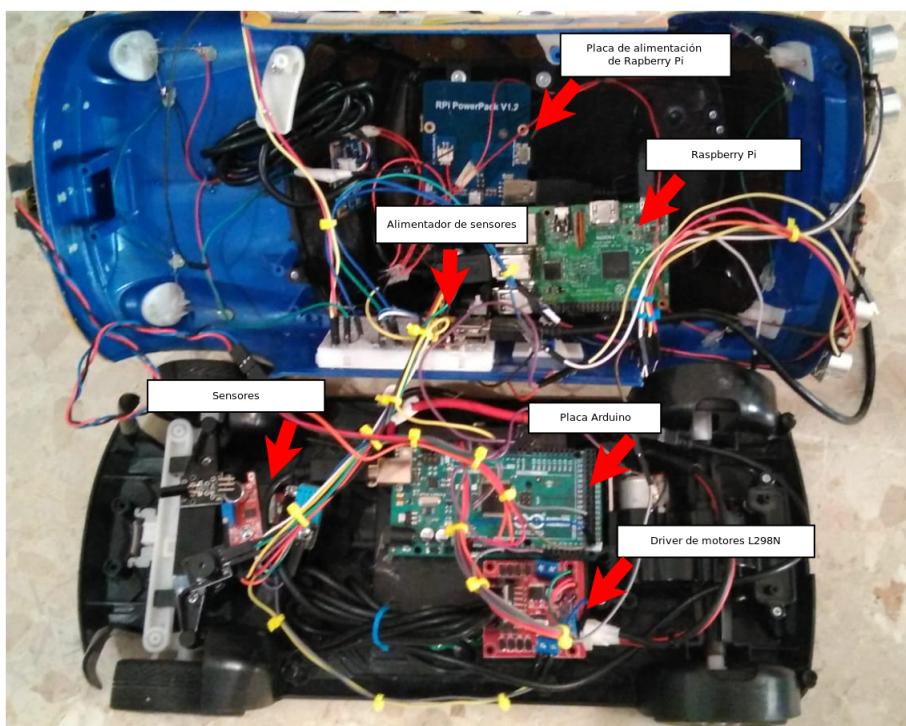


Figura 1.4: Imagen de los diferentes elementos que integran SensorRS.

### 1.3. Acerca de este documento

El documento se ha sido elaborado en un lenguaje claro y sencillo para permitir que un estudiante universitario de Ingeniería Informática pueda comprender los contenidos sin apenas dificultad añadida.

Este documento se organiza en los siguientes capítulos:

- En el capítulo 1, Introducción, se comentan las razones que han motivado la creación de este proyecto, así como el propósito del mismo.
- En el capítulo 2, Conceptos básicos, se incluyen definiciones de aquellos conceptos considerados de interés para la correcta comprensión del contenido de la presente memoria.
- En el capítulo 3, Estado del arte y herramientas utilizadas, se realiza una descripción de las diferentes elementos hardware y software empleados durante el desarrollo del proyecto y necesarios para la utilización del mismo. Así como una breve descripción del conocimiento acumulado y tecnologías existentes hasta la fecha.
- En el capítulo 4, Análisis de requisitos, se realiza un análisis sobre la metodología empleada para el desarrollo del proyecto, describiendo los modelos de ciclo de vida utilizados en el caso del desarrollo software y la descripción de los requisitos funcionales y no funcionales del mismo.
- En el capítulo 5, Construcción del robot, se recogen aquellos aspectos referentes al montaje del robot SensorRS, conexiónado y sensores utilizados.
- En el capítulo 6, Desarrollo software, se recogen aquellos aspectos referentes a la programación del robot y los diferentes canales de comunicación existentes.
- En el capítulo 7, Pruebas, se detallan las pruebas a las que se ha sometido el sistema.
- En el capítulo 8, Organización temporal, se recoge todo lo que concierne a la distribución y duración de cada una de las tareas llevadas a cabo durante el desarrollo del proyecto que el presente documento describe.
- En el capítulo 9, Guía de usuario, se describen los diferentes aspectos necesarios para la correcta utilización del conjunto software y hardware de los que se compone el presente proyecto.
- En el capítulo 10, Presupuesto, se hace un desglose de los diferentes gastos aplicables al proyecto.
- En el capítulo 11, Comentarios finales, se hace mención de las conclusiones obtenidas tras la realización del proyecto además de las posibles mejoras aplicables.
- En el apéndice Anexos A, aparecen los manuales de instalación del software que ha sido necesario para la realización del proyecto.

# **Capítulo 2**

## **Conceptos básicos**

En el presente capítulo se recogen aquellos conceptos, definiciones, protocolos y diferentes aspectos que resultan de especial interés y que ayudarán a la comprensión de los diferentes puntos tratados en el resto de la memoria sin profundizar demasiado en detalles técnicos.

Todos estos conceptos se encuentran estrechamente ligados con tecnologías de la comunicación y transmisión de información y señales. Además de una sección destinada a los sensores, recogiendo sus características y tipos existentes a modo general.

### **2.1. Telemetría**

La telemetría es una tecnología destinada la medición de magnitudes físicas de manera remota para un posterior envío de la información hacia un sistema que actúa como operador. La palabra telemetría procede de las palabras griegas *tele* (tele), la cual significa distancia, y la palabra (*metron*), que indica medida [7].

El envío de información se realiza típicamente mediante comunicación inalámbrica, característica fundamental de estos sistemas aunque también se pueden emplear otros medios como redes de computadoras, fibra óptica, GSM, etc. Los sistemas de telemetría reciben las órdenes o instrucciones desde un sistema externo que hace las funciones de control.

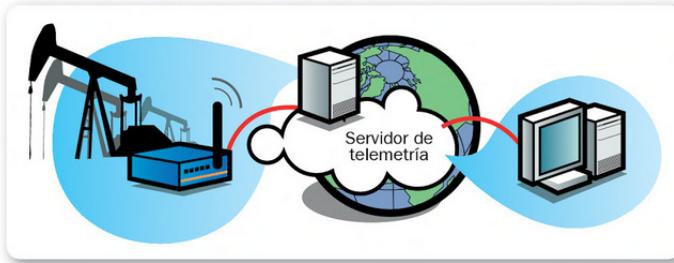


Figura 2.1: Esquema básico de comunicación en un sistema de telemetría.

### 2.1.1. Aplicaciones

La telemetría se utiliza tanto en sistemas simples como en sistemas de muy elevada complejidad. Estos sistemas pueden ser de muy diversa índole como por ejemplo, naves espaciales, plantas químicas, redes de suministro (agua, gas, electricidad...) entre muchos otros. Dado que la telemetría posibilita la monitorización automática y el registro de las mediciones permiten el envío de alertas o alarmas al centro de control con el fin de que el funcionamiento sea seguro y eficiente.

Por ejemplo, las agencias espaciales como la NASA, la UK Space Agency, la ESA y otras, utilizan sistemas de telemetría y de telecontrol para operar con satélites y naves espaciales, o en la Fórmula 1 donde se utilizan para la medición de las condiciones de la pista y transmitir al piloto información relevante para bajar sus tiempos por vuelta. En las fábricas donde el monitoreo de variables que afecten a la producción permitirá elaborar sistemas de mayor eficiencia energética reduciendo los costes.

## 2.2. Sensor

Un sensor es un dispositivo módulo o subsistema que permite cuantificar una variable física cuyo propósito es detectar eventos o cambios en su entorno y enviar la información a otros componentes electrónicos, con frecuencia un procesador de computadora. Un sensor siempre es utilizado en combinación con otros dispositivos electrónicos, para su posterior procesamiento, transmisión, activación de una acción, etc. [15].

### 2.2.1. clasificación

Dada la gran cantidad de sensores existentes resulta necesario aplicar una clasificación con la finalidad de comprender mejor su naturaleza y funcionamiento. Los sensores se pueden clasificar de diversas formas según los distintos criterios atendidos [3]:

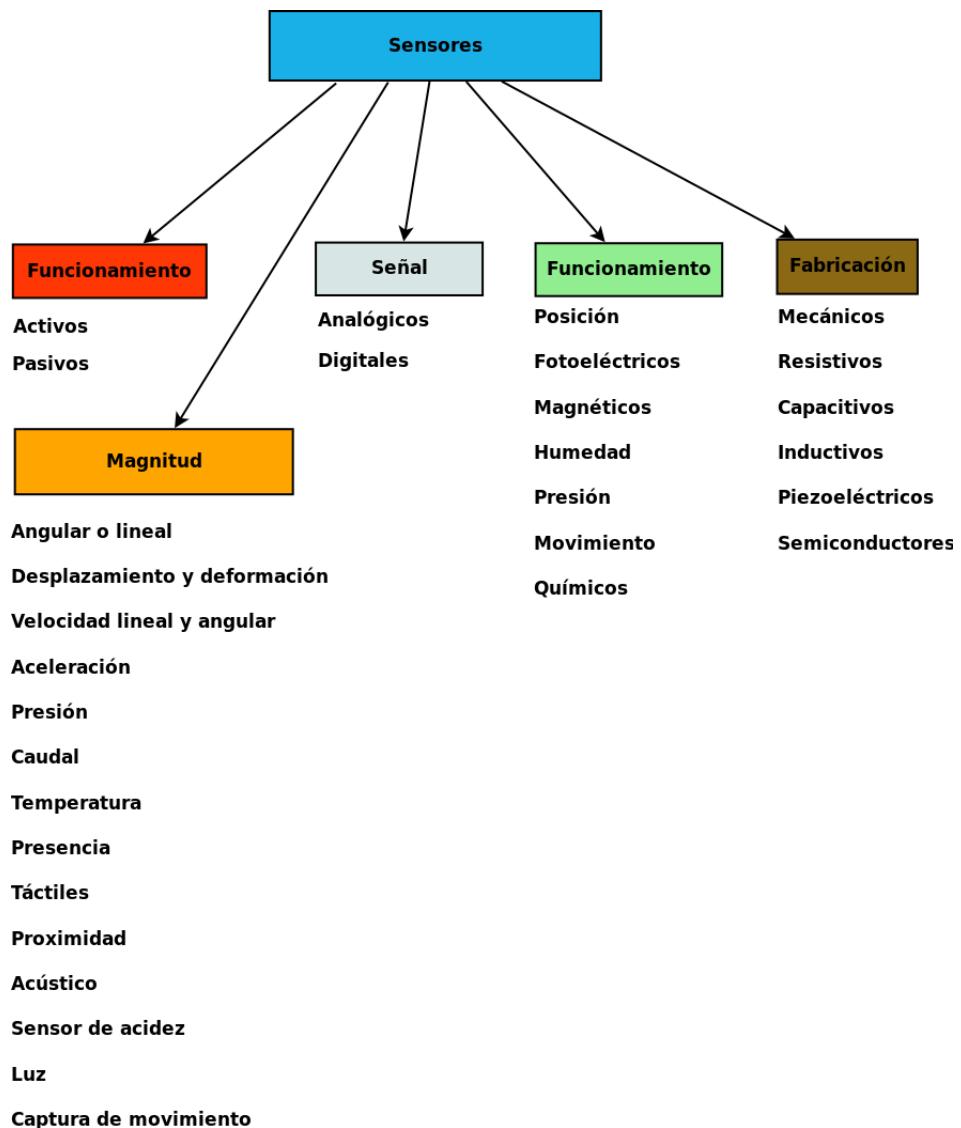


Figura 2.2: Esquema de clasificación de los sensores según criterio.

### 2.2.1.1. Atendiendo a su funcionamiento:

- **Activos:** Requieren de una fuente externa de energía de la que recibir alimentación de corriente para su funcionamiento.
- **Pasivos:** No requieren de una fuente de energía externa, sino que las propias condiciones medioambientales son suficientes para que funcionen según su cometido.

### 2.2.1.2. Atendiendo a las señales que proporcionan:

- **Analógicos:** Proporcionan la información mediante una señal analógica (tensión, corriente), es decir, que pueden tomar infinidad de valores entre un mínimo y un máximo.

- **Digitales:** Proporcionan la información mediante una señal digital que puede ser un 0 o un 1 lógicos, o bien un código de bits.

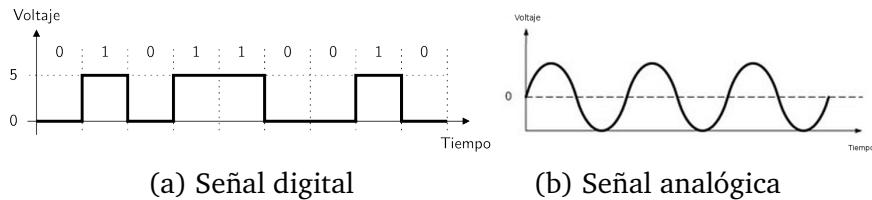


Figura 2.3: Ejemplos de señal digital y analógica.

#### **2.2.1.3. Atendiendo a la naturaleza de su funcionamiento:**

- **Posición:** Son aquellos que experimentan variaciones en función de la posición que ocupan en cada instante los elementos que lo componen.
  - **Fotoeléctricos:** Son aquellos que experimentan variaciones en función de la luz que incide sobre los mismos.
  - **Magnéticos:** Son aquellos que experimentan variaciones en función de la temperatura del lugar donde están ubicados.
  - **Humedad:** Son aquellos que experimentan variaciones en función del nivel de humedad existente en el medio en el que se encuentran.
  - **Presión:** Son aquellos que experimentan variaciones en función de la presión a la que son sometidos.
  - **Movimiento:** Son aquellos que experimentan variaciones en función de los movimientos a los que son sometidos.
  - **Químicos:** Son aquellos que experimentan variaciones en función de los agentes químicos externos que pudieran incidir sobre ellos.

#### **2.2.1.4. Atendiendo a los elementos utilizados en su fabricación:**

- **Mecánicos:** Son aquellos que utilizan contactos mecánicos que se abren o cierran.
  - **Resistivos:** Son aquellos que utilizan en su fabricación elementos resistivos.
  - **Capacitivos:** Son aquellos que emplean condensadores.
  - **Inductivos:** Son aquellos que emplean bobinas.
  - **Piezoeléctricos:** Son aquellos que emplean cristales como el cuarzo.
  - **Semiconductores:** Son aquellos que emplean materiales Semiconductores.

**2.2.1.5. Atendiendo a la magnitud****Posición angular o lineal:**

- Potenciómetro
- Encoder

**Desplazamiento y deformación:**

- Gala extensiométrica
- Magnetostrictivos
- LVDT

**Velocidad lineal y angular:**

- Dinamo tacométrica
- Encoder
- Inclinómetros
- RVDT
- Giróscopio

**Aceleración:**

- Acelerómetro
- Fuerza y par (deformación)
- Galgas extensiométrica
- Triaxiales

**Presión:**

- Membranas
- Piezoeléctricos
- Manómetros digitales

**Caudal:**

- Turbina
- Magnético

**Temperatura:**

- Termopar

- RTD
- Termistor NTC
- Termistor PTC
- Bimetal

**Presencia:**

- Inductivos
- Capacitivos
- Ópticos

**Táctiles:**

- Matriz de contactos
- Piel artificial

**Proximidad:**

- Capacitivo
- Inductivo
- Fotoeléctrico

**Acustico:**

- Micrófono

**Sensor de acidez:**

- ISFET

**Luz:**

- Fotodiodo
- Fotoresistencia
- Fototransistor

**Captura de movimiento:**

- Sensor inercial

Algunas magnitudes pueden calcularse mediante la medición y cálculo de otras, por ejemplo, la velocidad de un móvil puede calcularse a partir de la integración numérica de su aceleración. La masa de un objeto puede conocerse mediante la fuerza gravitatoria que se ejerce sobre él en comparación con la fuerza gravitatoria ejercida sobre un objeto de masa conocida.

### 2.2.2. Atendiendo a los errores de medición

La mayoría de los sensores tienen una función de transferencia lineal. La sensibilidad se define entonces como la relación entre la señal de salida y la propiedad medida. Por ejemplo, si un sensor mide la temperatura y tiene una salida de voltaje, la sensibilidad es constante con las unidades [V / K]. La sensibilidad es la pendiente de la función de transferencia. La conversión de la salida eléctrica del sensor (por ejemplo V) a las unidades medidas (por ejemplo K) requiere dividir la salida eléctrica por la pendiente (o multiplicar por su recíproco). Además, con frecuencia se agrega o resta una compensación. Por ejemplo, se debe agregar -40 a la salida si la salida de 0 V corresponde a la entrada de -40 °C.

- Sensibles a la propiedad medida.
- Insensibles a cualquier otra propiedad que pueda encontrarse en su aplicación.
- No influyen en la propiedad medida.

Para que una señal de sensor analógico sea procesada o utilizada en un equipo digital, necesita convertirse en una señal digital, utilizando un convertidor de analógico a digital añadiendo un posible error a la salida.

### 2.2.3. Características de los sensores

El sensor ideal sería aquel en que la relación entre la magnitud de entrada y la magnitud de salida fuese proporcional y de respuesta instantánea e idéntica para todos los elementos de un mismo tipo.

Sin embargo, la respuesta real de los sensores nunca es del todo lineal, tiene un rango limitado de validez que suele estar afectada por perturbaciones del entorno exterior además de tener un cierto retardo en la respuesta.

Las características de los sensores se pueden agrupar en dos grandes bloques en cuanto a sus características se refiere:

1. **Características estáticas**, que describen la actuación del sensor en régimen permanente o con cambios muy lentos de la variable a medir.
2. **Características dinámicas**, que describen el comportamiento del sensor en régimen transitorio.

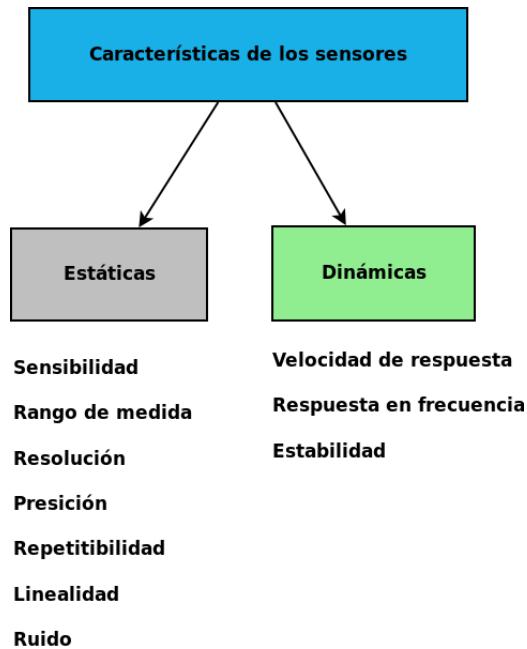


Figura 2.4: Esquema clasificatorio de las características estáticas y dinámicas de los sensores.

### 2.2.3.1. Estáticas

Instrumentos basados en la medición de parámetros estables, es decir, mediciones de valores que no presenten variaciones bruscas en su magnitud de salida.

#### Sensibilidad:

La sensibilidad de un sensor indica cuánto cambia la salida del sensor cuando cambia la cantidad de entrada que se mide. Por ejemplo, si el mercurio en un termómetro se mueve 1 cm cuando la temperatura cambia en 1 grado, la sensibilidad es 1 cm/C (es básicamente la pendiente Dy/Dx asumiendo una característica lineal).

#### Rango de medida:

El conjunto de valores que puede tomar la señal de entrada comprendidos entre el máximo y el mínimo detectados por el sensor con una tolerancia de error aceptable.

#### Resolución:

Indica la capacidad del sensor para discernir entre valores muy próximos de la variable de entrada. Indica que variación de la señal de entrada produce una variación detectable en la señal de salida.

#### Presición:

Define la variación máxima entre la salida real obtenida y la salida teórica dada como patrón para el sensor.

**Repetitibilidad:**

Indica la máxima variación entre valores de salida obtenidos al medir varias veces la misma entrada con el mismo sensor y en idénticas condiciones ambientales.

**Linealidad:**

Un transductor es lineal si existe una constante de proporcionalidad única que relaciona los incrementos de la señal de salida con los respectivos incrementos de la señal de entrada en todo el rango de medida.

**Ruido:**

Cualquier perturbación aleatoria del propio sistema de medida que afecta la señal que se quiere medir.

### 2.2.3.2. Dinámicas

Puede ocurrir que la cantidad bajo medición sufra una variación en un momento determinado y por lo tanto es necesario que conozcamos el comportamiento dinámico del instrumento cuando sucedan estas variaciones.

**Velocidad de respuesta:**

Mide la capacidad del sensor para que la señal de salida siga sin retraso las variaciones de la señal de entrada.

**Respuesta en frecuencia:**

Mide la capacidad del sensor para seguir las variaciones de la señal de entrada a medida que aumenta la frecuencia, generalmente los sensores convencionales presentan una respuesta del tipo paso bajo.

**Estabilidad:**

Indica la desviación en la salida del sensor con respecto al valor teórico dado, al variar parámetros exteriores distintos al que se quiere medir (condiciones ambientales, alimentación, etc.).

## 2.3. Transmisión y comunicación

Se denomina *transmisión* como el proceso de transporte de una señal de un lugar a otro y *comunicación* como el intercambio entre dos entes mediante una transmisión, los cuales son capaces de interpretar la información circundante entre ellos y en el cual existen un conjunto de reglas definidas, los protocolos<sup>1</sup>, que rigen el proceso.

## 2.4. Socket

*Socket*, o también conocido como conector, designa un concepto abstracto mediante el cual dos programas, generalmente situados en computadoras distintas, pueden intercambiar cualquier flujo de datos de manera fiable y ordenada.

La comunicación a través de una red de ordenadores es una tarea compleja en la que para resolverla se ha empleado un enfoque de diseño por capas, pudiéndose hablar por tanto de una arquitectura o pila de protocolos donde cada capa utiliza servicios (funciones) de la capa inferior y ofrece servicios a la capa superior.

El modelo de referencia para la comunicación de ordenadores es el denominado modelo de Interconexión de Sistemas Abiertos OSI ( Open Systems Interconnection ), el cual queda descrito en la sección [2.6](#) junto con la localización de los sockets en dicho modelo.

El término *socket* es también usado como el nombre de una interfaz de programación de aplicaciones (API) para la familia de protocolos de red TCP/IP<sup>2</sup>, provista usualmente por el sistema operativo.

Los sockets constituyen el mecanismo para la entrega de paquetes de datos provenientes de la tarjeta de red a los procesos o hilos apropiados. Un socket queda definido por un par de direcciones IP local y remota, un protocolo de transporte y un par de números de puerto local y remoto.

Cuando se habla de dirección y puerto local/remoto, se sobreentiende que nos referimos a dos procesos (cliente/servidor o nodo/nodo) ya que ambas direcciones IP y puerto pueden coincidir para el intercambio de información entre procesos dentro de una misma máquina y; además, la comunicación puede ser perfectamente bidireccional, asumiendo que el par que la inicia es el cliente y su contrapartida un servidor pero pudiendo ejercer de forma ambivalente ambas partes.

---

<sup>1</sup>Protocolo: reglamento o serie de instrucciones que se fijan por tradición o por convenio.

<sup>2</sup> TCP/IP es un conjunto de protocolos que permiten la comunicación entre los ordenadores pertenecientes a una red. La sigla TCP/IP significa Protocolo de control de transmisión/Protocolo de Internet. Proviene de los nombres de dos protocolos importantes incluidos en el conjunto TCP/IP, es decir, del protocolo TCP y del protocolo IP.

## 2.5. WebSocket

Comprendido previamente el concepto de *socket* descrito en el punto 2.4, definimos *Websocket* como una tecnología que proporciona un canal de comunicación bidireccional y full-duplex<sup>3</sup> utilizada por cualquier aplicación cliente/servidor.

La API de WebSocket está siendo normalizada por el W3C, mientras que el protocolo WebSocket ya fue normalizado por la IETF<sup>4</sup> como el RFC 6455.

Debido a que las conexiones TCP comunes sobre puertos diferentes al 80 son habitualmente bloqueadas por los administradores de redes, el uso de esta tecnología proporcionaría una solución a este tipo de limitaciones proveyendo una funcionalidad similar a la apertura de varias conexiones en distintos puertos, pero multiplexando diferentes servicios WebSocket sobre un único puerto TCP a costa de una pequeña sobrecarga del protocolo.

## 2.6. La arquitectura TCP/IP y el modelo OSI

En 1977 la Organización Internacional de Estandarización ( International Standards Organization , ISO) estableció un subcomité encargado de diseñar una arquitectura de comunicación. El resultado fue el modelo de referencia para la Interconexión de Sistemas Abiertos OSI ( Open Systems Interconnection ). Dicho modelo define una arquitectura de comunicación estructurada en siete niveles verticales. Dicho modelo es utilizado como base teórica para el desarrollo de la arquitectura TCP/IP, la cual está compuesta por una serie de capas o niveles en los que se encuentran los protocolos que implementan las funciones necesarias para la comunicación entre dos dispositivos en red.

Siendo, por tanto, el modelo OSI el empleado en el estudio de las redes de datos y el modelo o arquitectura TCP/IP como un modelo real empleado es las redes actuales.

La Figura 2.5 representa los distintos niveles o capas de los modelos OSI y TCP/IP y la disposición de los sockets dentro de ellos.

---

<sup>3</sup>Full Duplex: definido como la capacidad de transmisión y recepción en ambas direcciones al mismo tiempo.

<sup>4</sup>Internet Engineering Task Force (IETF) (en español, Grupo de Trabajo de Ingeniería de Internet) es una organización internacional abierta de normalización, que tiene como objetivos el contribuir a la ingeniería de Internet, actuando en diversas áreas, como transporte, encaminamiento, seguridad. Se creó en los Estados Unidos, en 1986. Es mundialmente conocido porque se trata de la entidad que regula las propuestas y los estándares de Internet, conocidos como RFC.

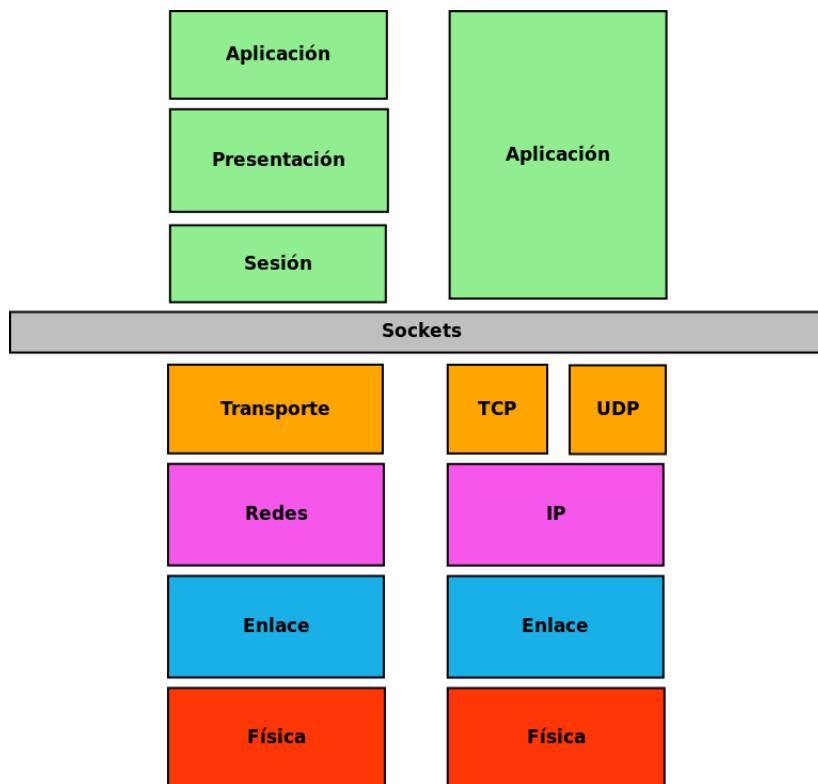


Figura 2.5: Representación de los modelos de capas o niveles OSI y TCP/IP.

## 2.7. Streaming

La retransmisión, streaming o también denominado transmisión, es la distribución digital de contenido multimedia a través de una red de computadoras, de manera que el usuario “consume” el producto medida que es descargado. La palabra retransmisión se refiere a una corriente continua la cual fluye sin interrupción, habitualmente audio o video.

Este tipo de tecnología funciona mediante un buffer de datos que va almacenando el flujo de descarga en la estación del usuario para inmediatamente mostrarle el material descargado. Esto se contrapone al mecanismo de descarga de archivos, que requiere que el usuario descargue los archivos por completo para poder acceder al contenido.

Como requisito fundamental, la retransmisión requiere de una conexión por lo menos de igual ancho de banda que la tasa de transmisión del servicio. Los servicios de streaming en Internet se popularizaron entre los años 2009 y 2010, cuando las compañías de telecomunicaciones pudieron ofrecer del suficiente ancho de banda para utilizar estos servicios en el hogar a un coste relativamente asequible.

## 2.8. Comunicación serie

La comunicación serie o comunicación secuencial, en telecomunicaciones e informática, es el proceso de envío de datos de un bit a la vez, de forma secuencial, sobre un canal de comunicación o un bus.

En cambio, en la “comunicación en paralelo” todos los bits de cada símbolo se envían al mismo tiempo, y por ello debe haber al menos tantas líneas de comunicación como bits tenga la información a transmitir.

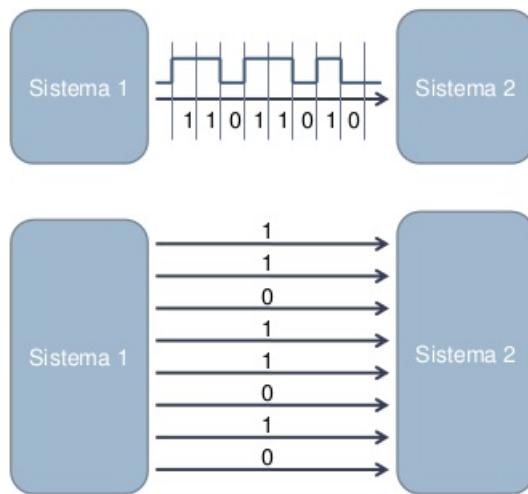


Figura 2.6: Diagrama representativo de la comunicación serie frente a la comunicación en paralelo.

### 2.8.1. Características

La ventaja de la comunicación serie es que necesita un número más pequeño de líneas de transmisión que una comunicación paralela que transmita la misma información. Esta última necesita tantas líneas de transmisión como la cantidad de bits que componen la información, mientras que la primera se puede llevar a cabo con una sola línea de transmisión. Por otra parte, surgen una serie de problemas en la transmisión de un gran número de bits en paralelo, como los problemas de interferencia o desincronización.

A la misma frecuencia de transmisión, la comunicación paralela tiene un mayor rendimiento. La comunicación serie tiene que compensar esta debilidad con una frecuencia más alta.

Ejemplos:

- Código Morse

- Ethernet
- Fibre Channel
- FireWire
- I<sup>2</sup>C
- MIDI
- PCI Express
- RS-232
- RS-485
- Serial ATA
- Serial Peripheral Interface
- Universal Serial Bus

## 2.9. PWM

La modulación de ancho de pulso (PWM) o la modulación de duración de pulso (PDM) es una técnica de modulación utilizada para codificar un mensaje en una señal de pulso. Aunque esta técnica de modulación puede utilizarse para codificar información para la transmisión, su uso principal es permitir el control de la potencia suministrada a los dispositivos eléctricos, especialmente a cargas inerciales tales como motores [16].

A continuación se muestran algunos gráficos demostrando señales PWM con diferentes ciclos de trabajo:

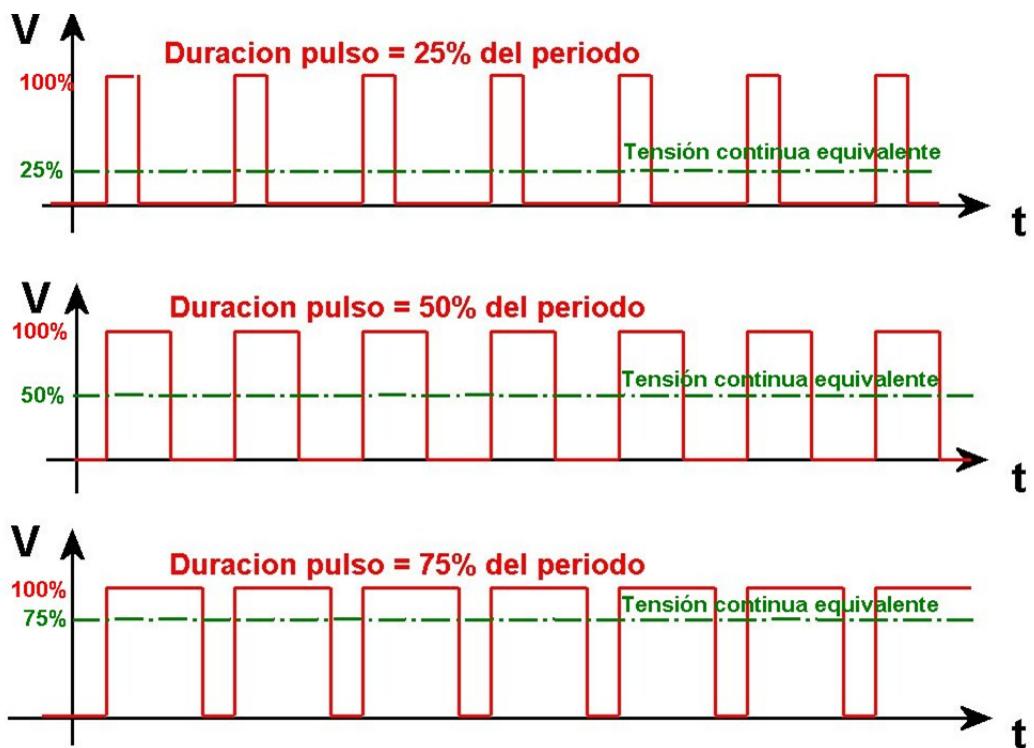


Figura 2.7: Representación de diferentes ciclos de trabajo de una señal PWM.

La principal ventaja de PWM es que la pérdida de potencia en los dispositivos de conmutación es muy baja. Cuando un interruptor está apagado prácticamente no hay corriente, y cuando está encendido y la energía se transfiere a la carga, casi no hay caída de tensión en el interruptor. La pérdida de potencia, al ser el producto de voltaje y corriente, es, por lo tanto, en ambos casos cercano a cero. PWM también funciona bien con controles digitales, que, debido a su naturaleza de encendido/apagado, pueden establecer fácilmente el ciclo de trabajo necesario.



# **Capítulo 3**

## **Estado del arte y herramientas utilizadas**

### **3.1. Estado del arte**

#### **3.1.1. Introducción**

Una de las mejores cosas de los robots es su capacidad para realizar trabajos que serían simplemente peligrosos para las personas. Los robots no solo pueden operar en entornos donde los humanos no pueden, sino que también pueden enfrentar desafíos que son peligrosos.

Los robots pueden explorar en las profundidades del océano, investigar volcanes, zonas radiactivas, etc. En muchos casos, sin embargo, simplemente se hacen cargo de trabajos que son tediosos y tienen un bajo margen de error como puede ser el caso de la soldadura, la cual es una parte muy importante de todo tipo de entornos de fabricación pesada.

En este tipo de trabajos, desafortunadamente, un instante de inatención puede significar un desastre para un soldador humano. No solo deben ser muy hábiles y precisos, sino que los errores pueden causar un gran desperdicio. Con los robots, se elimina el riesgo: el ruido, el calor intenso y los humos tóxicos no son un problema. Además, los robots pueden cambiar de un proyecto a otro más rápido que las personas.

En el caso de la exploración de aguas profundas, incluso los buzos con equipo moderno y años de experiencia pueden sucumbir a diversos peligros, incluidos los depredadores del mar y los efectos de las elevadas presiones. En la actualidad existen robots de exploración submarina capaces explorar y realizar tareas a elevadas profundidades sin el peligro humano que estas acciones conllevarían.

Los robots han sido probados para una amplia gama de aplicaciones de respuesta a emergencias, incluyendo la extinción de incendios forestales, inundaciones y otras catástrofes naturales.

Con el tiempo, los robots se especializarán y adaptarán cada vez mejor a situaciones que serían potencialmente letales para los humanos. Con una capacidad de maniobra y adaptación, de lo que podemos concluir que los robots se han transformado en el compañero perfecto para cualquier tipo de misión de exploración o de rescate de alto riesgo.

Estas situaciones anteriormente descritas combinadas a que en la actualidad la robótica cada vez se encuentra más presente en nuestras vidas cotidianas hacen que cualquier usuario pueda elaborar sus propios proyectos robóticos adaptables a cualquier entorno. Prueba de ello es que si realizamos cualquier búsqueda en internet podemos encontrar multitud de proyectos robóticos en la red. Existen excelentes portales como <http://blog.bricogeek.com/> por nombrar alguno de ellos, donde se publican guías paso a paso de proyectos novedosos y originales para que cualquier usuario pueda hacerlos realidad.

Teniendo en cuenta éste contexto definido en actualidad, en la sección 3.1.2 se citan algunas de las referencias consultadas con la finalidad de contextualizar el proyecto en un ámbito más científico.

Por todo lo anterior nace SensorRS, un robot diseñado para operar en zonas peligrosas, el cual ha sido construido con medios asequibles para cualquier usuario pueda hacerlo realidad y personalizable y adaptable a situaciones concretas.

### 3.1.2. Referencias

Tras la realización de un sondeo entre numerosas bases de datos de artículos científicos existentes, indicar la existencia de artículos cuya temática guardan similitud con la problemática presentada en cuanto a los requisitos y características del presente proyecto. Dichos artículos han sido tomados como punto de partida para el abordaje del problema y su estudio.

Existen numerosos artículos en los que se aborda el diseño y desarrollo y teleoperación de sistemas robóticos mediante la utilización de WebSockets y mediante la programación de microcontroladores. Por citar algunos de ellos como; *Design and Development of a Robotic Teleoperation System using Duplex WebSockets suitable for Variable Bandwidth Networks* [27], en el que se describe el desarrollo de un sistema teleoperable por WebSockets destacando entre otros aspectos sus ventajas como un menor consumo del ancho de banda.

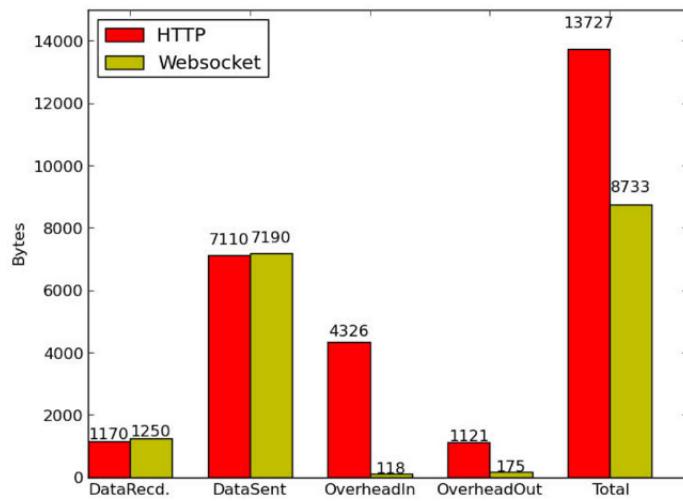


Figura 3.1: Comparativa del consumo de ancho de banda entre HTTP y WebSockets.

Otros estudios, en cambio, se centran más en analizar las diferentes situaciones de sobrecarga en la red, anchos de banda, cargas del sistema, etcétera, con la finalidad de someter a pruebas de estrés los diferentes protocolos implicados. Como por ejemplo: *Analysis of WebSockets as the New Age Protocol for Remote Robot Tele-operation* [26].

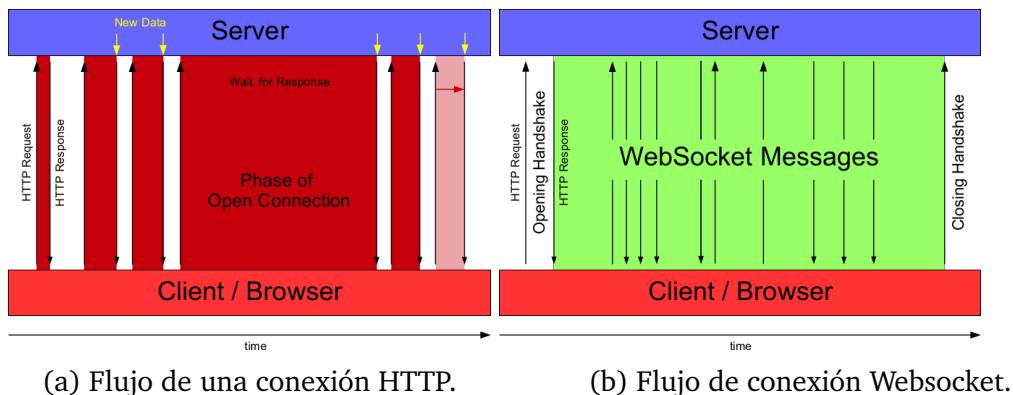


Figura 3.2: Gráfico representativo de las etapas en una petición HTTP frente a una petición por Websocket [31].

Por otra parte, el artículo *Remote Monitoring System based on a Wi-Fi Controlled Car Using Raspberry Pi* [31], describe un sistema de monitorización construido sobre una placa Raspberry Pi de las mismas características que la empleada para el robot SensorRS.

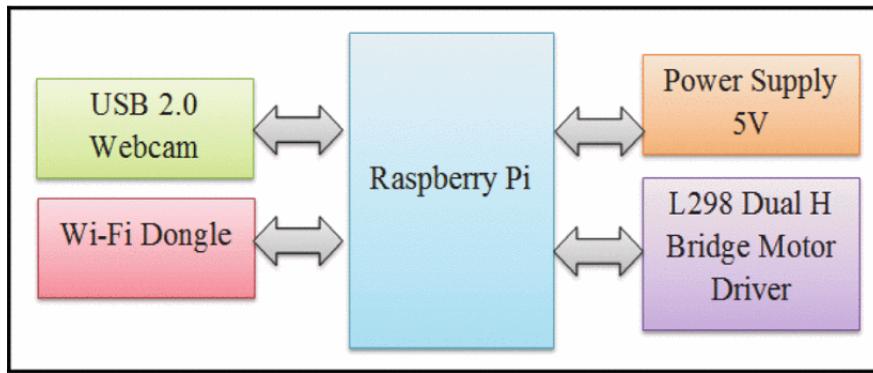


Figura 3.3: Diagrama de bloques del sistema referenciado en [31].

También existen numerosos artículos donde detallándose el procedimiento para la construcción de proyectos robóticos con Arduino, como por ejemplo *Design and Control of a Two-Wheel Self-Balancing Robot usin g the Arduino Microcontroller Board* [?] o proyectos que combinan el uso de una placa Raspberry Pi con una Arduino, como puede ser *A Robotic-agent Platform For Embedding Software Agents usin Raspberry Pi and Arduino Boards* [14] el cual emplea un sistema de formato de mensajes para la comunicación entre Arduino y Raspberry Pi muy similar al empleado en el presente trabajo.

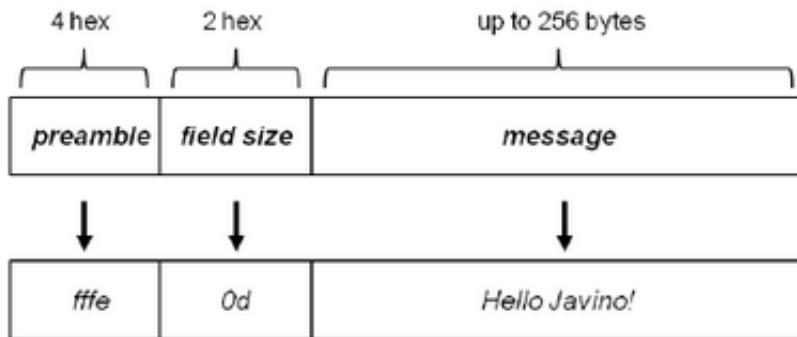


Figura 3.4: Formato de mensaje para la comunicación entre Arduino y Raspberry Pi [14].

Es, por todo lo anterior, que en el presente proyecto se ha intentado dotar al sistema de una característica añadida. Permitir el seguimiento por parte parte del usuario que realiza las funciones de teleoperador. Además de enfocarlo desde la perspectiva del entretenimiento y la enseñanza.

## 3.2. Tecnologías software utilizadas

A continuación se detallan las diferentes tecnologías/bibliotecas/lenguajes que se han empleado para la elaboración del proyecto y por qué se han escogido por encima de otras posibles soluciones.

### 3.2.1. L<sup>A</sup>T<sub>E</sub>X

Web: <https://www.latex-project.org/>

L<sup>A</sup>T<sub>E</sub>X es un lenguaje de marcado que sirve para la redacción de documentos científicos o técnicos. Con esta herramienta o lenguaje se ha desarrollado la memoria actual del proyecto de final de carrera.

Para el estudio de la herramienta y realización de consultas se han empleado sobre los recursos bibliográficos [28] y [30].

### 3.2.2. WebStorm



Web: <https://www.jetbrains.com/webstorm/>

WebStorm es un IDE de JavaScript ligero pero potente, perfectamente equipado para el desarrollo del lado del cliente (vehículo robótico) con Node.js. Para el desarrollo de la aplicación se optó por este IDE.

### 3.2.3. Fritzing



Web: <http://fritzing.org/home/> [23]

Fritzing es una iniciativa de hardware de código abierto que hace que los productos electrónicos sean accesibles como material creativo para cualquier persona.

Permitiendo la realización de diagramas, circuitos electrónicos y montajes, también sirve para hacer circuitos impresos PCB de manera profesional.

### 3.2.4. Arduino IDE



Web: [\[23\]](https://http://arduino.cc/)

El entorno de desarrollo ofrecido por Arduino resulta extremadamente sencillo de utilizar y de gran simpleza ya que el programa se escribe en la zona del sketch. Para compilarlo y cargarlo a cualquier placa tan solo es necesario pulsar un botón.

### 3.2.5. Github



Web: [\[19\]](https://about.github.com/)

Repositorio: <https://github.com/lopi87/SensorRS>

GitHub es una forja (plataforma de desarrollo colaborativo) para alojar proyectos utilizando el sistema de control de versiones Git. Utiliza el framework Ruby on Rails por GitHub, Inc. (anteriormente conocida como Logical Awesome). Desde enero de 2010, GitHub opera bajo el nombre de GitHub, Inc. El código se almacena de forma pública, aunque también se puede hacer de forma privada, creando una cuenta de pago.

### 3.2.6. Git



Web: [\[11\]](https://git-scm.com/)

Git es un sistema open-source de control de versiones diseñado para manejar íntegramente las fases de desarrollo de proyectos, simples y complejos, con velocidad y eficiencia.

Para su estudio y afianzado de conocimientos, ya que este sistema de control de versiones me resultaba muy familiar incluso antes de comenzar con el desarrollo del presente proyecto, se ha empleado la referencia bibliográfica [11]. La cual hace un recorrido por

todas las posibilidades que brinda esta herramienta.

### 3.2.7. Amazon Web Services (AWS)



Web: <https://aws.amazon.com/es>

Amazon Web Services (AWS) es una plataforma de servicios de nube que ofrece potencia de cómputo, almacenamiento de bases de datos, entrega de contenido y otra funcionalidad para ayudar a las empresas a escalar y crecer. Explore cómo millones de clientes aprovechan los productos y soluciones de la nube de AWS para crear aplicaciones sofisticadas y cada vez más flexibles, escalables y fiables.

Destacar la importancia de la referencia bibliográfica [4] para el conocimiento de los aspectos básicos a la hora de gestionar los servidores virtuales proporcionados por AWS.

### 3.2.8. Node.js



Web: <https://nodejs.org/es/>  
documentación: <https://nodejs.org/es/docs/> [10]

Node.js es un entorno de ejecución para JavaScript construido con el motor de JavaScript V8 de Chrome. Node.js usa un modelo de operaciones E/S sin bloqueo y orientado a eventos, que lo hace liviano y eficiente. Incorpora un sistema de gestión de paquetes

llamado, npm, es el ecosistema mas grande de librerías de código abierto en el mundo.

Node.js tiene una arquitectura basada en eventos capaz de E/S asíncronos. Estas opciones de diseño apuntan a optimizar el rendimiento y la escalabilidad en aplicaciones Web con muchas operaciones de entrada/salida, así como para aplicaciones Web en tiempo real (por ejemplo, programas de comunicación en tiempo real y juegos de navegador), lo que lo hacen ideal para este proyecto.

Destacar la importancia de la referencia bibliográfica [6] para el estudio de tanto Node.js como JavaScript y que han servido de guía y consulta en la elaboración del presente proyecto. Sin olvidar la documentación oficial de Node.js [10].

### 3.2.9. Npm



Web: <https://www.npmjs.com/>

Npm es el gestor de paquetes por defecto para Node.js, un entorno de ejecución para JavaScript. Utilizado para la descarga de las librerías incorporadas al proyecto.

### 3.2.10. SocketIO



Web: <https://socket.io/>

Socket.IO es una biblioteca de JavaScript para aplicaciones web en tiempo real. Permite la comunicación bidireccional en tiempo real entre clientes web y servidores. Consta de dos partes: una biblioteca del lado del cliente que se ejecuta en el navegador y una biblioteca del lado del servidor para Node.js. Ambos componentes tienen una API casi idéntica. Al igual que Node.js, es impulsado por eventos.

Socket.IO puede usarse simplemente como un wrapper para WebSocket aunque proporciona muchas más funciones, incluyendo la transmisión a múltiples sockets, almacenamiento de datos asociados a cada cliente y E/S asíncronas.

### 3.2.11. FFmpeg



Web: <https://ffmpeg.org/>

FFmpeg es una colección bibliotecas software libre que permiten grabar, convertir (transcodificar) y hacer streaming de audio y vídeo. Incluye libavcodec, una biblioteca de codecs. FFmpeg está desarrollado en GNU/Linux, pero puede ser compilado en la mayoría de los sistemas operativos, incluyendo Windows.

FFmpeg es un programa bastante sencillo y de fácil utilización, orientado tanto a personas con conocimientos avanzados como usuarios inexpertos.

El proyecto FFmpeg está compuesto por:

- **ffmpeg:** es una herramienta de línea de comandos para convertir audio o video de un formato a otro. También puede capturar y codificar en tiempo real desde DirectShow, una tarjeta de televisión u otro dispositivo compatible.
- **ffserver:** es un servidor de streaming multimedia de emisiones en directo que soporta HTTP (la compatibilidad con RTSP está en desarrollo). Todavía no está en fase estable, y de momento no está disponible para Windows.
- **ffplay:** es un reproductor multimedia basado en SDL y las bibliotecas FFmpeg.
- **libavcodec:** es una biblioteca que contiene todos los codecs de FFmpeg. Muchos de ellos fueron desarrollados desde cero para asegurar una mayor eficiencia y un código altamente reutilizable.
- **libavformat:** es una biblioteca que contiene los multiplexadores/demultiplexadores para los archivos contenedores multimedia.
- **libavutil:** es una biblioteca de apoyo que contiene todas las rutinas comunes en las diferentes partes de FFmpeg.
- **libpostproc:** es una biblioteca de funciones de postproceso de vídeo.
- **libswscale:** es la biblioteca de escalado de vídeo.

Para el desarrollo de SensorRS, concretamente para la transmisión de vídeo desde el robot desarrollado hacia el servidor (aplicación web), el módulo utilizado ha sido el de la herramienta de línea de comandos.

### 3.2.12. Bootstrap



Web: <http://getbootstrap.com/>

Bootstrap es un framework o conjunto de herramientas de código abierto para diseño de sitios y aplicaciones web. Contiene plantillas de diseño con tipografía, formularios, botones, cuadros, menús de navegación y otros elementos de diseño basado en HTML y CSS, así como, extensiones de JavaScript opcionales adicionales. Se ha utilizado en el presente proyecto para la maquetación de la aplicación. En el presente proyecto se ha aprovechado para actualizarlo a la versión 4.1.3, la más reciente hasta la fecha.

### 3.2.13. JQuery



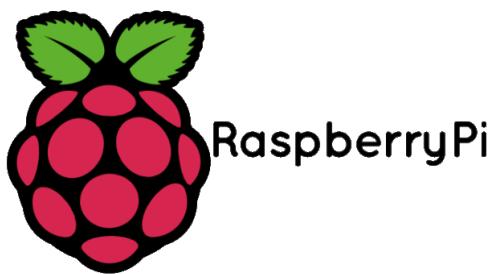
Web: <https://jquery.com/>

JQuery es una biblioteca de JavaScript rápida, pequeña y característica. Hace que las cosas como manipulación del código HTML, manejo de eventos, animación, y permite la realización de peticiones Ajax de manera mucho más simple gracias a API de fácil manejo, la cual funciona a través de una multitud de navegadores. Gracias a su combinación de versatilidad y extensibilidad JQuery ha cambiado la forma en que millones de personas desarrollan con JavaScript.

## 3.3. Tecnologías hardware y materiales utilizados

A continuación se detallan las diferentes tecnologías hardware que se han empleado para la elaboración del vehículo robótico con la finalidad de servir de prototipo durante el desarrollo de la aplicación y a modo demostrativo de la misma. En los sucesivos puntos se describen las características de cada una de ellas junto con el motivo de su elección.

### 3.3.1. Raspberry Pi Model B



Web: <https://www.raspberrypi.org>

Raspberry Pi es un ordenador de tamaño reducido y de bajo coste desarrollado en Reino Unido por la Fundación Raspberry Pi, con el objetivo de estimular la enseñanza de ciencias de la computación en las escuelas. Es ampliamente utilizado y de uso muy extendido por lo que ha sido el principal motivo de su elección, además de su bajo cote y versatilidad.

La Raspberry Pi 3 es la tercera generación de Raspberry Pi. Sus especificaciones son las siguientes:

- Una CPU ARMv8 quad-core de 64 bits de 64 bits y 1.2 GHz.
- LAN inalámbrica 802.11n.
- Bluetooth 4.1.
- Bluetooth baja energía (BLE).
- 1 GB de RAM.
- 4 puertos USB.
- 40 conexiones GPIO.
- Puerto HDMI.
- Puerto Ethernet.
- Conector de audio combinado de 3,5 mm y vídeo compuesto.
- Interfaz de la cámara (CSI).
- Interfaz de pantalla (DSI).
- Ranura para tarjeta Micro SD.
- VideoCore IV núcleo de gráficos 3D.



Figura 3.6: Imagen de una Raspberry Pi 3 Model B

### 3.3.2. Arduino



Web: <https://www.arduino.cc/>

Arduino es una compañía open source y open hardware, así como un proyecto y comunidad internacional que diseña y manufactura placas de desarrollo de hardware para construir dispositivos digitales y dispositivos interactivos que puedan sensar y controlar objetos del mundo real. Arduino se enfoca en acercar y facilitar el uso de la electrónica y programación de sistemas embebidos en proyectos multidisciplinarios. Los productos que vende la compañía son distribuidos como Hardware y Software Libre, bajo la Licencia Pública General Reducida de GNU (LGPL) o la Licencia Pública General de GNU (GPL), permitiendo la manufactura de las placas Arduino y distribución del software por cualquier individuo. Las placas Arduino están disponibles comercialmente en forma de placas ensambladas o también en forma de kits hazlo tu mismo (DIY, por sus siglas en inglés de "Do It Yourself").

Los diseños de las placas Arduino usan diversos microcontroladores y microprocesadores. Generalmente el hardware consiste de un microcontrolador Atmel AVR, conectado bajo la configuración de "sistema mínimo" sobre una placa de circuito impreso a la que se le pueden conectar placas de expansión (shields) a través de la disposición de los puertos de entrada y salida presentes en la placa seleccionada. Las shields complementan la funcionalidad del modelo de placa empleada, agregando circuitería, sensores y módulos de comunicación externos a la placa original. La mayoría de las placas Arduino pueden ser energizadas por un puerto USB o un puerto barrel Jack de 2.5mm. La mayoría de las placas Arduino pueden ser programadas a través del puerto Serial que

incorporan haciendo uso del Bootloader que traen programado por defecto. El software de Arduino consiste de dos elementos: un entorno de desarrollo (IDE) (basado en el entorno de processing y en la estructura del lenguaje de programación Wiring), y en el cargador de arranque (bootloader, por su traducción al inglés) que es ejecutado de forma automática dentro del microcontrolador en cuanto este se enciende. Las placas Arduino se programan mediante un computador, usando comunicación serial.

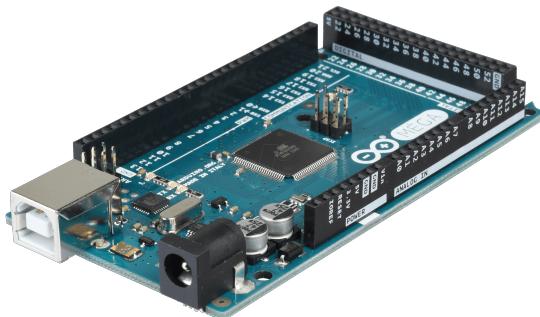


Figura 3.7: Placa Arduino MEGA 2560

Para el presente trabajo se ha utilizado la placa Arduino Mega, la cual es una tarjeta de desarrollo open-source construida con un microcontrolador modelo Atmega2560 que posee pines de entradas y salidas (E/S), analógicas y digitales. Esta tarjeta es programada en un entorno de desarrollo que implementa el lenguaje Processing/Wiring. Arduino puede utilizarse en el desarrollo de objetos interactivos autónomos o puede comunicarse a un PC a través del puerto serial (conversión con USB) utilizando lenguajes como Flash, Processing, MaxMSP, etc. Las posibilidades de realizar desarrollos basados en Arduino tienen como límite la imaginación.

El Arduino Mega tiene 54 pines de entradas/salidas digitales (14 de las cuales pueden ser utilizadas como salidas PWM), 16 entradas analógicas, 4 UARTs (puertos serial por hardware), cristal oscilador de 16MHz, conexión USB, jack de alimentación, conector ICSP y botón de reset. Arduino Mega incorpora todo lo necesario para que el microcontrolador trabaje; simplemente conéctalo a tu PC por medio de un cable USB o con una fuente de alimentación externa (9 hasta 12VDC). El Arduino Mega es compatible con la mayoría de los shields diseñados para Arduino Duemilanove, diecimila o UNO.

Microcontroller	ATmega2560
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	54 (of which 15 provide PWM output)
Analog Input Pins	16
DC Current per I/O Pin	20 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
LED_BUILTIN	13
Length	101.52 mm
Width	53.3 mm
Weight	37 g

Figura 3.8: Características Arduino MEGA 2560

### 3.3.3. Arduino Sensor Kit

Se ha utilizado un pack de diversos sensores para arduino, los sensores utilizados quedarán descritos a mayor detalle en el capítulo 5 referente a la construcción del robot y sensores utilizados.

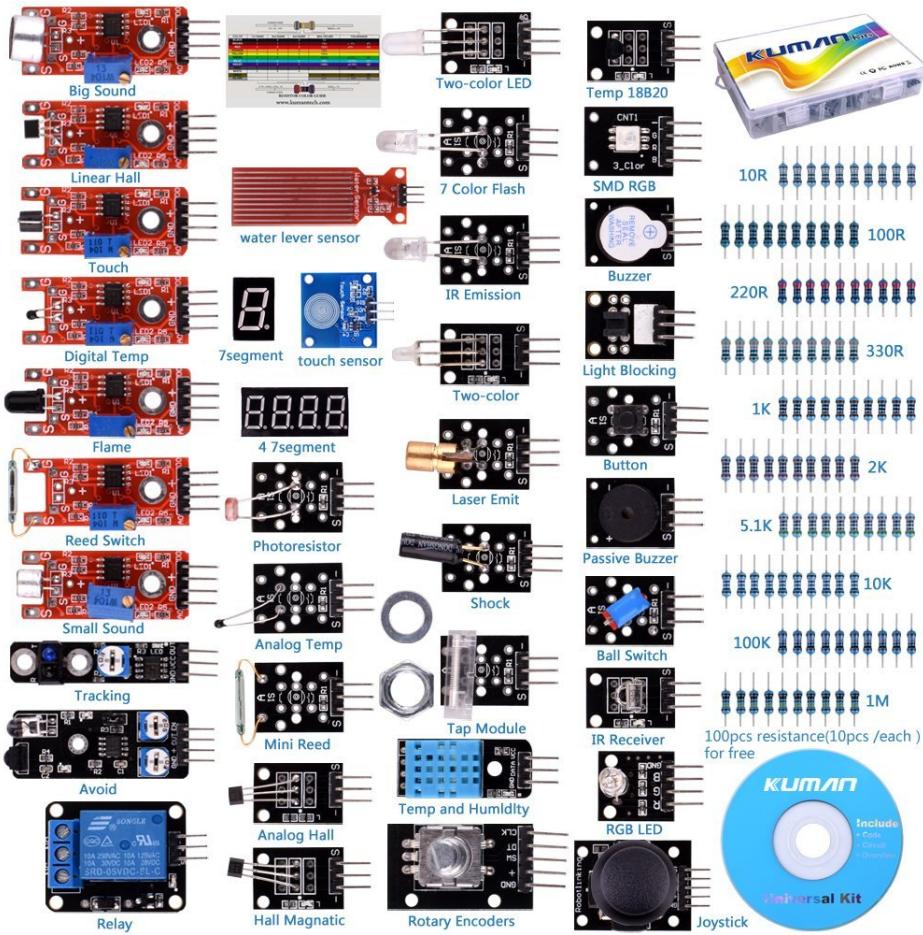


Figura 3.9: Pack de sensores para Arduino

### 3.3.4. Controlador de motores doble puente H - L298N

El módulo controlador de motores L298N H-bridge o puentes H, nos permite controlar la velocidad y la dirección de dos motores de corriente continua o un motor paso a paso de una forma muy sencilla, gracias a los 2 los dos H-bridge que dispone.

Básicamente un puente-H o H-bridge es un componente formado por 4 transistores que nos permite invertir el sentido de la corriente, y de esta forma podemos invertir el sentido de giro del motor <sup>1</sup>.

El rango de tensiones en el que trabaja este módulo va desde 3V hasta 35V, y una intensidad de hasta 2A. A la hora de alimentarlo hay que tener en cuenta que la electrónica del módulo consume unos 3V, así que los motores reciben 3V menos que la tensión con la que alimentemos el módulo.

<sup>1</sup>Podrá acceder a información más detallada de la presente controladora en la sección correspondiente al capítulo 5.1.4

Además el L298N incluye un regulador de tensión que nos permite obtener del módulo una tensión de 5V, perfecta para alimentar nuestro Arduino. Eso sí, este regulador sólo funciona si alimentamos el módulo con una tensión máxima de 12V.

Es un módulo que se utiliza mucho en proyectos de robótica, por su facilidad de uso y su reducido precio, lo cual ha servido para que sea seleccionado para el presente trabajo.

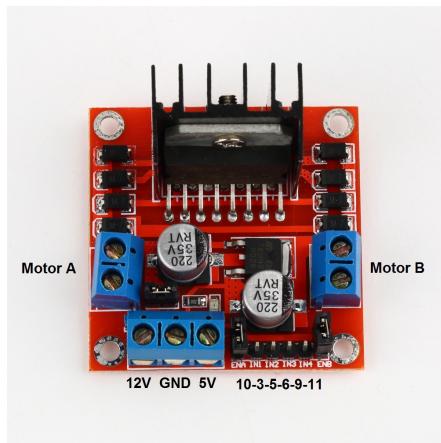


Figura 3.10: Imagen de la controladora de motores doble puente H - L298N utilizada.

### 3.3.5. Batería LiPo

Para alimentar los motores y su controladora se ha empleado una batería LiPo de 2200mAh a 11,1V. Las baterías de polímero de iones de litio, son pilas recargables (células de secundaria), compuestas generalmente de varias células secundarias idénticas en paralelo para aumentar la capacidad de la corriente de descarga.

Estas baterías son un tipo de batería recargable muy habitual en el mundo del modelismo, robótica, etc. Nacen como una opción aceptable frente a la utilización de combustibles siendo muy recomendables ya que ofrecen una prestaciones superiores a las NiCd y NiHMM.



Figura 3.11: Imagen de la batería LiPo utilizada.

### 3.3.6. Alimentación USB/Protoboard

Para alimentar la protoboard sin tener que modificar un cable USB, se ha decidido utilizar una pequeña tarjeta que se conecta en los carriles de tensión de la tarjeta protoboard, y proporciona los 5 V de entrada del USB (o de un conector Jack) a las líneas de alimentación de la protoboard. Además este dispositivo integra un interruptor, que permitirá activar y desactivar los motores, y puede regular la tensión a 3,3 Voltios.

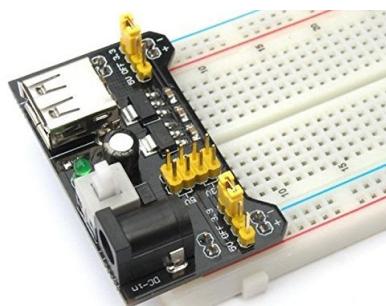


Figura 3.12: Adaptador de entrada USB a protoboard utilizado.

### 3.3.7. Tarjeta de expansión con batería de Litio para Raspberry Pi

Para la alimentación de la placa se ha optado por un módulo de potencia diseñado especialmente para la Raspberry Pi 3 Model B, permitiendo que la placa maestra trabaje sin conexión hasta 9 horas de forma ininterrumpida.

Por otra parte, esta placa dispone de 2 puertos USB adicionales: uno suministra energía para la Raspberry Pi y el otro para una posible pantalla LCD, resultando interesante

para otros proyectos.

Sus características principales son las siguientes:

1. Capacidad de la batería: 3800mAH.
2. Corriente de descarga máxima: 1.8A.
3. Tensión de salida sin carga:  $5.1V \pm 0.1V$ .
4. Corriente / voltaje de carga estándar: 1.0A / 5.0V.
5. Tensión de corte de la carga completa de la batería de iones de litio: 4.18V - 4.2V.

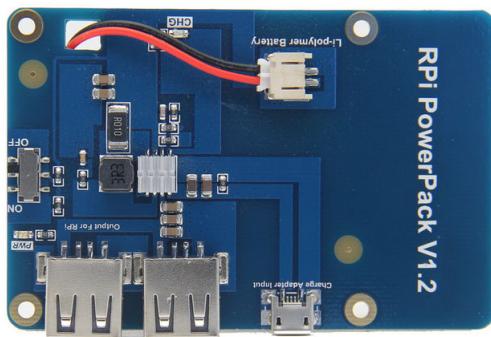


Figura 3.13: Imagen de la tarjeta de expansión con batería de Litio utilizado.

### 3.3.8. Cámara USB de alta definición

Cámara USB para su conexión en la Raspberry para la emisión de imágenes. La cámara seleccionada dispone de las siguientes características:

- 2 megapíxeles de resolución.
- Ángulo de visión de 170 grados.
- Interfaz USB 2.0 de alta velocidad, refresco de 60 fps en resolución 1280X720, 30 fps en resolución 1920X1080.
- Tamaño reducido y perfil delgado ideal para aplicaciones embebidas.



Figura 3.14: Imagen de la cámara USB utilizada.

Los motivos de su elección ha sido principalmente su facilidad de puesta en funcionamiento ya que al tratarse de una cámara USB. Tan sólo debemos conectarla para comenzar con su funcionamiento ya que es detectada por prácticamente todas las distribuciones Linux.

### 3.3.9. Gamepad

Gamepad inalámbrico con conexión Bluetooth para el manejo del vehículo desde un dispositivo móvil gracias al sistema de acoplamiento del que dispone, Ver figura 3.15.



Figura 3.15: Vista del gamepad utilizado.



# **Capítulo 4**

## **Especificación y análisis de requisitos**

En el presente capítulo se recopilan las diferentes características base de obligado cumplimiento por el producto final a desarrollar. Siendo estas características definidas en una etapa inicial del presente proyecto.

Debido a que el proyecto implica un desarrollo tanto a nivel software como hardware se realizó una recopilación de requisitos abarcando ambas áreas por separado.

### **4.1. Requerimientos hardware**

Se ha optado por la construcción de un robot móvil dotado de un chasis de 4 ruedas donde las dos ruedas traseras serán accionadas por un motor mientras que las dos ruedas delanteras harán de directrices.

El motor propulsor funcionará a corriente continua de manera que en función de la polarización de los terminales haga girar las ruedas en una dirección o la contraria (marcha adelante o atrás), mientras que la dirección será accionada por un servomotor.

El chasis utilizado deberá permitir añadir multitud de componentes necesarios para construir el robot, deberá disponer de paneles para instalar las diferentes placas electrónicas y sensores además de ser lo suficientemente liguero para que el sistema en su conjunto tenga agilidad en sus movimientos a la vez que resistencia.

El robot además deberá poder obtener imágenes de vídeo y transmitirlas al servidor donde se encuentre alojada la aplicación de control. Por lo que resulta necesario incorporar una cámara de pequeñas dimensiones de alta definición.

Por otra parte, todo robot necesita de una unidad de central de procesamiento donde se localizará el programa de control. Este programa tendrá la función de interpretar las diferentes señales recibidas, control de sensores y dispositivos conectados. Además, esta placa es la encargada de distribuir la alimentación por los diferentes componentes hardware que lo necesiten y recibir las señales de los sensores y enviarla a los motores.

Necesidad de poder acoplar multitud de sensores para medición de parámetros ambientales resultando necesario incorporar alguna placa dotada de un microcontrolador de fácil programación como podría ser algunas de las placas proporcionadas por Arduino.

Tanto la placa que incorpore el microcontrolador y sus sensores como la unidad central de procesamiento deben disponer de un canal de comunicación para el intercambio de datos entre las mismas.

#### 4.1.1. Análisis y selección de componentes electrónicos

La informática, en los últimos años ha dado un gran impulso con proyectos como el de la Raspberry Pi Foundation, en este caso concreto en áreas como la del IOT<sup>1</sup>. En lo referente al hardware, nunca ha sido más fácil coger componentes, juntarlos y con una mínima programación hacer algo totalmente nuevo, interesante y útil al mismo tiempo gracias a la existencia de proyectos como Arduino.

En este caso, nos planteamos la siguiente incógnita; ¿Empleamos una Raspberry Pi o un Arduino?. En internet encontramos tantos y tantos proyectos por hacer en los que muchas veces es difícil decidirse.

Arduino se compone de una parte Hardware y Software. Es por ello que gracias a los emuladores existentes y sin tocar una sola pieza hardware, podemos simular un proyecto desde nuestro ordenador. Podemos programarlo y hacer las conexiones virtuales para ver cómo se comportaría. Arduino es una plataforma simple y dedicada precisamente a eso, a montar sobre ella los componentes necesarios para los proyectos.

La Raspberry Pi Foundation en cambio, ha diseñado y elaborado un ordenador para enseñar informática a la antigua usanza. La Raspberry Pi es un ordenador asequible, suficientemente potente para facilitar el aprendizaje y realizar tareas básicas. Incluso programar y compilar programas que se ejecuten en la Raspberry Pi. Y todo ello en un tamaño mínimo, similar al de una tarjeta de crédito, alimentado con un cargador de móvil de 2 amperios y que da muchísimo juego para todo tipo de proyectos.

Por tanto, en respuesta a la pregunta inicial, se ha decidido que la mejor placa la elaboración del proyecto es aprovechar lo mejor de cada una de ellas. Utilizaremos tanto una Raspberry Pi como una Arduino Mega aprovechando al máximo el potencial que ofrece cada una de ellas, cada una con sus virtudes y sus defectos.

---

<sup>1</sup> Internet de las cosas (en inglés, Internet of Things, abreviado IoT o IdC, por sus siglas en español) es un concepto que se refiere a la interconexión digital de objetos cotidianos con Internet.

#### 4.1.1.1. Ventajas y desventajas de Raspberry Pi y de Arduino

Sabemos que Arduino no destaca precisamente por su potencia de cálculo, ni la memoria de la placa, ni la frecuencia del procesador. Entonces, ¿Por qué Arduino es masivamente utilizado en multitud de proyectos? Arduino destaca por su la facilidad de conectarse con el mundo, gracias a las entradas tanto analógicas como digitales con las que cuenta y de lo fácil que resulta activar o desactivar una de las entradas/salidas gracias a su software.

La placa Arduino Mega empleada en el presente trabajo dispone 54 pines de entrada salida digital, de los cuales quince de ellos pueden ser utilizados como salidas PWM y controlar con ellos la velocidad de motores, lectura de sensores, activación de servomotores, por citar algunos de los múltiples ejemplos y posibilidades. También tiene 16 entradas analógicas, una frecuencia de 16 MHz y un conector USB y un ICSP y 4 UART. Existen muchísimos tipos de placas Arduino y cada una con sus características específicas.

Otro punto a favor su la facilidad de prototipado donde los Shields o placas de expansión, ofrecen a nuestra placa funcionalidades añadidas que van desde conectividad Wi-Fi, GPS, conectividad por radio, pantallas, displays táctiles, etc. Y muchas de ellas con un bajo coste.

Por el contrario la Raspberry Pi puede presumir de una potencia de cálculo mucho más elevada comparada con las placas Arduino. Posee además mayor cantidad de memoria y capacidades multimedia, que van desde la reproducción de vídeo en HD, pasando por una salida de audio, así como una salida HDMI. Esto hacen que pese a no contar con las capacidades de interconexión de Arduino, sí que existan placas de expansión en forma de shields. Posible gracias a los conectores GPIO, I2S, etc. que incorpora la Raspberry Pi.



Figura 4.1: Esquema GPIO de una Raspberry Pi Model B+.

Por otra parte, otro de los puntos a tener muy en cuenta es la posibilidad de incorporar las diferentes cámaras desarrolladas de diversas características y tipos diferentes como

la captación de imágenes por infrarrojos, nos damos cuenta de que la Raspberry Pi puede sustituir a un ordenador en tareas simples. Y además podemos usar sus puertos de conexión para interconectar nuestro proyecto con el mundo como lo haríamos con un Arduino.

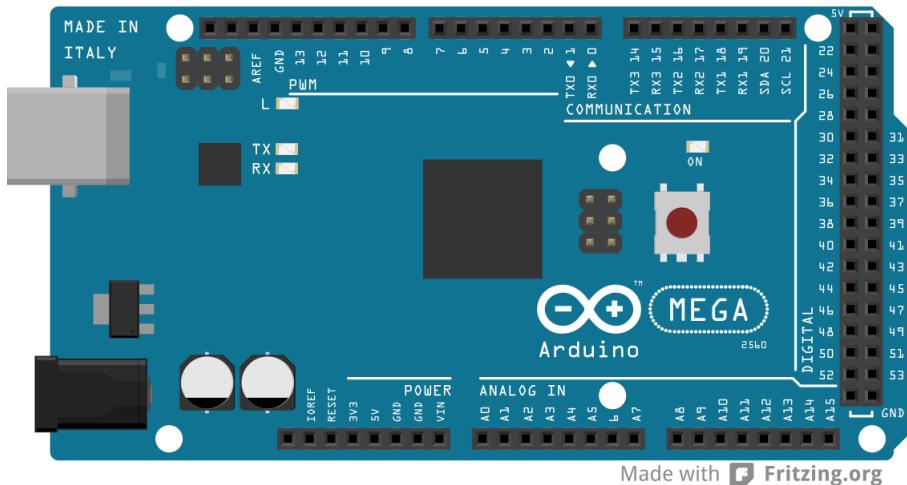


Figura 4.2: Placa Arduino Mega donde se visualiza la disposición de sus pines E/S.

Por tanto, de todo lo anterior podemos concluir Arduino y Raspberry Pi son herramientas complementarias y perfectamente utilizables para cumplir con los requisitos del presente proyecto.

## 4.2. Requerimientos software

Habiendo detallado los requerimientos hardware para la construcción del robot pasamos al análisis de los diferentes requerimientos software para la correcta gestión de los diferentes elementos hardware utilizados y para su correcto funcionamiento.

En cuanto a la programación del robot, uno de los requisitos fundamentales es el de disponer de una vía de comunicación bidireccional junto con la posibilidad de configurar una interfaz de control personalizada en función de los sensores que se deseen utilizar y según las características del medio al que queramos adaptar nuestro vehículo.

Otro requerimiento software es el de la posibilidad de integrar la lectura de valores obtenidos por sensores mostrando al usuario los parámetros obtenidos y envío de órdenes desde un servidor externo. Además de captación y transmisión de vídeo en tiempo real.

Todas estas especificaciones quedan resueltas mediante la utilización de la aplicación web RobotUI, la cual se ha decidido utilizar como parte software del presente proyecto.

Para el caso que nos concierne en el proyecto, dentro del marco de investigación que define la totalidad de la infraestructura, la funcionalidad principal del mismo a nivel software será:

- Definir los pasos para dar de alta un dispositivo robótico en el sistema.
- Una vez configurado el dispositivo, configurar la interfaz de control con las acciones de control específicas.
- Realizar un sistema de monitorización y visualización para los usuarios espectadores en tiempo real.
- Sistema de gestión de base de datos en donde se encuentren los datos de la aplicación recogidos.
- Panel de administración donde visualizar la información de los usuarios conectados y dispositivos en uso en tiempo real.

### 4.3. Especificación de los requisitos

En esta etapa del modelado de requisitos se captura el propósito general del sistema:

- Se analiza qué debe hacer el sistema.
- Se obtiene una versión contextualizada del sistema.
- Identifica y delimita el sistema.
- Se determinan las características, cualidades y restricciones que debe satisfacer el sistema.

#### 4.3.1. Requisitos funcionales

Los requisitos funcionales que se han obtenido en el sistema son los siguientes:

- Ser una herramienta multiplataforma y que permita a cualquier usuario definir sus propias interfaces para el control de robots.
- Dotar de funcionalidad gráfica que permita en tiempo real con mecanismos visuales (en web) visualizar el control de dispositivos robóticos por parte de otros usuarios, modo espectador de la aplicación.
- Proporcionar un sistema de streaming de vídeo para la difusión de imágenes a los usuarios espectadores procedentes de los robots dispongan de cámara.
- Implementar un panel de administración para la visualización de usuarios y dispositivos conectados en tiempo real.

### **4.3.2. Requisitos no funcionales**

Los requisitos no funcionales son aquellos que describen cualidades o restricciones del sistema que no se relacionan de forma directa con el comportamiento funcional del mismo. A continuación se especifican los más importantes del sistema:

- No requiere un conocimiento específico del sistema una vez puesto en funcionamiento.
- La aplicación tendrá manual de uso.
- La interfaz debe reflejar claramente la distinción entre las distintas partes del sistema.
- El sistema se desplegará sobre una versión GNU Linux Debian 8 Jessie.
- El código fuente de la aplicación seguirá un estilo uniforme y normalizado para todos los módulos del mismo.
- El sistema deberá ser fácilmente escalable permitiendo la incorporación de nuevos sensores.

# **Capítulo 5**

## **Construcción del robot**

En el presente capítulo se recogen todos los procedimientos para la construcción del vehículo robótico, desde el montaje a la interconexión de cada uno de sus elementos.

### **5.1. Montaje**

En esta sección se recogen todas las descripciones y procedimientos seguidos y que han resultado de mayor interés a la hora de la construcción del robot y sus diferentes interconexiones.

#### **5.1.1. Chasis**

El chasis consiste en una estructura interna que sostiene, aporta rigidez y da forma a un vehículo. Es análogo al esqueleto de un animal. Para el caso que nos compete, el vehículo desarrollado consta de un armazón que sirve de sujeción de los componentes mecánicos, como el sistema de propulsión y suspensión de las ruedas, incluyendo la carrocería además de los diferentes componentes electrónicos. Este robot debe ser capaz de acceder a y desenvolverse por zonas de difícil acceso, por ello de que debe disponer de un tamaño compacto que facilite el desplazamiento, lo mantenga en equilibrio en todo momento y que le permita cambiar de dirección fácilmente.

#### **5.1.2. Tracción y dirección**

Para generar el movimiento, se necesita algún dispositivo que proporcione una fuerza motriz encargada de desplazar el chasis y dotar de la capacidad de movimiento al robot. Se ha optado por la incorporación de ruedas para desplazarse, debido a su mayor agilidad y fácil manejo.

Estos equipos suelen utilizar baterías para alimentar los motores y electrónica, y por tanto, la fuente de alimentación suele ser corriente continua. Utilizando este tipo de

fuente de alimentación, el tipo de dispositivo a utilizar suele variar respecto de las necesidades de cada equipo, en la sección 5.2 se encuentra información referente a las mismas.

### 5.1.2.1. Motores de corriente continua

El motor de corriente continua es un dispositivo eléctrico que transforma la energía eléctrica en energía mecánica, de manera que genera un movimiento rotatorio gracias a la acción producida por el campo magnético. Este tipo de motores son también denominados como motores de corriente directa, motor CC o motor DC.

En caso de nuestro vehículo dispondrá de un motor para proporcionar movimiento a las ruedas traseras y otro de menor potencia para dotar de movimiento lateral a las delanteras a modo de dirección.

### 5.1.2.2. Servomotores

Los servomotores son dispositivos capaces de llevar el motor a posiciones angulares específicas al enviar una señal codificada manteniendo la posición angular del engranaje mientras la señal persista. Si esta señal cambia, la posición cambia, y si desaparece, el motor deja de mantener la posición. Dentro de un servomotor hay un motor de corriente continua, una caja reductora, un potenciómetro y una electrónica de control.



Figura 5.1: Imagen de un servomotor.

Un servomotor se compone de un motor, un circuito de control, un potenciómetro y un conjunto de engranajes. Los cuales se caracterizan por un consumo energético muy reducido.

Podemos definir un servomotor como un motor al que se le ha añadido un sistema de control, un potenciómetro y un conjunto de engranajes. Con anterioridad los servomotores no permitían que el motor girara 360 grados, solo aproximadamente 180; sin

embargo, hoy en día existen servomotores en los que puede ser controlada su posición y velocidad en los 360 grados. Los servomotores son comúnmente usados en modelismo para controlar y dotar de movimiento multitud de elementos.

En cuanto al conexionado, los servomotores poseen tres cables de conexión externa, alimentación, tierra y control. Mediante éste último indicamos el ángulo al cuál queremos llegar. Un servomotor normal tiene un movimiento angular de 0 a 180°, caso del utilizado en este proyecto. El ángulo está determinado por la duración de un pulso. El servomotor espera ver un pulso cada 20 milisegundos. La longitud del pulso determinará el giro que ha de realizar el motor. Un pulso de 1.5 ms hará que el motor se posicione en la posición neutra, a 90° respecto al eje. En caso contrario, si el pulso es menor de 1.5 ms, el motor se acercará a 0°, y si el pulso es mayor de 1.5ms, el eje se acercará a los 180 grados.

Inicialmente se pensó en utilizar un dispositivo de este tipo para el control de la dirección del vehículo pero debido a que el chasis utilizado ya incorporaba un motor de continua tradicional que cumplía muy bien su cometido, éste ha sido descartado. Dicho servomotor podría ser utilizado para dotar de movimiento de elevación, por ejemplo a la cámara que incorpora el vehículo tal y como se recoge en la sección 11.1 correspondiente a mejoras futuras.

### 5.1.3. Interconexión de elementos

En el presente punto se recogen todos aquellos puntos de interés referentes a la interconexión de elementos utilizados, los cuales quedaron descritos en la sección *Tecnologías hardware* 3.3 junto con sus especificaciones.

Comenzando por el módulo central del sistema, la placa Raspberry Pi Model B<sup>1</sup>, dispone de una serie de pines denominados GPIO (General Purpose Input/Output) es, como su propio nombre indica, un sistema de E/S (Entrada/Salida) de propósito general, es decir, una serie de conexiones que se pueden usar como entradas o salidas para usos múltiples. Estos pines se encuentran en todos los modelos de Raspberry Pi.

Comentar que debido a la incorporación de una placa Arduino no se han utilizado ninguno de los puertos GPIO que la Raspberry Pi ofrece a pesar de que hubiesen resultado perfectamente funcionales y encontrándose disponibles para cualquier ampliación que se deseé realizar con posterioridad. Se ha optado por dejar a la placa tan solo como unidad central de procesamiento, conexión del vehículo con una red de internet y transmisión de los datos capturados al servidor y escucha de comandos recibidos.

Pero existe una problemática y es que no podemos conectar directamente los pines de salida de nuestra placa Arduino directamente a los motores. Esto es debido a que la

---

<sup>1</sup> Todo lo referente a la utilización de los puertos de Entrada/Salida, puesta en funcionamiento y utilización de la placa Raspberry Pi se encuentra disponible en el manual de usuario [8] elaborado por uno sus creadores.

placa no dispone de potencia suficiente para mover actuadores. De hecho, la función de la placa no debe ser ejecutar acciones sino mandar ejecutar acciones a drivers que realicen el trabajo pesado.

### 5.1.4. Driver motores

Un driver motor es un dispositivo, o grupo de dispositivos, que se utilizan para controlar de una manera predeterminada el funcionamiento de un motor eléctrico. El control se consigue mediante unas señales de entrada, ya sean analógicas o bien digitales, con las que conseguimos:

- Seleccionar el sentido de giro del motor: gracias a la utilización de un driver, podemos elegir el sentido horario o anti horario del motor de manera muy simple.
- Regular la velocidad mediante el uso de una señal PWM, el driver permitirá regular la velocidad de variación de un motor DC
- Permite la alimentación necesaria para el correcto funcionamiento del motor.
- Protección del circuito: al incluir un driver en el circuito, nos ofrece protección contra posibles sobrecargas y fallas que puedan aparecer.

Para ello empleamos el L298N, un controlador (driver) de motores integrado en un módulo, el cual nos permite encender y controlar dos motores de corriente continua desde una Raspberry Pi, Arduino o cualquier placa similar, variando tanto la dirección como la velocidad de giro.

La corriente máxima que el L298N es capaz de suministrar a los motores es, en teoría, 2A por salida, permitiendo alcanzar hasta los 3A de pico, y una tensión de alimentación de 3V a 35V. Sin embargo, el L298N tiene una eficiencia energética extremadamente baja. La electrónica supone una caída de tensión de unos 3V, es decir, la tensión que recibe el motor es unos 3V inferior a la tensión de alimentación, todo ello unido a la gran cantidad de calor que disipa el driver.

Estas pérdidas se traducen en que, a efectos prácticos, es difícil que podamos obtener más de 0.8-1A por fase sin exceder el rango de temperatura de funcionamiento.

El L298N tiene la ventaja de que incorpora protecciones contra efectos que pueden producirse al manejar motores de corriente continua. Dispone de protecciones contra situaciones de elevada intensidad, temperatura, y diodos de protección contra polaridades incorrectas.

Dicho módulo cuenta con todos los componentes necesarios para funcionar sin necesidad de elementos adicionales entre los que incluye un regulador **LM7805** que suministra 5V a la parte lógica del integrado L298N. Además cuenta con jumpers de selección para habilitar cada una de las salidas del módulo (A y B). La **salida A** esta conformada

por **OUT1** y **OUT2** y la **salida B** por **OUT3** y **OUT4**. Los pines de habilitación son **ENA** y **ENB** respectivamente.

Los pines **IN1**, **IN2**, **IN3** y **IN4**; son los pines de entrada donde se conectan a pines digitales de Arduino para poder controlar el motor.

La alimentación del driver es variable existiendo dos modos de funcionamiento, ya que dispone de un regulador de tensión de 5v incorporado. Cuando el jumper de selección de 5V se encuentra activo, el módulo permite una alimentación de entre 6V a 12V DC. Como el regulador se encuentra activo, el pin marcado como +5V tendrá un voltaje de 5V DC. Este voltaje se puede usar para alimentar la parte de control del driver ya sea un micro-controlador o un Arduino no debiendo superar los 500 mA.

Cuando el jumper de selección de 5V se encuentra inactivo, el driver permite una alimentación de entre 12V a 35V DC. Como el regulador no está funcionando, tendremos que conectar el pin de +5V a una tensión de 5V para alimentar la parte lógica del L298N.

En este caso se utilizará una fuente de 11,1V manteniendo el jumper activo, y así alimentar la parte de control del driver.

En la figura 5.2 se muestra el módulo empleado con sus diferentes pines al detalle.

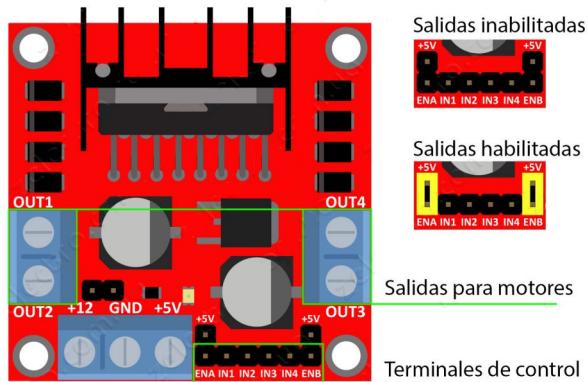


Figura 5.2: Pines de entrada/salida del módulo L298N empleado.

Centrándonos a un nivel más bajo, dicho módulo incorpora, lo que conocemos en electrónica como puente en H, el cual es un circuito electrónico que permite cambiar el sentido de giro de los motores de corriente continua. Este circuito lo componen cuatro interruptores, los cuales pueden ser mecánico o electrónicos (diodos, transistores...) de forma que, en función de cuales se abran o cierren, permitirán cambiar el sentido de alimentación de un motor. Los podemos encontrar como circuitos integrados, pero también se pueden crear con componentes discretos.

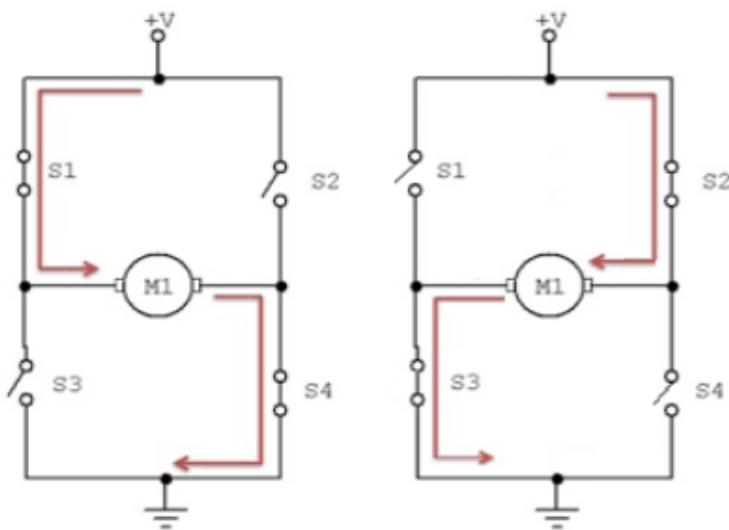


Figura 5.3: Diagrama de funcionamiento del puente H.

El funcionamiento de un puente H es bastante simple. Siguiendo la Figura 5.3 podemos observar que, cuando se cierran los interruptores **S1** y **S4**, y se abren **S2** y **S3**, se aplica una tensión positiva en bornes del motor, así que este gira en un sentido y cuando se abren los interruptores **S1** y **S4** y se cierran **S2** y **S3**, se invierte la tensión en bornes del motor, haciendo que el motor gire en sentido contrario.

Y, por contra, si todos los interruptores están abiertos el motor no girará.

Para el caso del proyecto, hemos empleado utilizado las salidas para la conexión de dos motores, empleando así todas las que el driver proporciona. Una de ellas para traccionar el vehículo y la segunda para el accionamiento de la dirección.

Se ha optado finalmente por emplear un motor DC para la dirección debido que el chasis empleado para la construcción del robot ya lo traía así incorporado y se ha decidido reutilizar. De igual modo se podría haber empleado un servomotor para el accionamiento de la dirección, pero debido a una mayor simplicidad en el montaje y puesto que el motor ya incorporado dispone de una mejor sujeción al chasis así se ha decidido.

### 5.1.5. Motores

En la figura 5.4 se muestra el motor de dirección empleado junto con el sistema de topes de recorrido que incorpora.

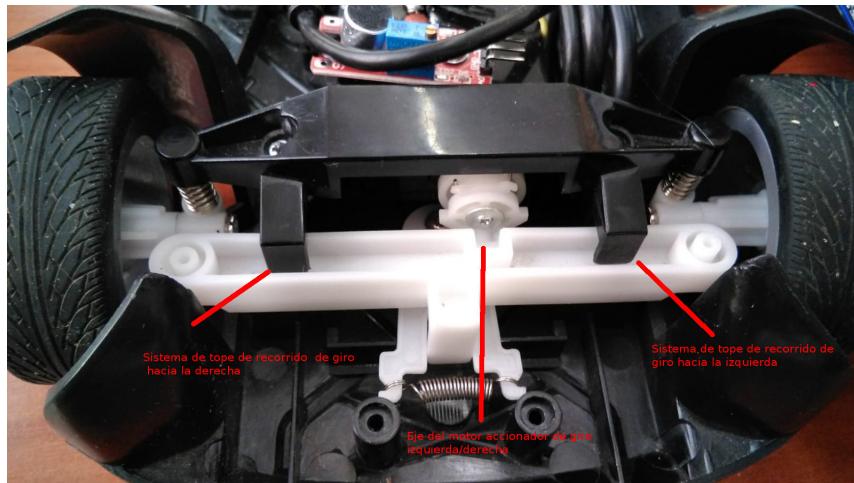


Figura 5.4: Sistema de accionamiento de la dirección.

Motor de tracción del vehículo situado en su parte trasera, figura 5.6.

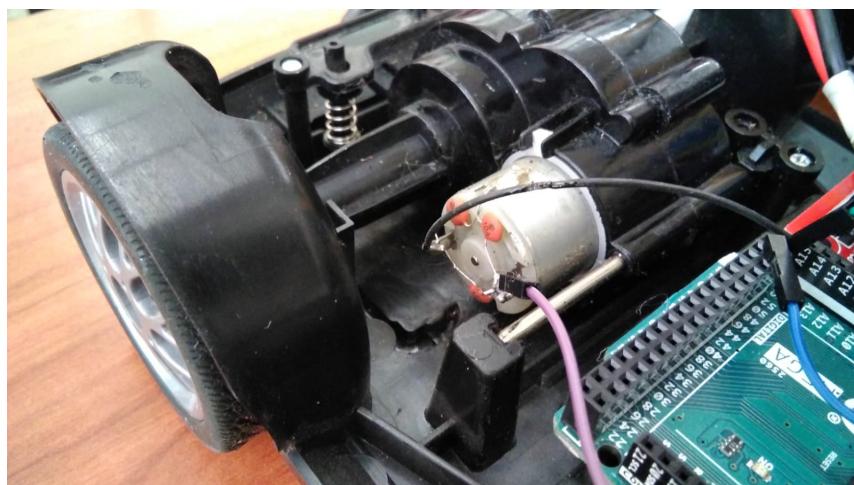


Figura 5.5: Vista del motor de tracción.

### 5.1.5.1. Conexiónado

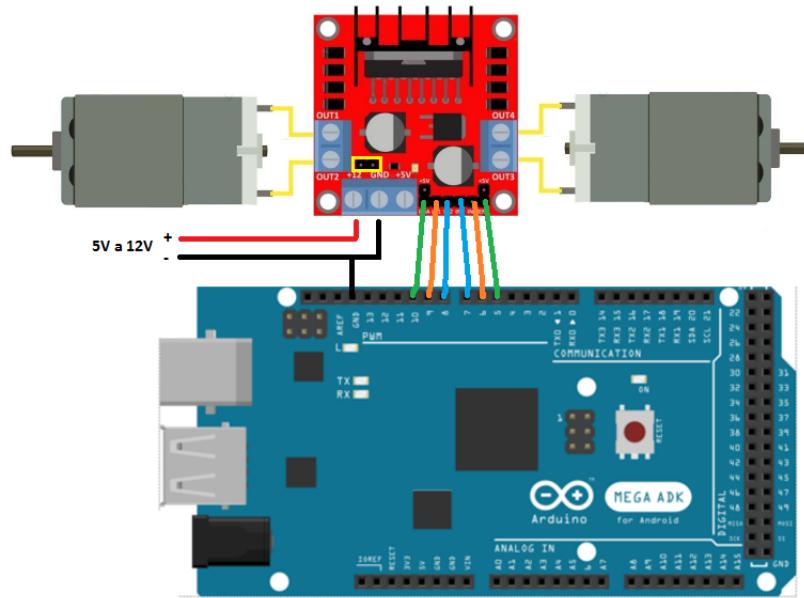


Figura 5.6: Conexiónado del conjunto motores, driver y Arduino.

## 5.2. Alimentación

Resulta de vital importancia el dotar de energía nuestro proyecto siendo una de las partes más importantes y críticas del mismo. La selección de los componentes adecuados requiere una realización previa de estudio de las necesidades que se desean cubrir puesto que una decisión equivocada puede implicar un defectuoso comportamiento del conjunto.

Por tanto, cuando se escoja la batería se deben tener en cuenta los siguientes puntos:

- Consumo máximo del robot:** A partir de las características de todos los componentes, se tiene que mirar cuál será su consumo máximo (motores a pleno rendimiento, electrónica consumiendo al máximo, etc.), y una vez determinado, buscar una fuente de alimentación que sea capaz de proporcionar esta corriente.
- Pico de consumo máximo:** Cuando un motor se pone en marcha, se origina un pico de consumo por la resistencia que tiene a moverse. Este pico será mayor cuanta más velocidad se da y/o mayor carga se desee mover. La batería debe ser capaz de proporcionar estos picos y no ver comprometido su funcionamiento. Si no fuera capaz, provocaría una bajada de tensión para proporcionar esta corriente, y podría apagar el resto de la electrónica. Este valor se suele encontrar en las baterías como C. Por ejemplo, si la batería es de 1000 mAh con un C de 10, podrá suministrar picos de hasta 10000 mA, o lo que es lo mismo, 10 Amperios.

3. **Capacidad:** Todas las baterías recargables dan una cifra de carga, se suele expresar en Ah (Amperios/hora) o mAh (miliamperios/hora). Esta cifra indica cuantos amperios/hora es capaz de dar la batería antes de descargarse. Por tanto, si el consumo es de 10 mA y la batería es de 100 mAh, el equipo podrá estar en marcha durante 10 horas, si por el contrario el consumo es de 100mA y la batería es de 10mAh, el equipo solo podrá estar en funcionamiento 6 minutos.
4. **Voltaje:** Este valor dependerá de las tensiones que se necesiten en el circuito, ya sea por parte del motor, o por parte de la electrónica de control.

En ocasiones, muchos de los diseños de los robots se realizan incorporando dos baterías separadas. Utilizando esta configuración sepáramos la parte digital (sistema de control y sensado) de la parte analógica ( motores). Los motores son muy ruidosos, y a través de la alimentación pueden inducir ruidos al resto de circuitería. En el mejor de los casos el sistema será lo suficientemente robusto para soportar estos problemas, pero muchas veces este ruido falsea las medidas, o hasta puede provocar que la parte del control se resetee.

Por esta razón, se ha optado por el uso de dos baterías para la alimentación del conjunto. Una para dotar de energía a la placa de control y otra para la alimentación de los motores debido a la gran cantidad de energía que éstos demandan y su elevado consumo.

Para la alimentación de la placa Raspberry Pi, se ha optado por la utilización de una batería de Litio desarrollada específicamente para su uso con este modelo de placas el cual permite una integración perfecta. Dicho módulo de alimentación queda descrito en la subapartado correspondiente de herramientas utilizadas [3.3.7](#).

Imagen de la Raspberry Pi junto con su módulo de expansión de batería:



Figura 5.7: Conjunto Raspberry Pi y módulo de expansión de alimentación.

Imagen de la batería LiPo para la alimentación de los motores:



Figura 5.8: Batería LiPo que alimenta los motores.



Figura 5.9: Alojamiento de la batería LiPo.

### 5.2.0.1. Alimentación USB/Protoboard

Para alimentar la protoboard sin tener que modificar un cable USB, se ha decidido utilizar una pequeña tarjeta que se conecta en los carriles de tensión de la tarjeta Protoboard, y proporciona los 5 V de entrada del USB (o de un conector Jack) a las líneas de alimentación de la protoboard. Además este dispositivo integra un interruptor, que permitirá activar y desactivar los motores, y puede regular la tensión a 3,3 Voltios.

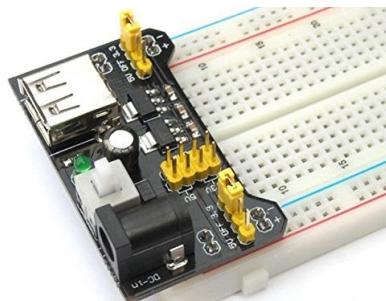


Figura 5.10: Adaptador de entrada USB a protoboard.

La alimentación de este módulo se puede realizar de dos formas: mediante el uso de una pila o batería, o mediante un cable USB. Para saber que está correctamente alimentado, este dispositivo incorpora un indicador led que avisará si se está usando de forma correcta. Otra característica importante de este módulo, es la posibilidad de elegir la tensión de salida que proporciona, pudiendo seleccionar una alimentación de 5V y otra de 3,3V a la vez.

En la Figura 5.11 puede verse el esquema eléctrico del módulo.

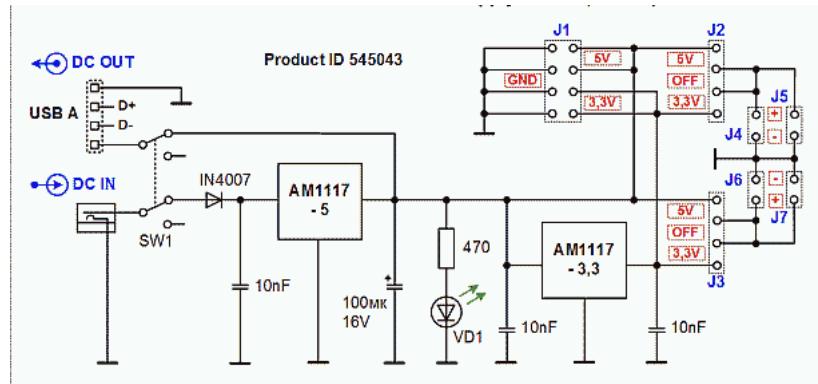


Figura 5.11: Esquemático del adaptador de entrada USB a protoboard.

Siguiendo el esquema 5.11 Podemos diferenciar los distintos componentes según su funcionalidad dentro del módulo:

- **Alimentación USB:** Esta entrada solo se utiliza para la alimentación de 5 voltios mediante un conector USB hembra. Los pines centrales, D+ y D- no están conectados a ningún elemento, por lo que no puede recibir datos.
- **Alimentación externa:** Se trata de un conector tipo Jack hembra al que se le puede colocar cualquier tipo de batería o pila que se encuentre entre los rangos de 6 y 12V permitidos por el módulo.

- **AM1117-5 y AM1117-3.3:** Son los dos reguladores de tensión que incorpora el módulo MB102 para regulación de tensión de salida hasta los valores 5 y 3,3V respectivamente.
- **Diodos y condensadores** También incorpora una serie de diodos zener<sup>2</sup> de protección ante posibles inversiones de polaridad junto con un diodo led para comprobar el correcto funcionamiento del módulo.
- **SW1:** Botón cuya finalidad es encender o apagar el módulo de alimentación.
- **J1:** Serie de pines en los cuales se tiene una tensión de 5V y 3.3V para la conexión de los diferentes elementos. De los ocho pines que dispone, cuatro pines son de GND, dos pines son de 5V y los otros dos pines restantes son de 3.3V.



Figura 5.12: Vista del conector J1.

- **J2 y J3:** Pines para la selección de la tensión de salida del módulo. Para seleccionar la tensión debemos conectar un jumper<sup>3</sup> entre los pines en función de la tensión deseada.



Figura 5.13: Vista del conector J2 y J3.

<sup>2</sup>El diodo Zener es un diodo de silicio altamente dopado ideado para que funcione en las zonas de rupturas. Estos tipos de diodos resultan fundamentales en los reguladores de tensión proporcionando valores prácticamente constantes con independencia ante variaciones de tensión, resistencia y temperatura.

<sup>3</sup>En electrónica y en particular en informática, un jumper es un elemento utilizado para interconectar dos terminales de manera temporal sin tener que efectuar una operación que requiera herramienta adicional cerrando circuito eléctrico del que forma parte.

- **J4 y J5:** Pines que incorpora el modulo en su parte inferior para la conexión de este en una protoboard.

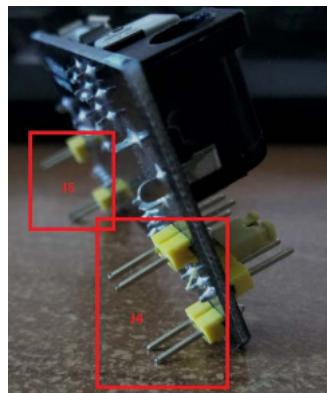


Figura 5.14: Vista del conector J4 y J5.

### 5.2.1. Interconexión entre módulos y fijación

La conexión entre los diferentes módulos se realiza utilizando cables USB, y cables de interconexión entre la Raspberry/Arduino con la protoboard utilizándose la técnica conocida como Wire-Wrap o cables tipo jumper-wire.

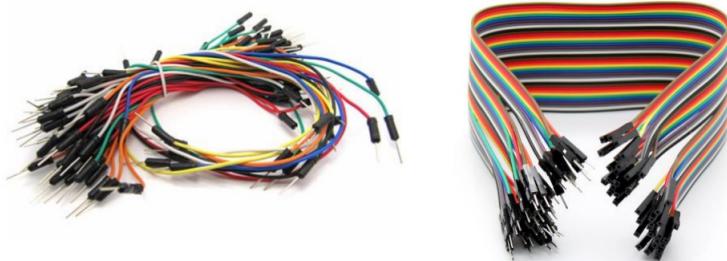


Figura 5.15: Cables de interconexión para protoboard.

Para la fijación de los diferentes módulos al chasis se ha empleado tornillería y bridas, éstas últimas para mantener el cableado más ordenado.

### 5.2.2. Conexionado general

La siguiente imagen 5.20 muestra una visión general de la disposición de todos los elementos que conforman el vehículo junto con sus conexiones:

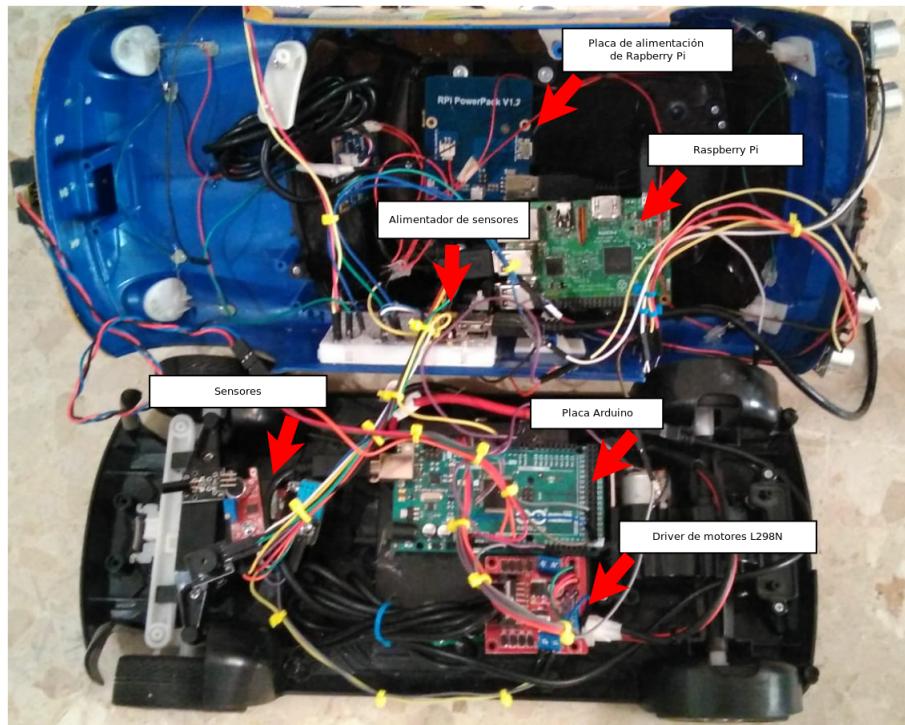


Figura 5.16: Vista del conexionado general del robot.

La Tabla 5.1 muestra los pines utilizados, modo de configuración y propósito en la placa Arduino:

Pin Arduino	Modo	Pin destino
A0	INPUT	Entrada analógica SHARP GP2D12
D48	INPUT	Entrada digital del Buzzer
D38	INPUT	Entrada digital del micrófono KY-038
A2	INPUT	Entrada analógica del sensor de llama YG1006
D4	OUTPUT	TRIG del sensor HC-SR04 trasero derecho
D5	INPUT	ECHO del sensor HC-SR04 trasero derecho
D7	OUTPUT	TRIG del sensor HC-SR04 trasero izquierdo
D2	INPUT	ECHO del sensor HC-SR04 trasero izquierdo

<b>D3</b>	INPUT	Entrada digital del sensor DTH11
<b>A1</b>	OUTPUT	Entrada analógica del fotoresistor
<b>D35</b>	INPUT	Entrada analógica del sensor de gas MQ-2 fotoresistor
<b>D13</b>	OUTPUT - PWM	Entrada ENA del driver L298N para control de tracción
<b>D12</b>	OUTPUT	Entrada IN3 del driver L298N para control de tracción
<b>D11</b>	OUTPUT	Entrada IN4 del driver L298N para control de tracción
<b>D10</b>	OUTPUT - PWM	Entrada ENB del driver de motores L298N para control de dirección
<b>D9</b>	OUTPUT	Entrada IN1 del driver L298N para control de dirección
<b>D8</b>	OUTPUT	Entrada IN2 del driver L298N para control de dirección
<b>D6</b>	OUTPUT	Entrada positiva de los leds de iluminación del vehículo
<b>D37</b>	OUTPUT	Entrada positiva del puntero láser del vehículo

Tabla 5.1: Pines utilizados en la placa Arduino Mega.

La Figura 5.17 muestra el esquemático global de SensorRS.

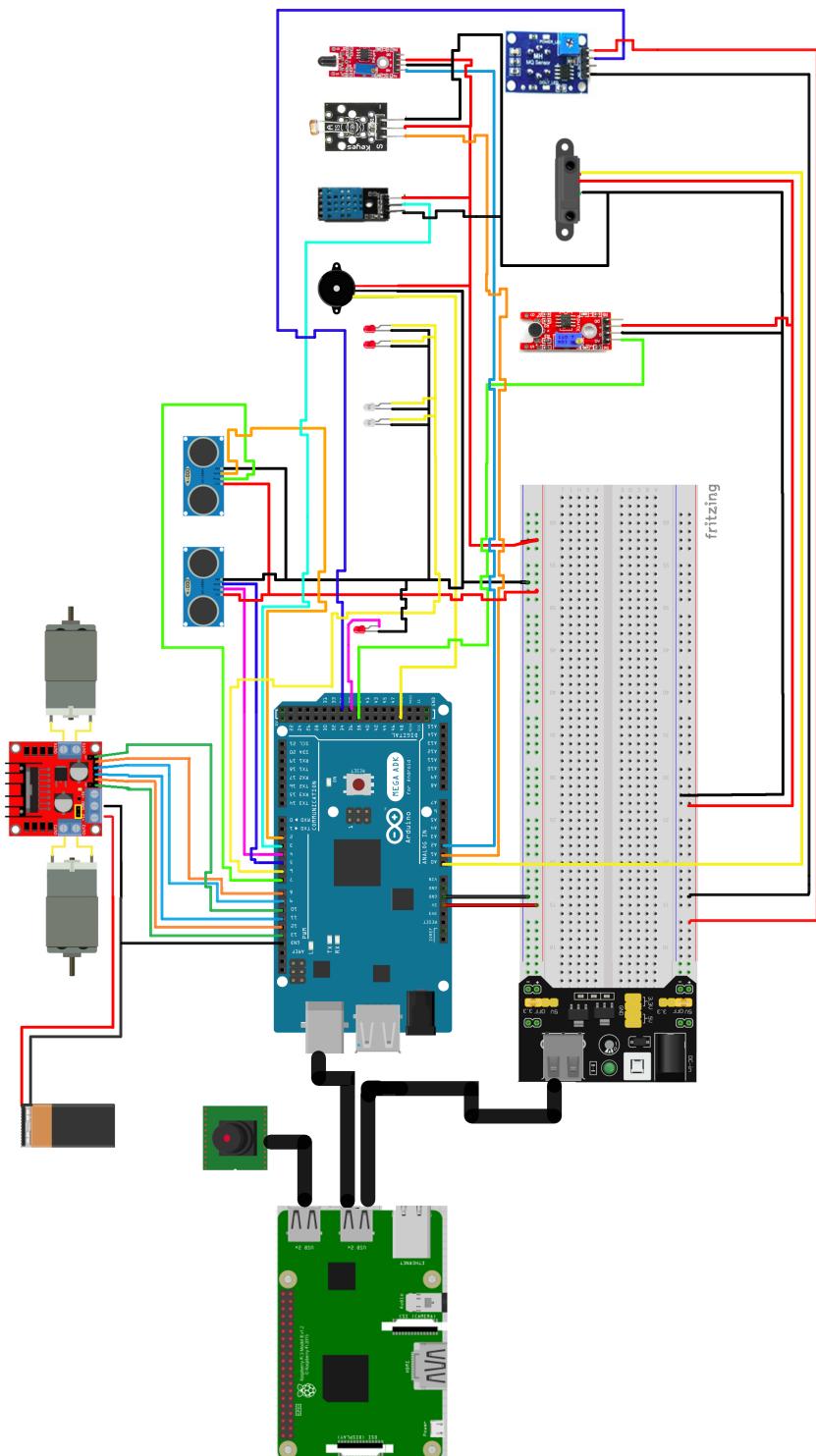


Figura 5.17: Esquemático del conexionado general del robot.

## 5.3. Iluminación

### 5.3.1. Leds

Con el fin de dotar de iluminación al vehículo, se ha incorporado una serie de leds en los faros del vehículo tanto en su parte delantera como trasera. Dichos leds van conectados a un pin del Arduino con la finalidad de poder encenderlos o apagarlos al deseo del operador.

La alimentación se encuentra conectada a un pin digital del arduino mientras que la masa a una entrada GND del mismo.

A continuación se muestra una imagen de los faros del vehículo en funcionamiento.



(a) Iluminación frontal

(b) Iluminación trasera

Figura 5.18: Vehículo SensorRS con la iluminación activada.

Por otra parte, se ha incorporado un sensor detector de luminosidad para el encendido automático de luces en lugares de baja iluminación.

#### 5.3.1.1. Conexionado

Representación del conexionado de los leds, figura 5.19.

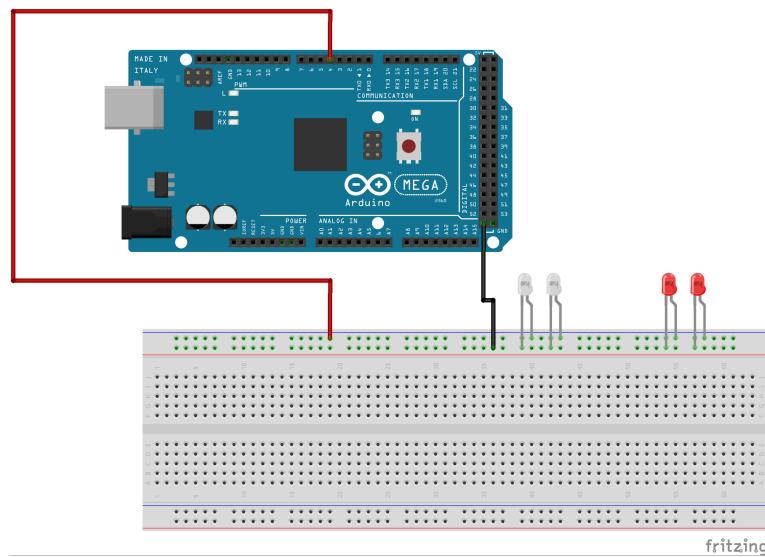


Figura 5.19: Conexiónado de los leds de iluminación.

### 5.3.2. Puntero láser

Junto con los leds de iluminación también se ha conectado un puntero láser accionable por el operador.

#### 5.3.2.1. Conexiónado

El conexionado del puntero láser se ha realizado del mismo modo que para un led convencional resultando válido el esquema de la Figura 5.19.



Figura 5.20: Vehículo SensorRS con el puntero láser situado en un lateral.

## 5.4. Sensores

En la presente sección recogemos una descripción del funcionamiento y conexionado de los sensores más característicos utilizados en el proyecto.

### 5.4.1. Sensores ultrasonidos

Durante el movimiento del robot, es necesario la utilización de sensores de proximidad para evitar la colisión con objetos. En este proyecto, los sensores empleados son dos sensores ultrasonidos del modelo HC-SR04.

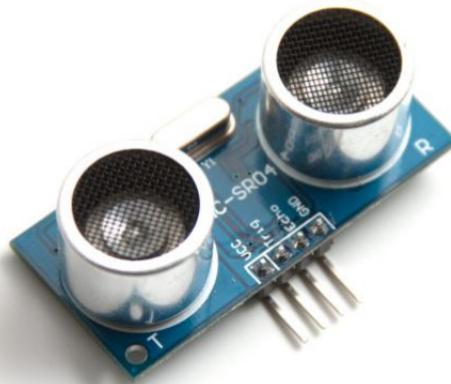


Figura 5.21: Sensores ultrasonidos empleados en el proyecto.

Los sensores irán dispuestos en la trasera del robot teniendo especial cuidado con las posibles interferencias entre dichos sensores, muy frecuentes debido a los rebotes. Es por ello que se han situado en ambos laterales orientados hacia el exterior.

Para el correcto funcionamiento de estos sensores, se da un pulso de nivel alto de un mínimo de 10 s en el pin al que se conecta el trig del sensor y a continuación se vuelve a poner a cero, tal y como se especifica en el datasheet. Como consecuencia de este pulso, se generará otro pulso en el pin de entrada donde se conecta la señal echo. El ancho del pulso recibido determina la distancia a la que se encuentra el objeto, cuanto mayor es la duración del pulso más lejos se encuentra el objeto. Utilizando una interrupción que se produzca por cada flanco de subida y bajada, se puede obtener el ancho de dicho pulso y mediante el uso de un timer se puede medir el tiempo transcurrido entre dichos flancos.

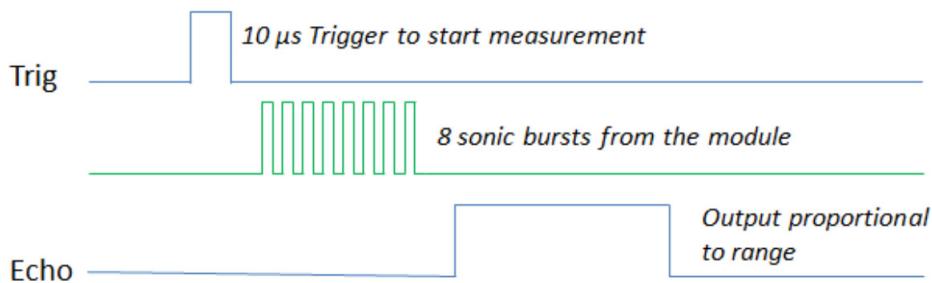


Figura 5.22: Pulses para la correcta lectura del sensor ultrasonidos.

Para calcular la distancia a la que se encuentran los objetos situados frente al sensor, se emplea la fórmula:

$$\text{Distancia(cm)} = \frac{\text{tiempo}(\mu\text{s})}{58} \quad (5.1)$$

En dicha fórmula, el tiempo se encuentra en microsegundos y la distancia a la que se encuentra el objeto se obtiene en centímetros.

Otra forma de calcular la distancia sería utilizando la velocidad del sonido (340 m/s) y sabiendo que la distancia que recorre es en sentido de ida y de vuelta, por tanto habría que dividir la distancia calculada entre dos.

El comportamiento de la función encargada de medir la distancia de los sensores ultrasonidos se puede descomponer en dos diagramas de flujo, uno encargado de dar los pulsos para iniciar la lectura y otro que se encarga de medir la distancia.

Indicar que la distancia que se puede medir utilizando este sensor de proximidad abarca el rango 2 cm – 400cm, tal y como se especifica en las características técnicas del mismo.

Por otra parte, es posible que objetos de pequeña dimensión o bien de superficie rugosa den problemas a la hora de medir las distancias, debido a que no se refleja la señal o bien porque se producen reflexiones.

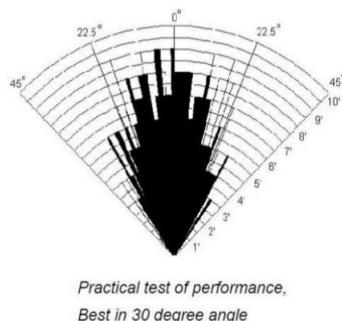


Figura 5.23: Directividad del sensor HC-SR04.

Los sensores ultrasonidos se emplearán tanto para medir distancias con los objetos próximos como para detener el vehículo, si es necesario, cuando hay riesgo de colisión.

#### 5.4.1.1. Conexionado

El sensor HC-SR04 dispone de 4 pines, la toma de tierra **GND**, pin de echo **ECHO**, trig **TRIG** y para la alimentación **VCC** (5V). En la siguiente imagen puedes ver el esquema de conexión con Arduino.

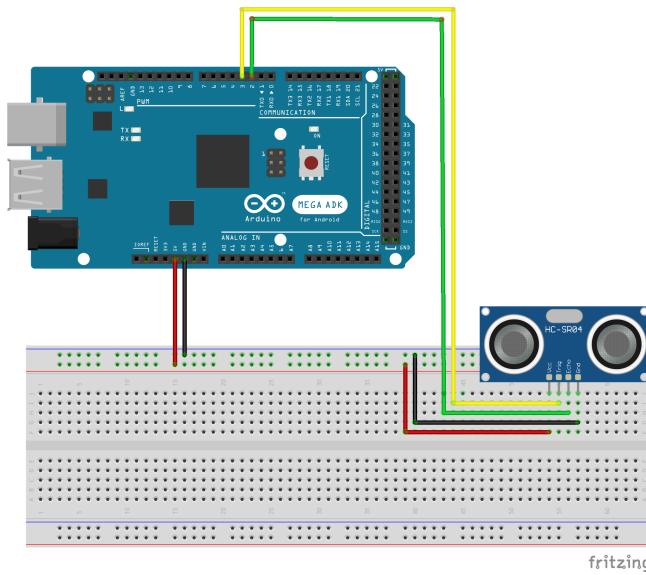


Figura 5.24: Conexionado del sensor HC-SR04 con Arduino.



Figura 5.25: Sensores ultrasónicos situados en la parte trasera del vehículo.

#### 5.4.2. Sensor de temperatura y humedad DHT11

Para poder controlar la temperatura y la humedad relativa de la zona por la que se mueve el robot, se ha decidido añadir un sensor de temperatura. En concreto se trata del sensor DHT11, que se controla por un pin de propósito general.

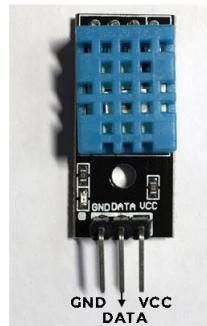


Figura 5.26: Sensor de temperatura DHT11.

El sensor DHT11 en su versión PCB consta de tres pines. Un pin **VCC** de alimentación, uno **DATA** para la transferencia de datos que se conecta al microcontrolador y un pin que conectado a tierra **GND**.

#### 5.4.2.1. transmisión de datos

A pesar de que conectemos el pin de datos a un pin digital de nuestro microcontrolador, se trata de un dispositivo analógico. Dentro del propio dispositivo se hace la conversión entre analógico y digital.

Por lo tanto, partimos de una señal analógica que luego es convertida en formato digital y se enviará al microcontrolador. La trama de datos es de 40 bits correspondiente a la información de humedad y temperatura del DHT11.

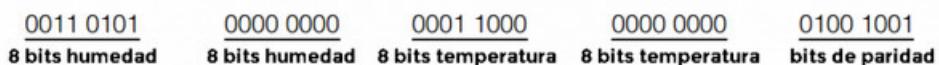


Figura 5.27: Disposición de los bits de datos del DHT11.

El primer grupo de 8-bit es la parte entera de la humedad y el segundo grupo la parte decimal. Lo mismo ocurre con el tercer y cuarto grupo, la parte entera de la temperatura y la parte decimal. Por último los bits de paridad para confirmar que no hay datos corruptos.

Estos bits de paridad lo único que hacen es asegurarnos de que la información es correcta, sumando los 4 primeros grupos de 8-bit. Esta suma debe ser igual a los bits de paridad. Si nos centramos en la imagen anterior y sumamos los bits, comprobamos que todo está correcto.

$$00110101 + 00000000 + 00011000 + 00000000 = 01001101 \quad (5.2)$$

El sensor DHT11 tiene una resolución en el sensor de temperatura de un grado centígrado y en la humedad relativa del 1 por ciento y se recomienda respetar un intervalo de al menos un segundo entre las tomas de medidas para el correcto funcionamiento del dispositivo. Ante frecuencias de trabajo mayores, se daña el dispositivo.

#### 5.4.2.2. Conexiónado

El sensor DHT11 integrado dentro de una PCB ya viene con la resistencia pull-up integrada. Esta resistencia puede resultar muy útil en ocasiones, pero si añadimos un cable de más de 20 metros, deberemos tener en cuenta dicho factor.

Este modelo de DHT11, como comentamos anteriormente, dispone de 3 pines, la toma de tierra **GND**, para los datos **DATA** y para la alimentación **VCC** (de 3,5V a 5V). En la siguiente imagen puedes ver el esquema de conexión con Arduino.

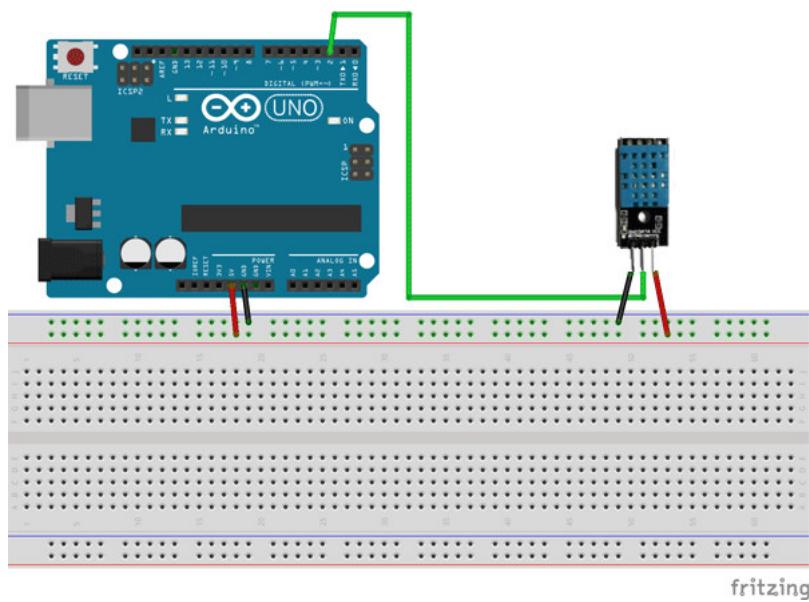


Figura 5.28: Conexiónado de DHT11 con Arduino.

#### 5.4.3. Sensor de gases

Para la detección de gas se ha empleado un sensor basado en la serie MQ ampliamente utilizados y bastante populares en muy diversas aplicaciones entre ellas incluida la robótica. Existen una lista grande de sensores MQ donde cada uno fue diseñado para detectar un tipo de gas o sustancia específica (o conjunto de ellas). El utilizado en el presente trabajo es un MQ-2 que detecta gases combustibles, incluyendo propano, butano, gas natural y humo. Para su activación y prueba basta con usar un encendedor para mostrar su operación. Otros modelos pueden detectar gases más raros como el

hidrógeno o CO (monóxido de carbono), o incluso gases tóxicos como el amoníaco.



Figura 5.29: Sensor MQ-2.

El sensor puede ser conectado de manera digital, analógica o mixta. Con la conexión digital podemos usarlo para detección de seguridad, con la analógica podemos medir la cantidad de gas en el aire en ppm (partes por millón), y si usamos ambos a la vez, podemos obtener una funcionalidad mixta.

Este tipo de sensores presentan la particularidad de que necesitan un tiempo de “calentamiento” antes de poder arrojar lecturas válidas. Es aconsejable un mínimo de 2 minutos de calentamiento previo comenzando una vez alimentado el sensor. Trascurrido un par de minutos entonces se podrá realizar una medición efectiva.

En el presente proyecto nos hemos decantado por utilizar la detección de gas mediante el uso de su salida digital. Dicha salida digital, con nombre **D0**, tendrá un estado 0 o 1, representando 0 = 0v, y 1 = 5v aproximadamente, y que representan 0 = detección de gas, y 1 = sin detección de gas.

#### 5.4.3.1. Calibración del punto de detección de gas

El sensor dispone de un potenciómetro (un componente que posee una ranura para su ajuste con un pequeño destornillador) que nos permite calibrar el umbral donde queremos que notifique haber detectado gas. La calibración nos permite decidir el nivel de sensibilidad del módulo a la detección de gases.

La manera de calibrarlo es mientras está conectado, para ello simplemente giramos el potenciómetro hacia la izquierda hasta que un led se active, siendo el testigo indicativo de detección de gas. El ajuste de la sensibilidad dependerá del resultado deseado, si se desea un mayor grado de sensibilidad, debemos girar hacia la derecha hasta que se apague el led de testigo. Sin embargo si se quiere un mayor grado de tolerancia, se deberá girar hacia la derecha hasta el nivel deseado.



Figura 5.30: Vista del potenciómetro del sensor MQ-2.

#### 5.4.3.2. Conexionado

Conexionado del sensor MQ-2 en su modo digital. Siendo **D0** pin digital, **GND** a tierra y **VCC** a alimentación 5V.

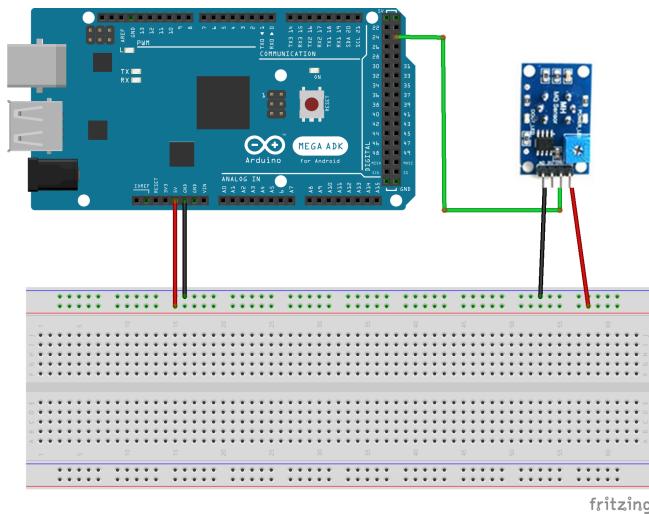


Figura 5.31: Sensor MQ-2 utilizando el pin digital del microcontrolador.

#### 5.4.4. Sensor de sonido

Para la detección de sonidos hemos empleado un micrófono el cual actúa como un transductor convirtiendo las ondas sonoras en señales eléctricas.

La salida producida por un micrófono es una señal eléctrica analógica que representa el sonido recibido. Sin embargo, esta señal suele ser demasiado baja para ser medida siendo necesario amplificarla.

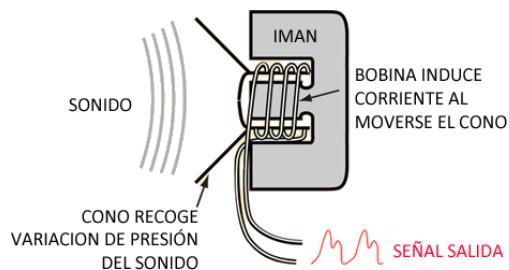


Figura 5.32: Amplificación de la señal de audio recibida.

La placa utilizada, KY-038 incorpora un micrófono junto con un comparador LM393, el cual permite obtener la lectura tanto analógica como digitalmente.

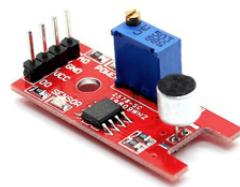


Figura 5.33: Vista del sensor de audio KY-038 utilizado.

Los pines de los que dispone el sensor son: **AO**, Analog Output, **GND**, Ground, **VCC**, Alimentación de 3.3V a 12V y **DO**, Digital Output.

Lo más característico de este sensor es que la señal que nos entrega es digital y analógica, lo cual nos permite decidir cual utilizar según nuestras necesidades. Si necesitamos saber el valor del sensor, podremos utilizar directamente la salida analógica para conseguir los datos crudos. Sino, podemos utilizar la salida digital, la cual se activa o se desactiva según si el sensor llega a medir la intensidad del sonido que le configuremos, mediante la definición de la sensibilidad del sensor, de igual modo al sensor de detección de gases.

#### 5.4.4.1. Conexiónado

Conexiónado del sensor KY-038 en su modo digital:

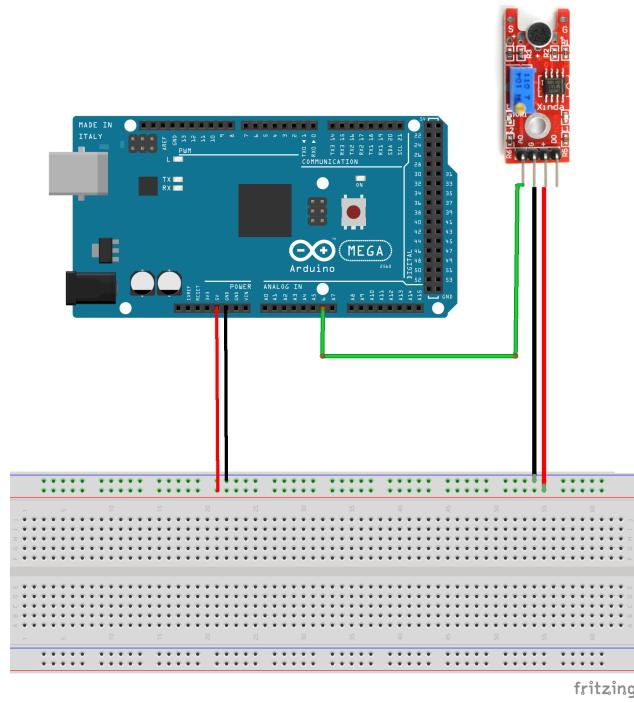


Figura 5.34: Sensor KY-038 utilizando el pin digital del microcontrolador.

#### 5.4.5. Fotoresistor

Un fotoresistor, o LDR (light-dependent resistor) es un dispositivo cuya resistencia varía en función de la luz recibida. Podemos usar esta variación para medir, a través de las entradas analógicas, una estimación del nivel del luz percibida en el entorno del vehículo siendo de utilidad para el encendido automático de la iluminación en aquellas situaciones donde resulte necesaria.

Un fotoresistor está formado por un semiconductor, típicamente sulfuro de cadmio CdS. Al incidir la luz sobre él algunos de los fotones son absorbidos, provocando que electrones pasen a la banda de conducción y, por tanto, disminuyendo la resistencia del componente. Por tanto, un fotoresistor disminuye su resistencia a medida que aumenta la luz sobre él. Los valores típicos son de 1 Mohm en total oscuridad, a 50-100 Ohm bajo luz brillante.

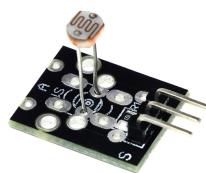


Figura 5.35: Vista del fotoresistor utilizado.

Por tanto, un fotoresistor disminuye su resistencia a medida que aumenta la luz sobre él. Los valores típicos son de 1 Mohm en total oscuridad, a 50-100 Ohm bajo condiciones de luz brillante.

#### 5.4.5.1. Conexiónado

Conexiónado del fotoresistor a la placa Arduino:

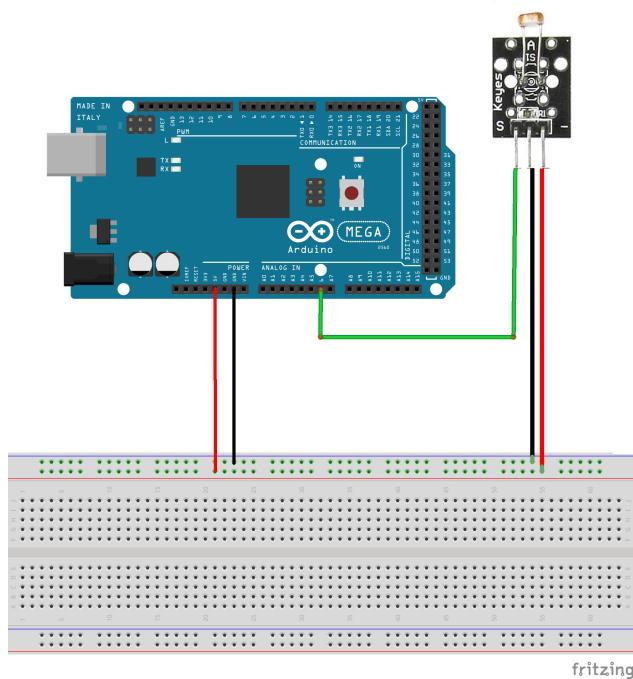


Figura 5.36: Vista del fotoresistor utilizado.

#### 5.4.6. Sensor de llama

Un sensor de llama óptico es un dispositivo que permite detectar la existencia de combustión por la luz emitida por la misma. Esta luz puede ser detectada por un sensor óptico, y ser capturado por las entradas digitales y las entradas analógicas de Arduino.

La llama es un fenómeno de emisión de luz asociado a los procesos de combustión siendo ésta un proceso que desprende grandes cantidades de energía en forma de calor generándose compuestos intermedios que liberan parte de su energía mediante la emisión de luz.

El espectro de emisión de la llama generada depende de los elementos que intervienen en la reacción. En el caso de combustión de productos con carbón en presencia del oxígeno tenemos dos picos característicos en ultravioleta en longitudes de onda de 185nm-260nm y en infrarrojo en longitudes de onda 4400-4600nm.

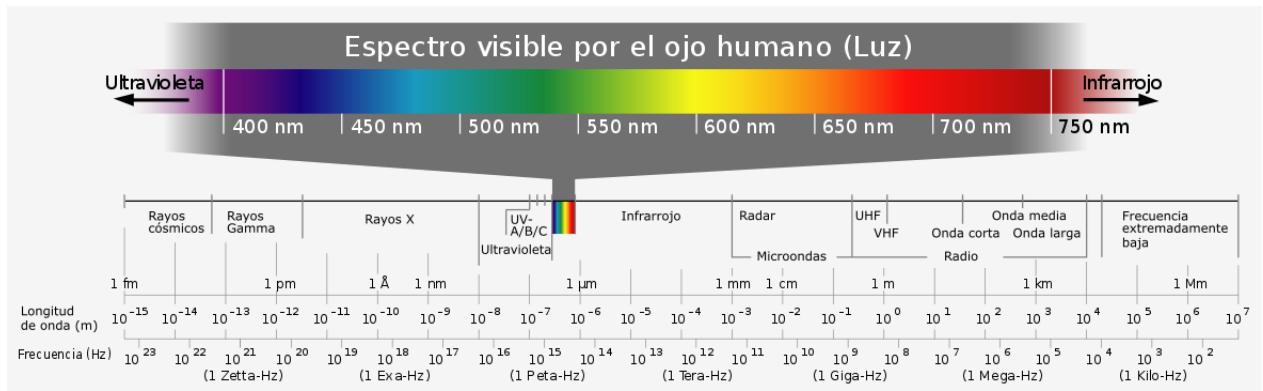


Figura 5.37: Espectro visible por el ojo humano.

Estos dispositivos se ajustan a las longitudes de onda características de la aparición de la llama y normalmente combinan las señales ultravioleta y de infrarrojo.

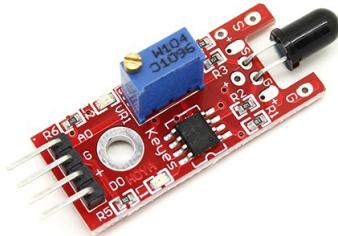


Figura 5.38: Vista del sensor de llama YG1006 utilizado.

Estos sensores constan únicamente de un sensor infrarrojo ajustado al rango de los 760-1100 nm siendo su ángulo de detección de 60°, y la distancia de detección entre 0.40 m a 0.80.

Este tipo de sensores de llama infrarrojos suelen incorporar una placa de medición estándar con el comparador LM393, que permite obtener la lectura tanto como un valor analógico como de forma digital cuando se supera un cierto umbral, que se regula a través de un potenciómetro ubicado en la placa.

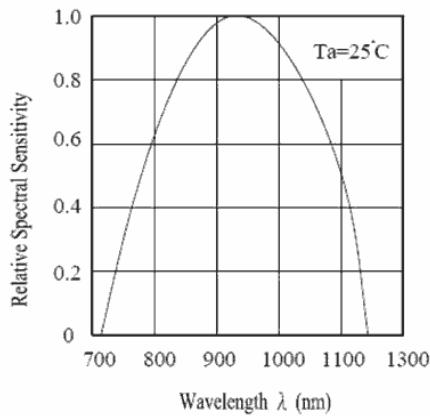


Figura 5.39: Gráfico de sensibilidadpectral del sensor YG1006.

Como vemos en la figura 5.39, las longitudes de onda de estos sensores poco tienen que ver con las emisiones características de las llamas, las cuales pueden variar su espectro según múltiples factores siendo uno de ellos el material en combustión. Pudiéndose abarcar espectros desde tonos azulados de las llamas de gas a los verdosos producidos por el sulfato de cobre. De hecho, estos sensores también son afectados incluso por la iluminación interior, dando lugar a numerosos falsos positivos.

Por tanto, la sensibilidad y fiabilidad de estos detectores de reducido coste no resultan suficientes para considerarlos un auténtico dispositivo de seguridad.

#### 5.4.6.1. Conexionado

El esquema eléctrico es muy simple. Debemos alimentar el módulo conectando **GND** y **VCC** a los pinos correspondientes de Arduino y salida **D0** a una de las entradas digitales de Arduino.

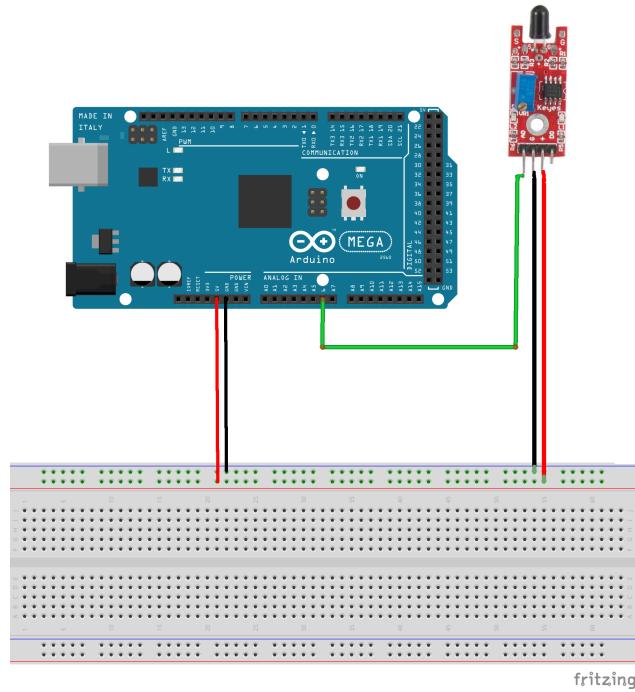


Figura 5.40: Vista del conexionado del sensor de llama YG1006.



Figura 5.41: Sensor de llama instalado en la parte delantera del vehículo.

#### 5.4.7. Sensor de proximidad

La familia de los sensores Sharp GP2Dxx son una gama de sensores muy utilizados tanto en lo que viene a denominarse robótica móvil casera como en el ámbito de inves-

tigación debido principalmente a su facilidad de integración y reducido coste (unos 15 Euros). En la Figura 5.42 puede verse una imagen de un GP2D12.

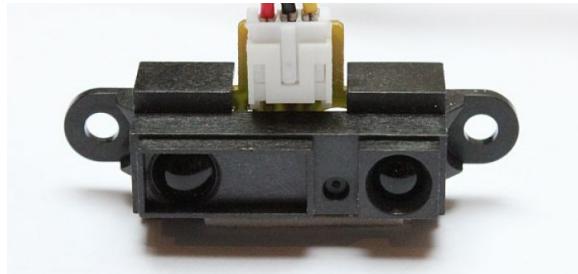


Figura 5.42: Sensor GP2D12.

Los sensores GP2D12 proporcionan una salida analógica entre 0 y 3 voltios dependiendo de la distancia a la que se encuentre el objeto. Dicha salida analógica no es lineal sino que sigue una curva, la cual se muestra en la figura 5.43.

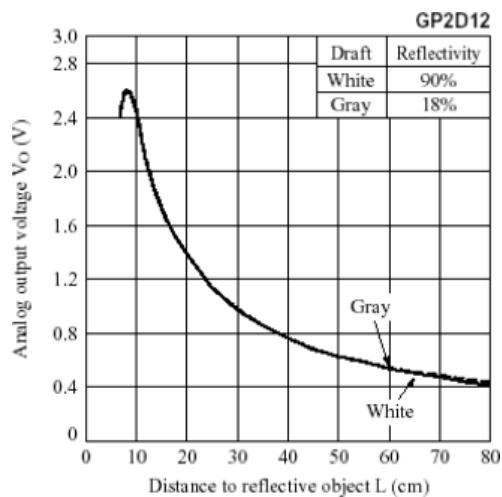


Figura 5.43: Curva de tensión de salida según la distancia al obstáculo.

Estos sensores se basan en el principio de triangulación para realización de las medidas. El elemento situado a la izquierda del sensor según vemos en la figura 5.42 es un led infrarrojo que emite un haz que ser a rebotado por el objeto y posteriormente recogido por el elemento situado a la derecha. Este ultimo se conoce como PSD (Position Sensing Device, Dispositivo de Percepción de Posición) y puede entenderse como una lente situada sobre un array de células sensibles a la luz infrarroja. Dependiendo del ángulo de incidencia del haz rebotado en la lente, se activa una u otra célula del array permitiendo estimar la distancia a la que se encuentra el objeto.

El conexionado del GP2D12 con un microcontrolador requiere de una entrada del conversor analógico-digital a la que se conectará el pin de salida del sensor (el de más a

la izquierda visto de frente según se muestra en la figura 5.48). Los otros dos pines corresponden, respectivamente, con **GND** y con **Vcc**, la tensión de alimentación, que deberá ser de 5 voltios.

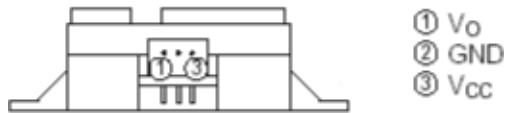


Figura 5.44: Conexiones del GP2D12.

En la siguiente tabla se recoge un resumen de las especificaciones del sensor:

Rango	18-80 cm
Periodo de lectura	40 ms
Máximo ángulo de reflexión	>40 °
Tensión de alimentación	4.5-5.5 V
Ruido de salida	<200 mV
Consumo medio	35 mA
Consumo de pico	200 mA

Tabla 5.2: Resumen de las especificaciones del GP2D12.

#### 5.4.7.1. Conexionado

El esquema eléctrico es muy simple. Debemos alimentar el módulo conectando GND y VCC a los pines correspondientes de Arduino y salida D0 a una de las entradas digitales de Arduino.

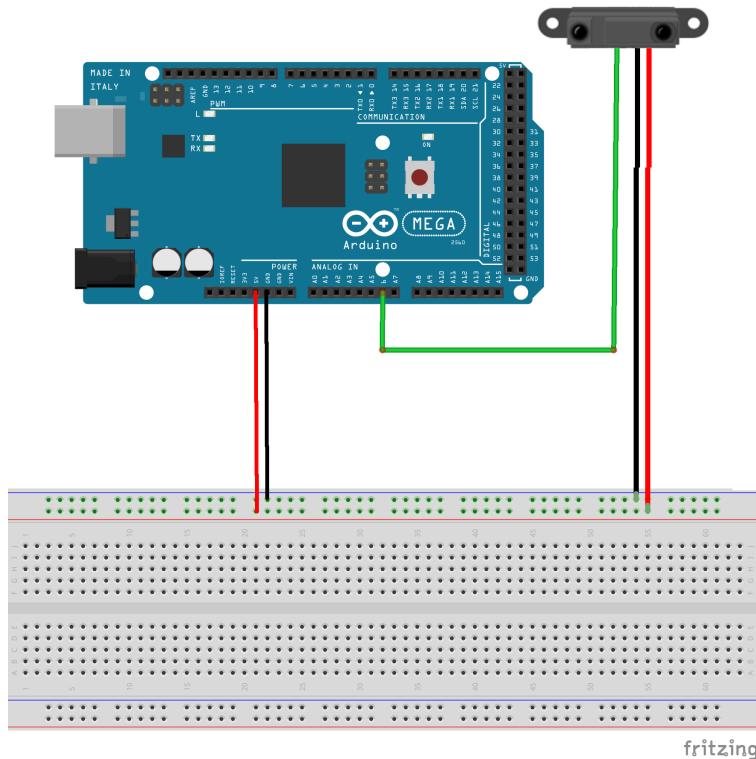


Figura 5.45: Conexión del GP2D12.

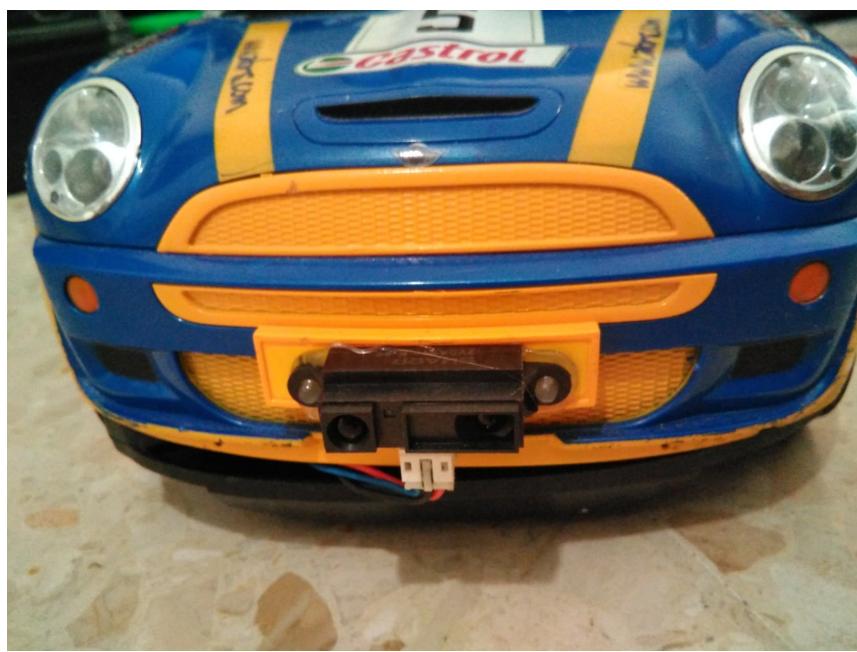


Figura 5.46: Sensor de proximidad Sharp GP2D12 instalado en la parte frontal del vehículo.

### 5.4.8. Buzzer o zumbador

Un buzzer pasivo, un zumbador o altavoz, son dispositivos que permiten convertir una señal eléctrica en una onda de sonido. Son unos dispositivos muy simples ya que no disponen de electrónica interna, por lo que tan solo debemos proporcionar una señal eléctrica para conseguir el sonido deseado.

Por contra, los buzzers activos disponen de un oscilador interno siendo exclusivamente necesario alimentarlos para que se emita un sonido.

Los buzzers pasivos, como el incorporado en el presente trabajo, tienen la particularidad de que al proporcionar y controlar nosotros la señal eléctrica, poseen la ventaja de que podemos variar el tono emitido modificando la señal que aplicamos al pin de control permitiendo la generación de melodías.

En el caso que compete a este proyecto se ha empleado un buzzer para la emisión de sonidos intermitentes a modo de alarma activados tras la detección de alguna situación de riesgo. Estas situaciones son determinadas por parte de alguno de los demás sensores de los que dispone el vehículo, como por ejemplo puede ser la detección de llamas cercanas o ante la presencia de gases peligrosos.

Tanto los buzzers como los altavoces son elementos denominados transductores electroacústicos, es decir, son dispositivos que poseen la propiedad de convertir señales eléctricas en sonido, que para el caso concreto de los buzzers entran en la categoría de los transductores piezoelectrinos. Esto significa que tienen la propiedad de variar su volumen al ser atravesados por corrientes eléctricas aprovechando este fenómeno para hacer vibrar una membrana al atravesarlo con una señal eléctrica y con la consecuente emisión del sonido.

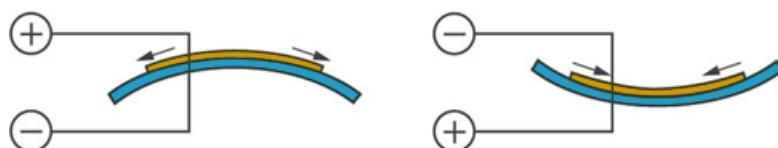


Figura 5.47: Representación de la variación de volumen del material piezoelectrino al paso de la corriente.

#### 5.4.8.1. Conexión

La conexión con Arduino como en la mayoría de los casos resulta bastante sencilla. Simplemente alimentamos el módulo conectando **Vcc** y **GND** a Arduino, y la entrada de señal conectada a cualquier pin digital de Arduino configurado como salida.

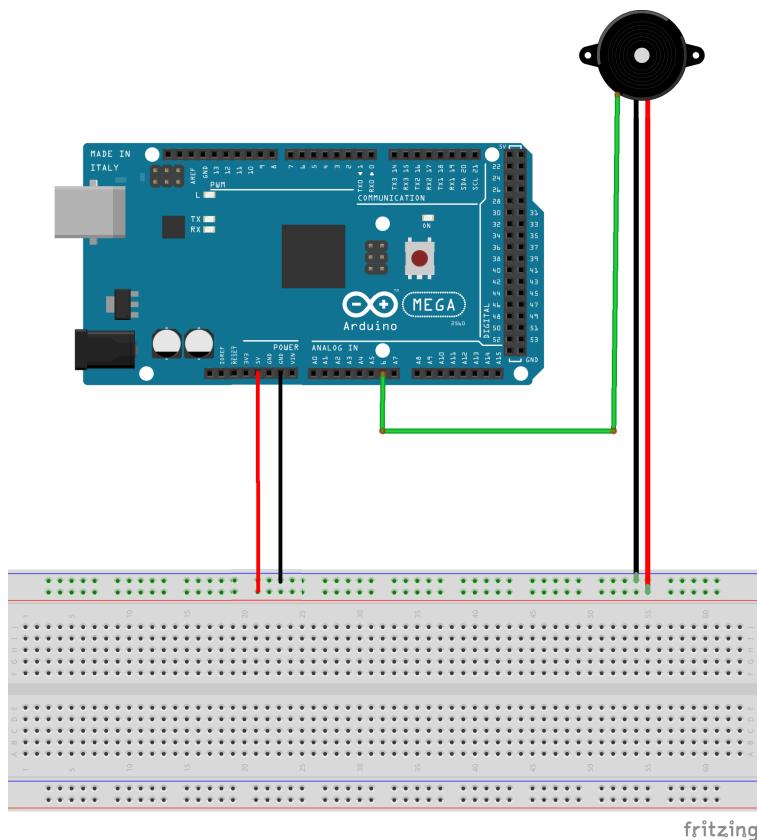


Figura 5.48: Conexionado del buzzer.



# **Capítulo 6**

## **Desarrollo software**

### **6.1. Metodología de desarrollo**

Este proyecto ha sido elaborado empleando una metodología de desarrollo basada en el modelo de desarrollo incremental para la parte software referente a todos los subsistemas que implican labores de programación, que son las referentes la programación de la placa Raspberry Pi y la programación de la placa Arduino.

El modelo de desarrollo incremental proporciona una serie de características que lo hacen idóneo para este proyecto. Dicho modelo se basa en la filosofía de construir e ir incrementando las funcionalidades del sistema mediante el desarrollo de los diferentes módulos. Esto permite ir aumentando gradualmente las capacidades del software.

Dicha metodología de desarrollo resulta especialmente útil en las siguientes situaciones:

- Facilita el desarrollo permitiendo a cada miembro del equipo desarrollar un módulo particular. En el caso del presente proyecto me ha permitido desarrollar un módulo tras otro de una manera secuencial.
- Es similar al ciclo de vida en cascada aplicándose un ciclo en cada nueva funcionalidad del programa.
- A final de cada ciclo se entrega el software al cliente. En el caso que compete a este proyecto se mantenía una reunión con el director del proyecto para su aprobación.

Centrándonos nuevamente en el desarrollo del proyecto, los motivos que llevaron a cabo la elección de un modelo de desarrollo incremental viene dada por la necesidad de simplificar e ir desarrollando de una forma gradual y modularizada debido a la extensión del proyecto. Más si cabe que el equipo de desarrollo solo consta de una persona.

Por tanto el proyecto queda distribuido en los siguientes subsistemas:

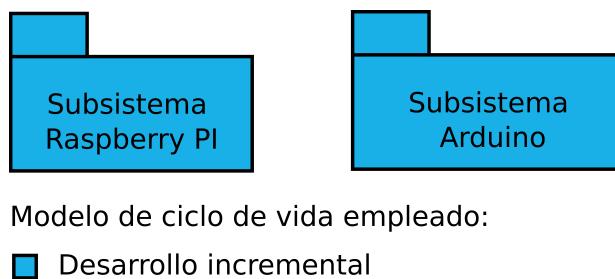


Figura 6.1: Subsistemas existentes en el proyecto junto con el modelo de ciclo de vida utilizado para su desarrollo.

Donde el subsistema Arduino abarca todas las tareas referentes a la programación del microcontrolador e interconexión con todos los sensores utilizados y el subsistema Raspberry Pi consta de todos aquellos procesos de captación de datos, procesamiento y envío al servidor de control.

#### 6.1.0.1. Diagrama de casos de uso

Una vez analizados los componentes hardware principales a utilizar, pasamos al análisis de los diferentes requerimientos funcionales para el vehículo a desarrollar.

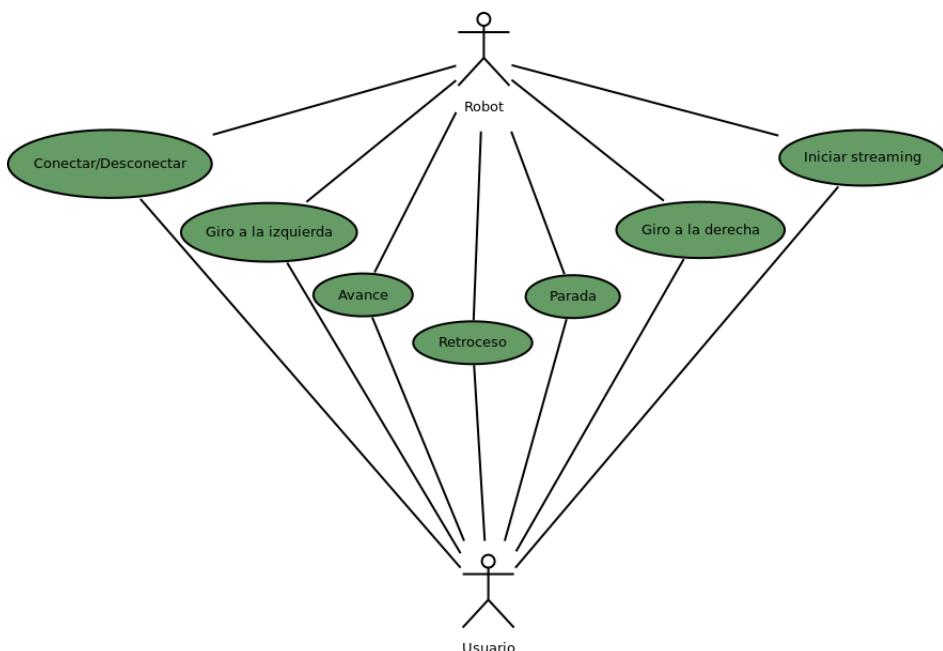


Figura 6.2: Diagrama de casos de uso para la interacción con el robot.

### 6.1.0.2. Estados del robot

Otro de los requerimientos es que el robot deberá de responder a una serie de estados de tal modo que en la aplicación se conozca las diferentes situaciones en la que se pueda encontrar un robot. En el autómata diseñado encontramos un modo desactivado, modo de espera y modo de conexión. El autómata representativo es el siguiente:

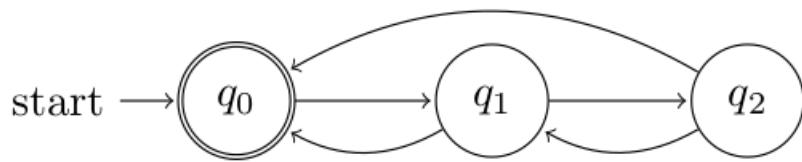


Figura 6.3: Autómata representativo de los diferentes estados del robot.

Donde:

- Estado  $q_0$  inicial y final se corresponde con el estado apagado.
- Estado  $q_1$  se corresponde con el estado en escucha.
- Estado  $q_2$  se corresponde con el estado en funcionamiento.

## 6.2. Software de control

Para la programación de la placa Raspberry Pi se ha empleado el lenguaje de programación JavaScript en un entorno de ejecución Node.js para la que se han utilizado las siguientes bibliotecas o módulos:

- *pigpio*: Módulo para la comunicación y control de los pines GPIO de la Raspberry pi.
- *child process*: El módulo *child\_process* proporciona la capacidad de generar procesos secundarios. Se ha empleado para la captura de vídeo mediante el lanzado de comandos *ffmpeg*.
- *socket.io*: Biblioteca que establece enlaces bidireccionales en tiempo real y en comunicación basada por eventos.

- *SerialPort* Módulo para la comunicación serie utilizado para establecer comunicación con la placa Arduino.

Por otra parte, para la programación de la placa Arduino se ha utilizado el lenguaje de programación *Arduino programming language* el cual está basado en C++ y aunque la referencia para el lenguaje de programación de Arduino se encuentra disponible en <http://arduino.cc/en/Reference/HomePage>, también es posible usar comandos estándar de C++ en la programación de Arduino. Las bibliotecas Arduino utilizadas han sido las siguientes:

- *TonePlayer*: Biblioteca para la activación de los zumbadores.
- *SimpleDHT*: Biblioteca para la lectura del sensor de temperatura y humedad del sensor DTH11 encontrándose la descripción en la sección [5.4.2](#).
- *NewPing*: Biblioteca para la lectura de los sensores de ultrasonido HC-SR04 cuya descripción se encuentra en la sección [5.4.1](#).

## 6.3. Comunicación interna

Para la comunicación entre elementos del dispositivo, como hemos estado comentando en diversas ocasiones, disponemos de dos módulos principales interconectados, uno de ellos es la placa Raspberry Pi, y el otro, la placa Arduino. En la presente sección se recogerán aquellos aspectos necesarios y de interés a nivel de programación para la comunicación entre elementos.

La Figura [6.4](#) corresponde con el diagrama de bloques de los diferentes canales de comunicaciones abiertos por SensorRS.

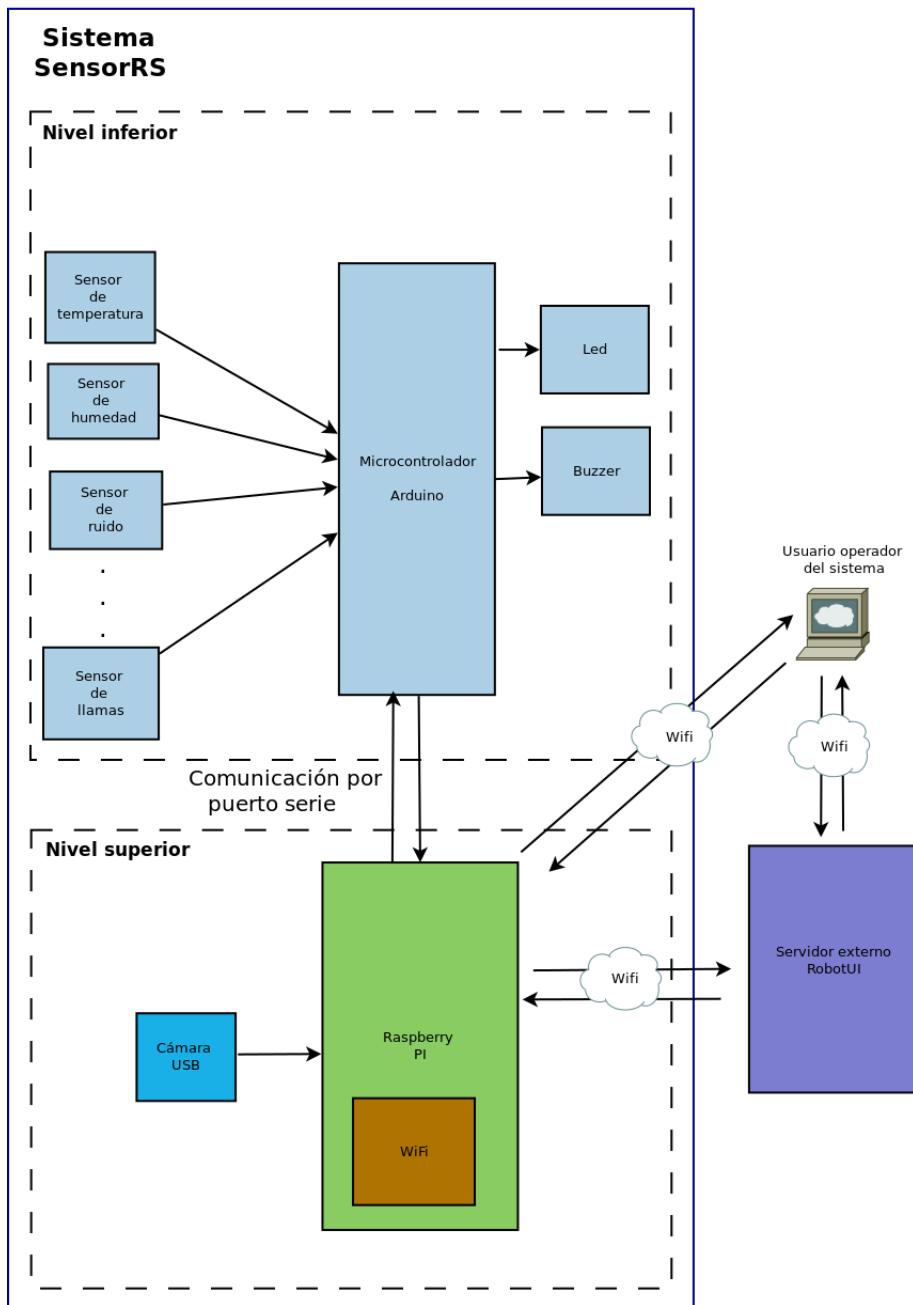


Figura 6.4: Diferentes flujos de comunicación abiertos por SensorRS.

### 6.3.1. Entrada/Salida Raspberry Pi

En la presente sección se describen los diferentes canales de comunicación de entrada/salida utilizados en la Raspberry Pi para la comunicación con la Placa Arduino o el servidor de control.

### 6.3.1.1. GPIO

En esta ocasión no ha resultado necesario utilizar los pines GPIO que dispone la Raspberry Pi puesto que disponemos de los pines de entrada/salida que proporciona el microcontrolador Arduino y han resultado suficientes.

Si, en cambio, para el vehículo que deseemos programar resultan necesarios más pines para la utilización de servos, sensores o cualquier otro elemento extra que utilice los pines GPIO de la placa Raspberry Pi, tan solo debemos inicializarlos e indicar si van a ser pines de entrada o de salida. Si existen dudas al respecto se puede acceder a la documentación de la biblioteca *pigpio* en el siguiente enlace: <https://www.npmjs.com/package/pigpio>.

Un posible ejemplo sería la inicialización de unos pines para una controladora de motores L298N que en lugar de conectarla a la placa Arduino la hubiésemos conectado directamente a la placa Raspberry Pi:

```

1 // Carga del módulo.
2 var Gpio = require('pigpio').Gpio;
3
4 // Pines utilizados. Motores izquierdos: 2 y 3, motores derechos: 17
5 // y 27
6 var gpio2 = new Gpio(2, {mode: Gpio.OUTPUT}),
7     gpio3 = new Gpio(3, {mode: Gpio.OUTPUT}),
8     gpio17 = new Gpio(17, {mode: Gpio.OUTPUT}),
9     gpio27 = new Gpio(27, {mode: Gpio.OUTPUT}),
10    gpio4 = new Gpio(4, {mode: Gpio.INPUT, alert: true});
```

Para la escritura o lectura en los puertos inicializados utilizamos las siguientes sentencias:

```

1 gpio2.digitalWrite(1); //Escritura en pin 2
2
3 gpio4.on('alert', (level, tick) => {
4     console.log('Activo!');
5 }); //Activacion del evento tras activación del pin 4
```

### 6.3.1.2. comunicación Serie

La comunicación serie mediante el uso del puerto USB se ha utilizado para la interconexión entre la placa Raspberry Pi y Arduino con la finalidad de transmitir información referente a las mediciones realizadas por los sensores y control de los mismos.

Para establecer dicha comunicación hemos utilizado el módulo JavaScript *SerialPort* resultando necesario conocer y especificar el puerto serie asignado a nuestra placa Arduino. Para ello abrimos una terminal en nuestra Raspberry Pi con la placa Arduino previamente conectada e introducimos el siguiente comando:

```
1 ls /dev | grep ttyACM
```

Una vez conocido el puerto, en nuestro caso */dev/ttyACM0*, lo emplearemos para la inicialización de la conexión con el puerto serie.

```
1 // Carga del módulo.
2 const SerialPort = require('serialport');
3 const Readline = SerialPort.parsers.Readline;
4 const port = new SerialPort('/dev/ttyACM0', {
5   baudRate: 9600
6 });
7
8 const parser = new Readline();
9 port.pipe(parser);
```

A continuación, se muestra el código implementado para el envío de datos a la placa Arduino mediante la función *serial\_transmission*:

```
1
2 function serial_transmission( data, delay ){
3   setTimeout(function() {
4     // Envío del string carácter por carácter
5     for(var i=0; i<data.length; i++){
6       port.write(new Buffer(data[i], 'ascii'));
7     }
8     // Envío del carácter final \n
9     port.write(new Buffer('\n', 'ascii'));
10   }, delay );
11 }
12 }
```

Como se observa en la función superior la transmisión de información se realiza transmitiendo la cadena deseada carácter por carácter introduciendo finalmente un carácter identificador de final de línea que, para nuestro caso, hemos empleado el ” indicativo de salto de línea.

El envío de cada carácter irá acompañado de un retraso, *delay*, el cual es introducido por parámetro a la función.

Para la captación por parte de la Raspberry Pi de los parámetros emitidos por la Placa Arduino se dispone de la siguiente función:

```
1 const parser = port.pipe(new Readline({ delimiter: '\r\n' }));
2 parser.on('data', function(data){
3   console.log(data);
4   var msg_data = data.split("%");
5   socket.emit(String(msg_data[0]), {msg: String(msg_data[1])} );
6 });
```

Dicha función se encarga de una vez capturado el mensaje, romperlo en dos secciones delimitadas por el carácter '%' siendo la parte izquierda (almacenada en `msg_data[0]`) la referente al tipo de dato recibido y la derecha (almacenada en `msg_data[1]`) el propio valor del mensaje. Por ejemplo, para el sensor de temperatura tenemos el identificador `tmp` siendo un posible mensaje resultante `tmp %22 *C.`

Finalmente se realiza el envío del dato a través del websocket al servidor de control, donde el nombre del evento se corresponde con el nombre del tipo de dato recibido con la sentencia `Socket.emit`.

### 6.3.2. Entrada/Salida Arduino

#### 6.3.2.1. Lectura puerto serie

A continuación se muestra una sección del código destinado a la recepción de un mensaje enviado desde la placa Raspberry Pi. El ejemplo mostrado recibe los comando necesarios para la activación de los motores:

```

1 #define E1 10 // Enable Pin motor 1
2 #define I1 8 // Control pin 1 motor 1
3 #define I2 9 // Control pin 2 motor 1
4
5 void setup() {
6     Serial.begin(9600); //Seteo de la tasa de transferencia
7
8     pinMode(E1, OUTPUT);
9     pinMode(I1, OUTPUT);
10    pinMode(I2, OUTPUT);
11
12    pinMode(E2, OUTPUT);
13    pinMode(D2, OUTPUT);
14    pinMode(D1, OUTPUT);
15
16    // Seteo de direccion de rotacion inicial
17    digitalWrite(I1, LOW);
18    digitalWrite(I2, HIGH);
19    digitalWrite(D1, LOW);
20    digitalWrite(D2, HIGH);
21
22 }
23
24 void loop() {
25
26
27     while (Serial.available() > 0) {
28         char received = Serial.read();
29         inData.concat(received);
30
31         if (received == '\n') {
32             inData.trim();
33

```

```

34     //MOTOR
35     if (inData.startsWith("MOTOR")){
36         if (sscanf(inData.c_str(), "MOTOR-%d", &motor) == 1) {
37             Serial.println(motor);
38         }
39
40         if( motor <= 51 && motor >= -51 ){
41             Serial.println('STOP');
42             digitalWrite(E1, LOW);
43             digitalWrite(I1, LOW);
44             digitalWrite(I2, LOW);
45         }
46
47         if( motor > 51){
48             analogWrite(E1, motor);
49             digitalWrite(I1, HIGH);
50             digitalWrite(I2, LOW);
51         }
52
53         if( motor < -51 and ! stop){
54             motor = motor * -1;
55             analogWrite(E1, motor);
56             digitalWrite(I1, LOW);
57             digitalWrite(I2, HIGH);
58         }
59     }
60     inData = "";
61 }
62 }
63
64 delay(100);
65
66 }
```

Como podemos observar en el código superior, la función consta de un bucle principal que va leyendo los diferentes caracteres que recibe concatenándolos en una variable hasta recibir el carácter utilizado como indicativo de final de mensaje '\n'. Por lo que el mensaje quedaría interceptado e interpretado como completo. El código análogo de la Raspberry Pi se localiza en la sección [6.3.1.2](#).

Centrándonos nuevamente en el ejemplo superior, se recibe la palabra clave *MOTOR*- seguida del valor de la velocidad de giro, siendo éstas positivas, avance o negativas para el retroceso en un rango que abarca entre los -255 a 255 y delimitado por '\n'. Por lo que si el mensaje cumple el patrón deseado activará las salidas correspondientes a la acción a realizar.

Otro punto a destacar es que los valores de detección del motor se encuentran en el rango situado entre -51 y 51. Ésto es debido a que a velocidades muy pequeñas el motor no adquiere la suficiente fuerza como para traccionar el vehículo evitando así consumos innecesarios y ruido ocasionado por los motores.

### 6.3.2.2. Escritura en puerto serie

A continuación se muestra un ejemplo de envío de los valores de temperatura y humedad del sensor DTH11 conectado al Arduino para su transferencia a la Placa Raspberry Pi:

```

1 #include <SimpleDHT.h>
2
3 int pinDHT11 = 4;
4 String inData = "";
5
6 SimpleDHT11 dht11;
7
8 void setup() {
9     Serial.begin(9600);
10 }
11
12 void loop() {
13     byte temperature = 0;
14     byte humidity = 0;
15     int err = SimpleDHTErrSuccess;
16     if ((err = dht11.read(pinDHT11, &temperature, &humidity, NULL)) !=
17         SimpleDHTErrSuccess) {
18         Serial.print("Read DHT11 failed, err=");
19         Serial.println(err); delay(1000);
20     }
21
22     dht11.read(pinDHT11, &temperature, &humidity, NULL);
23     if ( (int)temperature != 0 and (int)humidity != 0 ){
24         Serial.print("tmp%");
25         Serial.print((int)temperature); Serial.print(" *C, ");
26         Serial.print((int)humidity); Serial.println(" H");
27     }
28
29     delay(30);
30
31     Serial.println(analogRead(photosensorPin));
32 }
```

Como podemos observar en el código superior, el mensaje transmitido presenta el siguiente formato: tmp% xx\*C xxH, donde *tmp* representa el tipo de dato transmitido actuando el carácter '%' como delimitador entre el tipo de dato y el dato en sí mismo.

## 6.4. Comunicación externa

En el punto anterior hemos visto las diferentes configuraciones de Entrada/Salida establecidas tanto para el control de los diferentes motores y sensores en Arduino y Raspberry Pi. En el presente y sucesivos puntos describiremos los diferentes canales de

comunicación abiertos y los flujos de información existentes entre la placa Raspberry Pi y el servidor de control donde se encuentra alojada la aplicación web para monitorización del vehículo SensorRS.

En la figura 6.5 se muestra un gráfico representativo de los diferentes flujos de datos existentes:

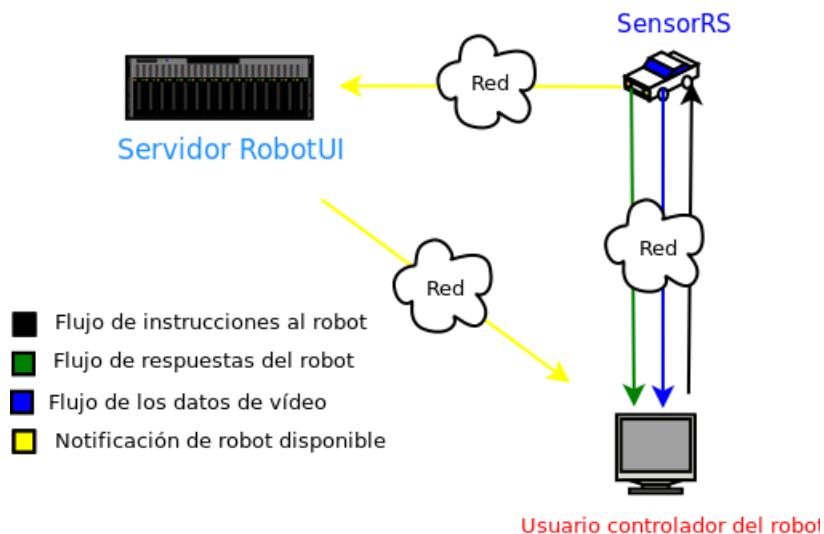


Figura 6.5: Canales de comunicación abiertos externos a SensorRS.

Primeramente al accionar el robot, éste envía una notificación al servidor indicando que se encuentra en modo *online* y permanece a la espera de que algún usuario de la aplicación decida conectarse para su control. Canal de comunicación representado en amarillo en la figura 6.5.

Un usuario ve disponible el robot y decide conectarse. Entonces se establece un enlace entre el robot, servidor, y el usuario, cliente. Flujo de comunicación representado en negro, figura 6.5.

El usuario envía comandos al robot a través del canal abierto y las respuestas son enviadas al cliente, transmisiones representados en verde para datos y en azul para el vídeo, figura 6.5.

Las tres vías de comunicación entre el robot y el usuario discurren dentro de un mismo canal de comunicación (socket). En el siguiente punto se describirá más detalladamente el procedimiento.

#### 6.4.0.1. Sockets

Ahora bien, una vez definidos los pines o comandos que derivaremos a la placa Arduino y que activarán nuestros motores o sensores y los canales de comunicación necesarios, necesitamos que éstos sean activados cuando desde una red externa lo indiquemos.

Para ello haremos uso de la biblioteca Socket.io.

Comenzamos el código incluyendo las librerías necesarias:

```
1 var io_client = require('./node_modules/socket.io-client');
2 var sails_client = require('./node_modules/sails.io.js');
```

Para la comunicaciones usuario y dispositivo robótico se ha empleado la biblioteca socket.io-client.

Para las comunicaciones directas con el servidor se ha empleado el SDK<sup>1</sup> proporcionado por el framework para comunicarse con Sails a través de sockets desde una aplicación Node.js o desde el propio navegador.

El siguiente paso una vez cargadas las librerías es establecer las diferentes conexiones. El primer comportamiento deseado es, por parte del robot, lanzar un mensaje al servidor indicando su disponibilidad al servidor con la finalidad de enviar las diferentes notificaciones a los usuarios que estén usando la aplicación. Para ello establecemos la conexión y enviamos un mensaje al servidor con el identificador del robot junto con su estado online igual a *true*. A continuación mostramos el código:

```
1 var io_server = sails_client(io_client);
2 io_server.sails.url = 'http://46.101.102.33:80';
3 io_server.socket.get('/robot/changetoonline/', {robot:
  '59188631c8e94ba54f7a4bdc', online: true});
```

Una vez realizado el paso anterior, tenemos al robot SensorRS que ha indicado a la aplicación web que se encuentra en estado online pero nada más. El siguiente paso sería establecer alguna comunicación que se mantuviera a la escucha a la espera de la llegada de nuevas conexiones. Para la creación del socket basta con la siguiente instrucción, la cual recibe un puerto que utilizará para mantenerse a la escucha:

```
1 var io = require('./node_modules/socket.io').listen(8085, { log:
  false });
```

Con la finalidad de ir capturando los diferentes eventos, se han definido las siguientes funciones para la conexión y desconexión de los clientes:

```
1
2 io.sockets.on('connection', function (socket)
3 {
4
5   //Almacenamiento del número total de clientes conectados.
6   sockets[socket.id] = socket;
```

<sup>1</sup>Un kit de desarrollo o SDK (siglas en inglés de software development kit) es generalmente un conjunto de herramientas de desarrollo de software que le permite al programador o desarrollador de software crear aplicaciones para un sistema concreto, por ejemplo ciertos paquetes de software, frameworks, etc.

```

7   console.log("Total clientes conectados : ",
8     Object.keys(sockets).length);
9
10 // Envío de un saludo.
11 socket.emit('robotmsg', {msg: "!!!HOLA!!!"});
12
13 //Salida de un cliente.
14 socket.on('disconnect', function() {
15   console.log('Bye!');
16   stopStreaming(socket);
17 });
18
19 }

```

Cuando un evento *action* es recibido se activada la función que procesa el comando recibido y activa las salidas correspondientes, la cual establece los pines necesarios a los valores 1 o 0 según el parámetro establecido.

#### 6.4.0.2. Streaming de vídeo

Para realizar la transferencia de vídeo desde el robot hacia el cliente se ha empleado la librería FFmpeg<sup>2</sup> haciendo uso de su herramienta de línea de comandos.

El procedimiento de captura de vídeo y su posterior transmisión es realizado mediante la siguiente instrucción de Ffmpeg:

```

1 ffmpeg -f video4linux2 -i /dev/video0 -s 300x150 -f mjpeg pipe:1
      -b:v 28k -bufsize 28k

```

En los puntos sucesivos analizaremos qué es lo que realiza la instrucción anterior y por qué resulta clave en todo el proceso de difusión, comprendido desde la captura del vídeo hasta su posterior transmisión al usuario que está controlando el robot.

Nada prodríamos transmitir si no disponemos inicialmente de los datos que queremos difundir. De ahí que inicialmente debamos realizar la captura de las diferentes imágenes a partir de la cámara USB conectada a la Raspberry Pi. Para ello se utiliza la API de captura de vídeo video4linux2<sup>3</sup> (o simplemente v4l2) la cual dispone las bibliotecas de Ffmpeg. Tan solo debemos especificar el dispositivo de captura.

El nombre del dispositivo de captura es un nodo de dispositivo de archivo, por lo general los sistemas Linux tienden a crear automáticamente estos nodos cuando el dispositi-

<sup>2</sup> Los conocimientos necesarios para la comprensión de la herramienta FFmpeg y sus modos de utilización se han adquirido accediendo a la documentación disponible en la referencia [?] correspondiente con la documentación oficial.

<sup>3</sup> Video4Linux o V4L es una API de captura de video para Linux. Muchas webcams USB, sintonizadoras de tv, y otros periféricos son soportados. Video4Linux está integrado con el núcleo Linux. V4L está en su segunda versión (V4L2). El V4L original fue incluido en el ciclo 2.1.X de desarrollo del núcleo Linux. Video4Linux2 arregla algunos fallos y apareció en los núcleos 2.5.X.

tivo está conectado al sistema, y tiene un nombre del tipo /dev/videoN, donde N es un número asociado al dispositivo.

Para proceder a la captura de las imágenes nos bastaría con introducir el siguiente comando:

```
1 ffmpeg -f video4linux2 -i /dev/video0 -s 300x150 -f mjpeg
          video_out.mpeg
```

Ahora bien, el comando anterior toma las imágenes de la cámara y las almacena en el archivo especificado *video\_out.mpeg* especificando una resolución de salida de 300x150 píxeles empleando la opción *-s*. Pero nosotros no deseamos exactamente ese comportamiento. Debemos canalizar esos datos capturados hacia el socket creado con la finalidad de ir transmitiendo los diferentes frames y no almacenándolos en disco tal y como realiza la instrucción anterior.

Para resolver este problema podemos emplear el sistema de tuberías que implementan los sistemas UNIX <sup>4</sup>.

En cualquier sistema Unix se puede hacer que la salida de una determinada orden sea la entrada estándar de otra, lo que le confiere a las órdenes Unix una enorme potencia. Para realizar dicha “canalización” debemos utilizar las siguientes opciones:

```
1 pipe:1 -b:v 28k -bufsize 28k
```

Con la opción *pipe:1* accedemos al protocolo pipe de UNIX, el cual lee y escribe de las *tuberías* UNIX siendo el número 1 la tubería correspondiente a la salida estándar *stdout* (0 para *stdin* y 2 para *stderr*), la cual podría ser omitida puesto que es la salida por defecto.

La opción *-b:v 28k* establece la tasa de transferencia, en nuestro caso una tasa de 28 kbit/s.

La opción *-bufsize 28k* establece un tamaño de buffer <sup>5</sup> de 28 kbits.

A continuación mostramos el código de transmisión de vídeo al completo junto con la captura de los diferentes eventos activados cuando se produce la salida de datos por cada una de las salidas estándar:

```
1
2   function startStreaming(socket) {
```

<sup>4</sup> Una tubería (pipe, cauce o '|') consiste en una cadena de procesos conectados de forma tal que la salida de cada elemento de la cadena es la entrada del próximo. Permiten la comunicación y sincronización entre procesos. Es común el uso de buffer de datos entre elementos consecutivos.

<sup>5</sup> Un buffer de datos es un espacio de la memoria en un disco o en un instrumento digital reservado para el almacenamiento temporal de información digital, mientras que está esperando ser procesada.

```

3 //ffmpeg -f video4linux2 -i /dev/video0 -s 300x150 -f mjpeg pipe:1
4   -b:v 28k -bufsize 28k
5
6 if (running_camera == false){
7   console.log('Starting streaming....');
8   var args = ["-f", "video4linux2", "-i", "/dev/video0", "-s",
9     "300x150","-f","mjpeg", "pipe:1", "-b:v 28k", "-bufsize 28k"]
10  ffmpeg_command = require('child_process').spawn("ffmpeg", args);
11  running_camera = true
12 }
13
14 ffmpeg_command.on('error', function(err, stdout, stderr) {
15   console.log("ffmpeg stdout:\n" + stdout);
16   console.log("ffmpeg stderr:\n" + stderr);
17   running_camera = false
18 });
19
20 ffmpeg_command.on('close', function (code) {
21   console.log('ffmpeg exited' + code );
22   running_camera = false
23 });
24
25 ffmpeg_command.stderr.on('data', function (data) {
26   //console.log('stderr: ' + data);
27 });
28
29 ffmpeg_command.on('end', function() {
30   console.log('Finished');
31   running_camera = false
32 });
33
34 ffmpeg_command.stdout.on('data', function (data) {
35   //console.log('stdout: ' + data);
36   var frame = new Buffer(data).toString('base64');
37   socket.emit('canvas',frame);
38 });
39 }
```

#### 6.4.0.3. Código de ejemplo completo

Finalmente se muestra el código completo para el robot SensorRS desarrollado. Dicho código puede emplearse como guía de referencia o plantilla para futuros proyectos con la idea de integrarlos en la aplicación RobotUI.

```

1 // Serial
2 const SerialPort = require('serialport');
3 const Readline = SerialPort.parsers.Readline;
4 const port = new SerialPort('/dev/ttyACM0', {
5   baudRate: 9600
6 });
7 }
```

```
8 const parser = new Readline();
9 port.pipe(parser);
10
11
12 var io_client = require('../node_modules/socket.io-client');
13 var sails_client = require('../node_modules/sails.io.js');
14 var io_server = sails_client(io_client);
15 // io_server.sails.url = 'http://192.168.1.135:1337';
16 io_server.sails.url = 'http://localhost:1337';
17 io_server.socket.get('/robot/changetoonline/', {robot:
18   '5b64917f7510da0d68a140ec', online: true});
19
20 // Inicia servidor socket.io en el puerto 8085.
21 var io = require('../node_modules/socket.io').listen(8085, { log:
22   false });
23
24 // Inicia servidor socket.io para video en el puerto 3535.
25 var io_video = require('../node_modules/socket.io').listen(3535, {
26   log: false });
27
28 // Carga de módulos necesarios.
29 var ffmpeg_command, running_camera = false, child_process =
30   require('child_process');
31
32 console.log('Esperando conexión...');

33 var sockets = {};
34
35 // Servo variables
36 var y, servo_previous_status = 0;
37
38 // Motor variables
39 var x, motor_previous_status = 0;
40
41
42 io.sockets.on('connection', function (socket)
43 {
44
45   sockets[socket.id] = socket;
46   console.log("Clientes totales conectados: ",
47     Object.keys(sockets).length);
48
49
50   socket.on('disconnect', function() {
51     console.log('Adios !');
52     //stopStreaming(socket);
53   });
54
55   socket.on('start-stream', function() {
56     startStreaming(socket);
57   });
58
59   socket.emit('robotmsg', {msg: "Bienvenido !!!"});
60   console.log('emitiendo: ' + "Bienvenido !!!");
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
```

```
58  socket.on('axes_action_a', function (data) {
59
60    if(data['y'] > 0){ //derecha
61      y = data['y'] * -2 * 100 - 50;
62    } else { //izquierda
63      y = data['y'] * -2 * 100 + 50;
64    }
65
66    if(y >= 49 && y <= 52 ){
67      y = 0;
68    }
69
70    y = Math.round(y);
71    var difference = Math.abs(servo_previous_status - y);
72    if( difference > 10){
73      servo_previous_status = y;
74      serial_transmission('SRV-' + y, 10 );
75      console.log(y);
76    }
77
78  });
79
80  socket.on('axes_action_b', function (data) {
81    if(data['y'] > 0){ //atrás
82      y = data['y'] * -2 * 100 - 50;
83    } else { //delante
84      y = data['y'] * -2 * 100 + 50;
85    }
86    if(y >= 49 && y <= 52 ){
87      y = 0;
88    }
89
90    if(y == -51){
91      y = 0;
92    }
93
94    y = Math.round(y);
95    var speed_difference = Math.abs(motor_previous_status - y);
96    if( speed_difference > 10){
97      motor_previous_status = y;
98      serial_transmission('MOTOR-' + y, 10 );
99      console.log(y);
100   }
101 });
102
103
104 socket.on('pad_action', function (pad_btn) {
105   console.log(pad_btn);
106   if(pad_btn == 2){
107     serial_transmission('H', 10);
108   }
109   if(pad_btn == 1){
110     serial_transmission('L', 10);
111   }
112});
```

```

113
114
115     socket.on('action', function (data){
116         console.log('Comando recibido: ' + data);
117         switch(data) {
118             case "H":
119                 serial_transmission(data, 10);
120             case "L":
121                 serial_transmission(data, 10);
122             case "UP":
123                 serial_transmission('MOTOR -255', 10);
124             case "DOWN":
125                 serial_transmission('MOTOR --255', 10);
126             case "LEFT":
127                 serial_transmission('SRV -255', 10);
128             case "RIGHT":
129                 serial_transmission('SRV --255', 10);
130             case "STOP":
131                 serial_transmission('MOTOR -0', 10);
132         }
133     });
134
135     // const parser = port.pipe(new Readline({ delimiter: '\r\n' }));
136     // parser.on('data', console.log);
137
138     const parser = port.pipe(new Readline({ delimiter: '\r\n' }));
139     parser.on('data', function(data){
140         console.log(data);
141         var msg_data = data.split("%");
142         socket.emit(String(msg_data[0]), {msg: String(msg_data[1])} );
143     });
144 });
145
146 io_video.sockets.on('connection', function (socket) {
147     console.log('Video socket oppened');
148
149     socket.on('disconnect', function() {
150         console.log(' Video desconectado!');
151         //stopStreaming(socket);
152     });
153
154     socket.on('video_canvas', function() {
155         startStreaming(socket);
156     });
157
158 });
159
160 function stopStreaming(socket) {
161     delete sockets[socket.id];
162     // no more sockets, kill the stream
163     if (Object.keys(sockets).length == 0) {
164         if (ffmpeg_command){
165             ffmpeg_command.kill();
166             running_camera = false;
167             console.log('Stop streaming');

```

```

168     }
169   }
170 }
171
172 function startStreaming(socket) {
173   // video
174   // ffmpeg -f video4linux2 -i /dev/video0 -s 300x150 -f mjpeg pipe:1
175   // -b:v 28k -bufsize 28k
176
177   if (running_camera == false){
178     console.log('Starting streaming....');
179     var args = [ "-f", "video4linux2", "-i", "/dev/video0", "-s",
180                 "300x150", "-f", "mjpeg", "pipe:1", "-b:v 16k", "-bufsize 28k"];
181
182     ffmpeg_command = child_process.spawn("ffmpeg", args);
183     running_camera = true
184   }
185
186   ffmpeg_command.on('error', function(err, stdout, stderr) {
187     console.log("ffmpeg stdout:\n" + stdout);
188     console.log("ffmpeg stderr:\n" + stderr);
189     running_camera = false
190   });
191
192   ffmpeg_command.on('close', function (code) {
193     console.log('ffmpeg exited' + code );
194     running_camera = false
195   });
196
197   ffmpeg_command.stderr.on('data', function (data) {
198     //console.log('stderr: ' + data);
199   });
200
201   ffmpeg_command.on('end', function() {
202     console.log('Fin');
203     running_camera = false
204   });
205
206   ffmpeg_command.stdout.on('data', function (data) {
207     // console.log('stdout: ' + data);
208     var frame = new Buffer(data).toString('base64');
209     socket.emit('video_canvas',frame);
210   });
211
212 }
213
214 function serial_transmission( data, delay ){
215   setTimeout(function() {
216     // Sending String character by character
217     for(var i=0; i<data.length; i++){
218       port.write(new Buffer(data[i], 'ascii'));
219     }
220     // Sending the terminate character

```

```
221     port.write(new Buffer('\n', 'ascii'));  
222 }, delay );  
224  
225 }
```

Para la ejecución del código introducimos el siguiente comando:

```
1 sudo node raspberry.js
```

Siendo *raspberry.js* el nombre del archivo que contiene nuestro código.

# **Capítulo 7**

## **Pruebas**

La realización de pruebas de software, es una de las fases del ciclo del software de gran importancia, la cual permite la identificación de posibles fallos en la implementación, calidad o usabilidad de la aplicación con la finalidad de que el producto final cumpla una serie de garantías y sea de calidad.

### **7.1. Plan de pruebas**

Debido a la arquitectura de SensorRS y a la metodología basada en desarrollo incremental que se ha seguido en el desarrollo y construcción del proyecto robótico, se ha establecido un plan de pruebas en el que las diferentes partes han ido siendo analizadas y testeadas de forma independiente. La aplicación y el robot ha sido sometido a una batería de pruebas.

Una vez finalizado el proceso de montaje y control, las pruebas que se realizaron para comprobar el vehículo cumplía con los requisitos propuestos en su diseño fueron las siguientes:

#### **Pruebas de funcionamiento:**

- Se controla que los diferentes elementos y componentes electrónicos se encuentran correctamente alimentados y con un conexiónado lo suficientemente fuerte y estable para que no de lugar a posibles fallos tras someter al vehículo a los movimientos ocasionados por su uso.

#### **Pruebas de comunicaciones:**

- Se comprueba que se realiza una comunicación correcta entre la placa Arduino - Raspberry Pi y entre Raspberry PI - Servidor sometiendo al sistema a situaciones de tráfico elevado y analizando su comportamiento.

#### **Pruebas de control:**

- Se comprueba que se realiza un manejo adecuado del dispositivo robótico.

- Se comprueba que se obtiene y muestra correctamente las diferentes mediciones obtenidas de los sensores comparando los resultados medidos con sensores externos al vehículo o comparando las distancias obtenidas con la realización de mediciones con una cinta métrica.

**Pruebas de alcance:**

- Se somete al vehículo a una prueba de alcance bordeando de la zona de cobertura abarcada por el punto de acceso wifi en el que el vehículo se encuentra conectado obteniendo un resultado satisfactorio en distancias superiores a los 15 metros.

**Pruebas de pista:**

- Se somete al vehículo a una prueba de movimientos por diversos tipos de pista, asfalto, hormigón tierra comprobando que se desenvuelve con soltura a todas las situaciones mencionadas.



(a) Pista de tierra.



(b) Pista de asfalto.



(c) Pista de hormigón.

Figura 7.1: Vehículo SensorRS en funcionamiento por diferentes tipos de pista.

# Capítulo 8

## Organización temporal

La planificación general del proyecto siguiendo un modelo SCRUM, empleando para ello el panel de tareas Trello; un gestor de proyectos que permite aplicar una metodología de desarrollo ágil.

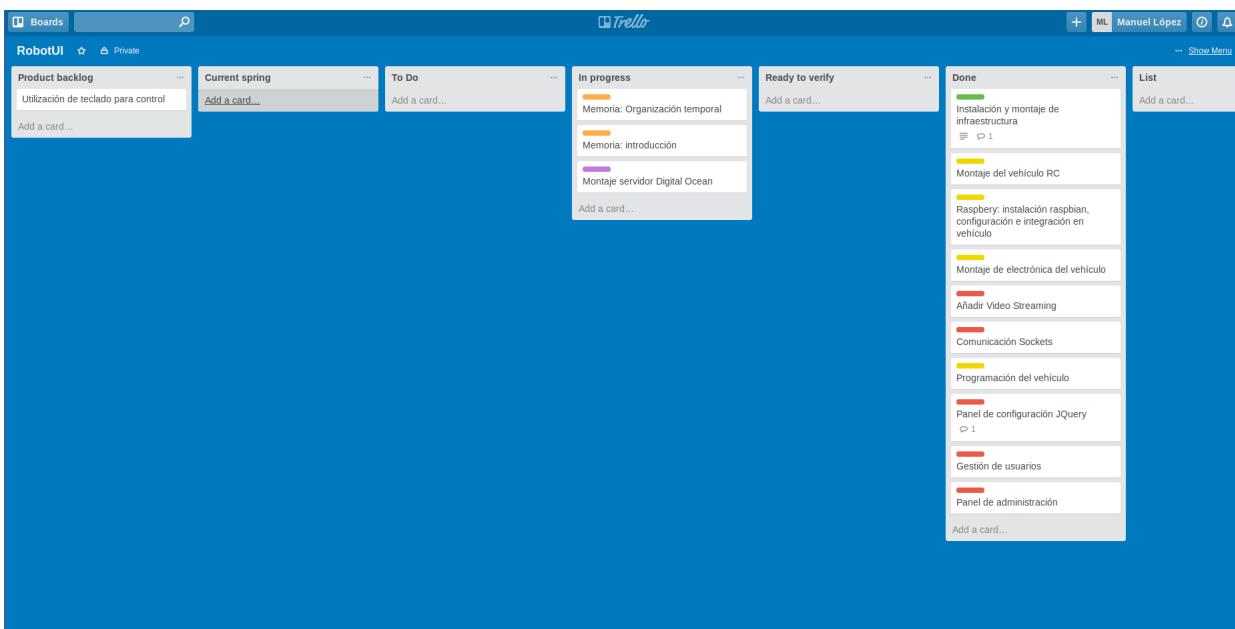


Figura 8.1: Panel de actividades - Trello

Cabe destacar que para el desarrollo de SensorRS ha sido necesario emplear varias herramientas, utilidades y bibliotecas. Algunas de ellas ya habían sido utilizadas en ciertas ocasiones, bien sea en el ámbito estudiantil o profesional. Sin embargo, otras han requerido un periodo de formación previo, en el que se han adquirido los conocimientos necesarios para poder desarrollar el presente proyecto.

La mayor parte del proceso de investigación fue dedicado al estudio de las diferentes tecnologías referentes a la programación de microcontroladores e interconexión de elementos electrónicos. Todo ello ha implicado un esfuerzo bastante considerable en el uso, aprendizaje e investigación de las diferentes tecnologías existentes y comprobar su

potencial.

Una vez determinadas las diferentes herramientas a utilizar se comenzó con la implementación comenzando por la programación de ejemplos básicos en Arduino.

La figura 8.2 muestra una visión de las diferentes tareas desarrolladas para la elaboración del proyecto junto con la descomposición de cada una de ellas:

Los puntos más importantes del proyecto se han dividido en hitos, así como entregas que se definieron en cada reunión que se realizaba con el director del proyecto. También se ha definido una planificación temporal del desarrollo del proyecto mediante un diagrama de Gantt con la duración de las tareas recogidas en el panel de actividades de Trello.

## CAPÍTULO 8. ORGANIZACIÓN TEMPORAL

---

WBS	Nombre	Trabajo
1	<b>▼ SensorRS</b>	<b>89d 3h</b>
1.1	Conocimiento del proyecto	5d
1.2	Planificación y estudio del proyecto	10d
1.3	<b>▼ Análisis de herramientas existentes</b>	<b>15d</b>
1.3.1	Estudio de Node.js	5d
1.3.2	Estudio de Arduino.js	5d
1.3.3	Estudio de Socket.io	2d
1.3.4	Códigos y pruebas	3d
1.4	<b>▼ Definición de requisitos</b>	<b>6d</b>
1.4.1	<b>▼ Requisitos no funcionales</b>	<b>3d</b>
1.4.1.1	Software	1d
1.4.1.2	Hardware	2d
1.4.2	<b>▼ Requisitos funcionales</b>	<b>2d</b>
1.4.2.1	Software	1d
1.4.2.2	Hardware	1d
1.4.3	Reunión de planificación con director del proyecto	1d
1.5	<b>▼ Montaje del vehículo</b>	<b>25d 3h</b>
1.5.1	Instalación y preparación placa Raspberry Pi	2d
1.5.2	Comunicación serie Raspberry Pi - Arduino	2d
1.5.3	Conexión de sensores	3d
1.5.4	Montaje sobre el chasis del vehículo	3d
1.5.5	Alimentación del conjunto	2d
1.5.6	Reunión de seguimiento con director del proyecto	3d 3h
1.5.7	<b>▼ Programación del vehículo</b>	<b>10d</b>
1.5.7.1	Placa Raspberry Pi	5d
1.5.7.2	Programación Arduino	5d
1.6	<b>▼ Mejoras en la aplicación RobotUI</b>	<b>4d</b>
1.6.1	Uso del gamepad	2d
1.6.2	Renovación de la interfaz	1d
1.6.3	Mejoras en comunicaciones	1d
1.7	<b>▼ Documentación</b>	<b>23d</b>
1.7.1	Memoria	19d
1.7.2	Resumen	2d
1.7.3	Presentación	2d
1.8	Reunión de seguimiento con el director del proyecto	1d

Figura 8.2: Descomposición de las tareas implicadas en la construcción del vehículo robótico SensorRS (Primera Parte).

## 8.1. Planificación temporal de tareas

A continuación se definirán los diferentes hitos que componen el diagrama de Gantt divididos en las diferentes subtareas principales de cada uno de ellos:

### 8.1.1. Hito 1: Planificación y análisis

En esta primera etapa de desarrollo del proyecto final de carrera se realizaron los estudios previos necesarios para abordar cualquier proyecto de cierta envergadura.

Este hito se descompone en las siguientes tareas principales:

1. Planificación y estudio del proyecto. En esta fase se centró en la elaboración de un documento, a modo borrador, con la idea a desarrollar, objetivos del proyecto y su alcance.
2. Análisis de herramientas existentes, de las tecnologías a implementar, la arquitectura del sistema, las tecnologías de BD, visualización, para la selección de las herramientas más adecuadas para afrontar el desarrollo con garantías y no sea necesaria una “vuelta atrás” por necesidad imperiosa de cambio de tecnología. En definitiva se buscaba una herramienta libre, con un respaldo de una comunidad importante y que resuelva la problemática o necesidad de trabajar con sensores y transmisión de datos.

### 8.1.2. Hito 2: Definición de requisitos

Este segundo hito queda dividido en las siguientes etapas:

1. Elaboración de un documento formal con la propuesta de proyecto definiendo sus objetivos y alcance, para la aprobación por parte del director del proyecto.
2. Se definen las las funcionalidades que debe cumplir el sistema, elementos electrónicos y software a utilizar, definición de requisitos funcionales y no funcionales.

### 8.1.3. Hito 3: Montaje del vehículo

En este segundo hito, uno de los de mayor magnitud, se comienza con el montaje e interconexión de los elementos hardware del vehículo, el cual queda dividido en las siguientes subtareas:

1. Instalación del sistema operativo y software necesario en la placa Raspberry Pi.
2. Conexión por puerto serie de la placa Arduino y Raspberry pi.
3. Interconexión de sensores.

4. Montaje de los elementos sobre el chasis del vehículo.
5. Alimentación de todo el conjunto.

Se realiza la construcción y montaje e instalación software del vehículo robótico y comprobación de las diferentes conexiones.

#### **8.1.4. Hito 4: Programación del vehículo SensorRS**

Este tercer hito comprende toda la programación lógica del vehículo dividiéndose principalmente en dos partes:

1. Programación de la placa Raspberry Pi, código en Node.js para la comunicación con el servidor de control, captura de valores de la placa Arduino y transmisión de datos.
2. Programación de la placa Arduino.

#### **8.1.5. Hito 5: Mejoras en la aplicación de control RobotUI**

Debido a nuevas necesidades del proyecto y diversos aspectos mejorables de la aplicación uno de los hitos del presente proyecto ha sido dedicado a la realización de estas tareas entre las que destacan:

1. Incorporación de la posibilidad de utilización de un Gamepad para el control de los diferentes dispositivos robóticos.
2. Renovación de la interfaz incluyendo la última versión del popular framework Bootstrap para Html, Css y JavaScript.
3. Mejoras en el apartado de comunicaciones estableciendo un canal específico para la transmisión de vídeo y otro para el de comando y valores de los sensores.

#### **8.1.6. Hito 6: Documentación**

Se finaliza la memoria para la revisión por parte del director del proyecto y su posterior impresión. Se prepara la presentación para la defensa ante tribunal.

### **8.2. Diagrama de Gantt**

A continuación se muestra el diagrama de Gantt donde quedan reflejados los diferentes hitos descritos en el punto anterior.

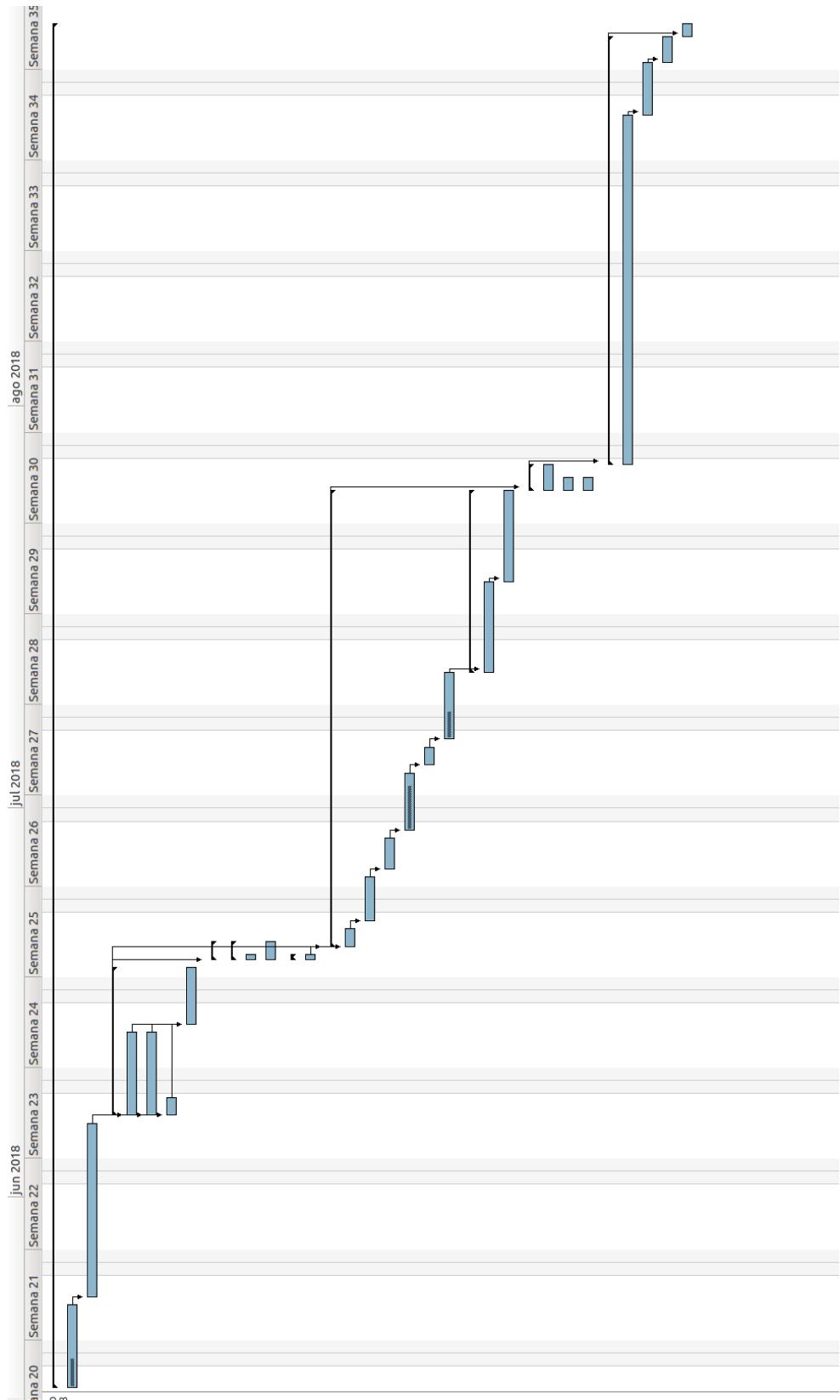


Figura 8.3: Diagrama de Gantt. Desarrollo del proyecto.

# **Capítulo 9**

## **Guía de Usuario**

La aplicación SensorRS ha sido creada por Manuel López Urbina para el trabajo fin de máster titulado Máster Universitario en Investigación en Ingeniería de Sistemas y de la Computación de la Universidad de Cádiz.

Resulta de vital importancia consultar esta guía antes y/o durante la utilización de los diferentes elementos tanto hardware pertenecientes al vehículo robótico, como software del presente proyecto ya que le proporcionará una guía paso a paso en el manejo correcto del sistema completo.

SensorRS ha sido elaborado con un claro propósito; el de proporcionar a los usuarios un medio donde realizar exploraciones de forma remota obteniendo información allá por donde se mueva el vehículo robótico ya sea por mero entretenimiento o por evitar la exposición a zonas peligrosas o de difícil acceso. Además de compartir dichos datos obtenidos, mediciones e imágenes, con el resto de usuarios. Esto es posible gracias a una serie de herramientas desarrolladas para que, sin necesidad de tener grandes conocimientos en programación, puedan configurar un entorno para el manejo de sus proyectos robóticos y tener posibilidad de compartir sus dispositivos y experiencias con el resto de usuarios.

La particularidad de RobotUI es que el usuario propietario del robot tiene la posibilidad de permitir el manejo de sus dispositivos robóticos al resto de usuarios que él mismo considere de una manera controlada o, por otra parte, permitir que otros usuarios visualicen, como si de espectadores se tratase, el control que un determinado usuario realiza de un determinado robot. Todo ello en tiempo real.

Por tanto, tras esta breve introducción en el ámbito del sistema, en este manual se describen los diferentes pasos a realizar para configurar correctamente el dispositivo y abrirlo a toda una comunidad de usuarios.

### 9.0.1. Objetivo de esta guía

Esta guía tiene como objetivo proporcionar al usuario un soporte de ayuda e iniciación a la utilización de SensorRS dentro de la aplicación web de control RobotUI.

Esta sección comprende:

- Introducción.
- Guía de acceso al código fuente de la aplicación.
- Guía de uso del sistema.
- Guía para la puesta en marcha y su utilización.

### 9.0.2. Dirigido a

Esta guía esta dirigida al usuario final del proyecto SensorRS. Tiene la finalidad de proporcionar una guía descriptiva de los procedimientos de creación, configuración y utilización del dispositivo robótico en sus dos modalidades disponibles, la de control y la de visualización dentro de la aplicación web de control.

### 9.0.3. Obtener SensorRS

El código fuente junto con la presente memoria se encuentra disponible en el repositorio GitHub en el enlace <https://github.com/lopomaster/SensorRS> o usando la herramienta Git, escribiendo en la consola el siguiente comando:

```
1 git clone git@github.com:lopomaster/SensorRS.git
```

## 9.1. Uso de SensorRS

### 9.1.1. Configuración

Para la puesta en funcionamiento es necesario establecer una series de configuraciones en el código del robot con la finalidad de conectarlo a la aplicación RobotUI.

El manual de usuario de RobotUI se encuentra accesible en el repositorio del proyecto localizado en <https://github.com/lopomaster/SAILS-RobotUI>. Para acceder a la aplicación, el usuario deberá acceder al siguiente enlace: [www.robotui.es](http://www.robotui.es).

Al acceder podrá ver el portal de entrada a la aplicación. En él puede acceder al resto de funcionalidades identificándose con sus credenciales y acceder a los formularios de registro de usuario.

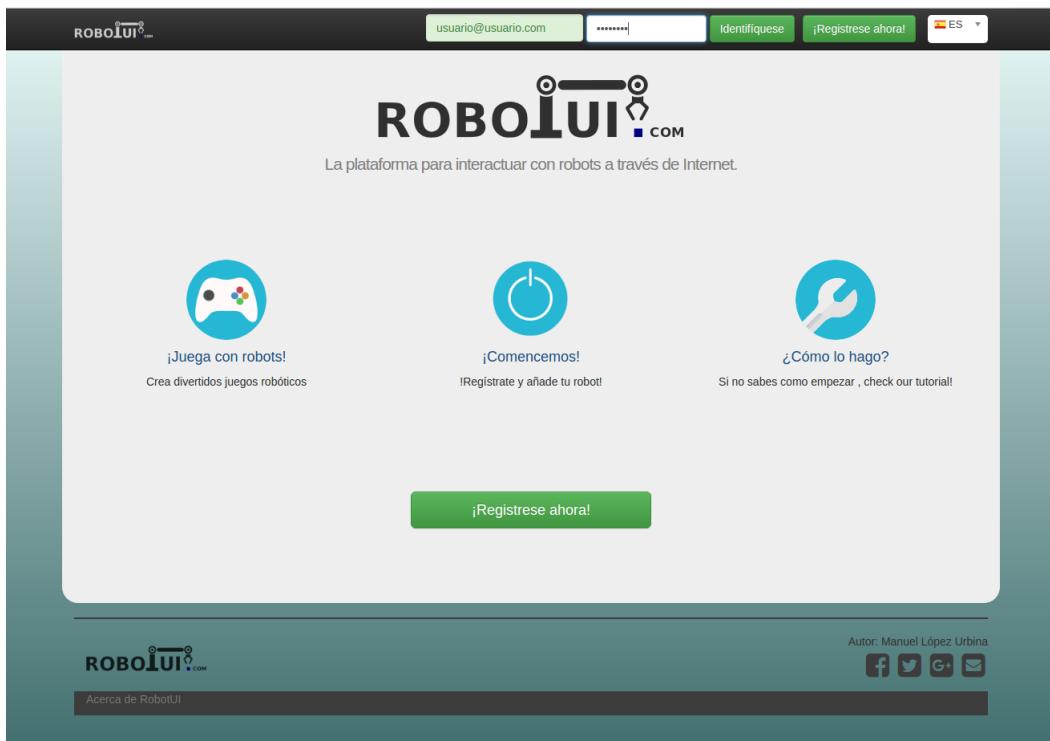


Figura 9.1: Página principal RobotUI.

### 9.1.2. Programa tu robot

**NOTA:** Esta guía comprende una serie de pasos para realizar la programación de un dispositivo robótico para la aplicación RobotUI. En este caso particular, el robot empleado posee como base una placa Raspberry Pi y una placa Arduino, la cual emplea para la conexión de sensores y motores haciendo uso de su sistema de Entrada/Salida programable con la ayuda del microcontrolador incorporado.

El sistema es compatible con cualquier dispositivo, no solo limitado a robots gestionados por placas Raspberry. La única diferencia radica en que se deberán emplear a nivel de programación las bibliotecas adecuadas para la activación o lectura de las Entradas/Salidas correspondientes al modelo de placa utilizado.

Asumido el paso de programación de nuestro robot, para más información puede acceder a la guía de usuario de RobotUI. Se procede al alta del robot en el sistema. Importante tomar nota del identificador único que la aplicación proporciona para dicho robot y que posteriormente necesitaremos.

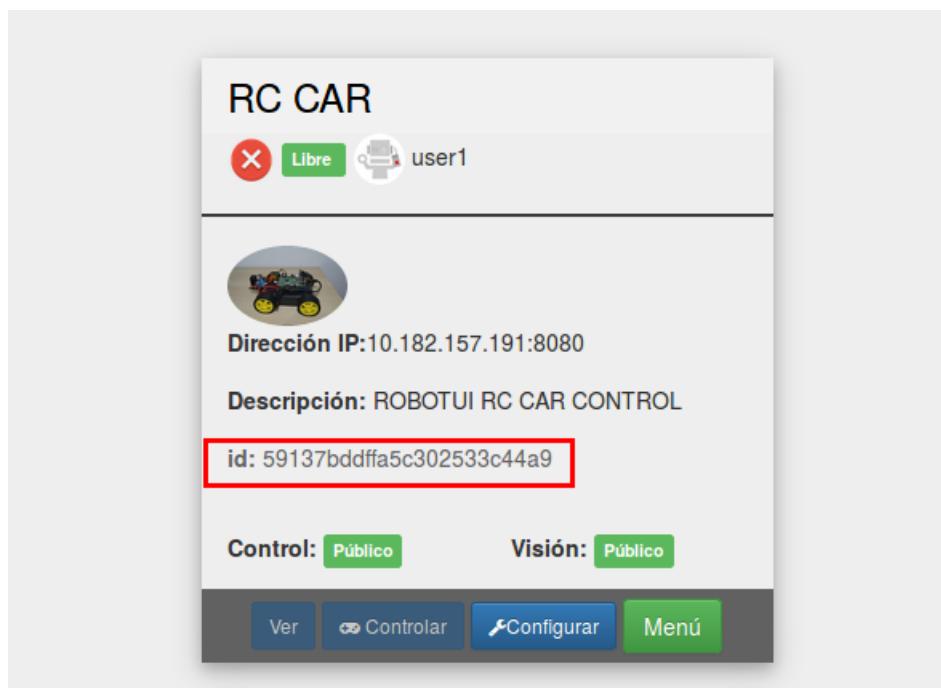


Figura 9.2: Panel informativo donde se aprecia el identificador del robot SensorRS en la aplicación.

Una vez creado el robot y configurada su interfaz, añadiendo los diferentes elementos disponibles como acciones, vídeo, etiquetas, etcétera, procedimiento recogido en la documentación de la aplicación RobotUI en la sección Configuración de la interfaz. Podemos comenzar con la programación del robot.

Para ello debemos seguir una serie de pasos:

1. Generar una archivo de extensión .js, con el código base mostrado en el punto [9.1.2.1](#).
2. Reemplazar en el código la palabra *IDENTIFICADOR* por el identificador único de nuestro robot obtenido en la vista informativa del mismo. Por ejemplo: 59188631c8e94ba54f7a4bdc.
3. Indicar el puerto en el que el robot permanecerá a la escucha. Para ello debemos reemplazar palabra *PUERTO* del código inferior por el puerto deseado. Por ejemplo 8085.
4. El siguiente paso es determinar qué puertos GPIO necesitaremos y si van a ser utilizado en modo Entrada, Salida o Entrada/Salida dentro de la sección *PINES* de la plantilla. En este caso se ha empleando la biblioteca *pigpio* cuya documentación se encuentra disponible en el siguiente enlace: <https://www.npmjs.com/package/pigpio>.
5. Definir el funcionamiento de los diferentes eventos dentro de la sección *EVENTOS*.

### 9.1.2.1. Código base

```

1  var io_client = require('./node_modules/socket.io-client');
2  var sails_client = require('./node_modules/sails.io.js');
3  var io_server = sails_client(io_client);
4  io_server.sails.url = 'http://46.101.102.33:80';
5  io_server.socket.get('/robot/changetoonline/', {robot:
6      'IDENTIFICADOR', online: true});
7
8 // Inicia servidor socket.io en el puerto PUERTO.
9 var io =io_client.listen( PUERTO, { log: false });
10
11 var ffmpeg_command, running_camera = false, child_process =
12     require('child_process');
13
14 var Gpio = require('pigpio').Gpio;
15
16 // SECCION PINES
17
18 // EJEMPLO:
19 //     var gpio2 = new Gpio(2, {mode: Gpio.OUTPUT}),
20 //         gpio3 = new Gpio(3, {mode: Gpio.OUTPUT}),
21 //         gpio17 = new Gpio(17, {mode: Gpio.OUTPUT}),
22 //         gpio27 = new Gpio(27, {mode: Gpio.OUTPUT});
23
24
25 // FIN SECCION PINES
26
27
28 console.log('Esperando conexión...');

29 var sockets = {};
30
31 io.sockets.on('connection', function (socket)
32 {
33
34     // SECCION EVENTOS
35
36     // FIN SECCION EVENTOS
37 });
38
39
40 function stopStreaming(socket) {
41     delete sockets[socket.id];
42     // no more sockets, kill the stream
43     if (Object.keys(sockets).length == 0) {
44         if (ffmpeg_command){
45             ffmpeg_command.kill();
46             running_camera = false;
47             console.log('Stop streaming');
48         }
49     }
50 }

```

```

52 function startStreaming(socket) {
53   if (running_camera == false){
54     console.log('Starting streaming....');
55     var args = ["-f", "video4linux2", "-i", "/dev/video0", "-s",
56                 "300x150", "-f", "mpeg", "pipe:1", "-b:v 28k", "-bufsize 28k"];
57     ffmpeg_command = child_process.spawn("ffmpeg", args);
58     running_camera = true
59   }
60
61   ffmpeg_command.on('error', function(err, stdout, stderr) {
62     console.log("ffmpeg stdout:\n" + stdout);
63     console.log("ffmpeg stderr:\n" + stderr);
64     running_camera = false
65   });
66
67   ffmpeg_command.on('close', function (code) {
68     console.log('ffmpeg exited' + code );
69     running_camera = false
70   });
71
72
73   ffmpeg_command.stderr.on('data', function (data) {
74     //console.log('stderr: ' + data);
75   });
76
77   ffmpeg_command.on('end', function() {
78     console.log('Fin');
79     running_camera = false
80   });
81
82   ffmpeg_command.stdout.on('data', function (data) {
83     //console.log('stdout: ' + data);
84     var frame = new Buffer(data).toString('base64');
85     socket.emit('canvas',frame);
86   });
87
88 }

```

A continuación mostramos dos posibles eventos para añadir al código base superior a modo orientativo:

El primer fragmento de código se activa al recibir un evento tipo *action* (evento lanzado desde la interfaz al presionar cualquier botón generado por el usuario), captura el comando recibido, y si es igual a *UP*, entonces habilita el pin *gpio1* con el valor 1. Pin inicializado previamente en la sección de pines del código base.

```

1
2 socket.on('action', function (data){
3   if (data == 'UP') {
4     gpio1.digitalWrite(1);
5   }
6 });

```

El segundo fragmento de código devuelve la lectura del pin *gpio2* cada vez que recibe el comando *READ* y mandando al cliente el valor de lectura obtenido:

```

1  socket.on('action', function (data){
2      if (data == 'READ') {
3          var temp = gpio2.digitalRead(1);
4          socket.emit('robot_temp', {msg: temp});
5      }
6  });
7

```

En la parte referente a la interfaz de control de la aplicación RobotUI, para lanzar o capturar los comandos correspondientes a los ejemplos superiores, en el primer caso, debemos crear un botón cuyo código a emitir sea *UP* y para el segundo, debemos añadir un botón cuyo código de emisión sea *READ* y una etiqueta cuyo nombre de evento se corresponda con *robot\_temp*.

Una vez generado el código para nuestro robot debemos copiarlo a la placa Raspberry Pi o computador que actuará como Robot para nuestra aplicación y ejecutarlo. Para la ejecución del código introducimos el siguiente comando:

```
1 sudo node raspberry.js
```

Siendo *raspberry.js* el nombre del archivo que contiene nuestro código. Obteniendo el siguiente resultado:

```

pi@raspberrypi: ~/Desktop/SAILS-RobotUI/client
pi@raspberrypi:~/Desktop/SAILS-RobotUI/client $ sudo node raspberry.js
Esperando conexión...

|> Now connected to Sails.
\__/_ For help, see: http://bit.ly/1DmTvgK
              (using sails.io.js node SDK @v1.1.12)

```

Figura 9.3: Robot a la espera de conexión entrante.

## 9.2. Control

El vehículo puede ser controlado gracias a la interfaz ofrecida por la aplicación RobotUI, la cual permite el control del vehículo mediante el uso de un ratón y teclado convencional pero, además, en el presente proyecto se ha aprovechado para desarrollar una ampliación de la misma permitiendo incorporar la posibilidad de utilizar un gamepad clásico tipo videoconsola como muestra la siguiente figura:



Figura 9.4: Vista del gamepad utilizado.

Por otra parte, también puede ser controlado mediante el uso de un ratón y teclados convencionales.

# Capítulo 10

## Presupuesto

Descripción	Unidades	Precio € unidad	Total €
Raspberry Pi 3 Modelo B	1	38,70	38,70
Arduino Mega	1	38,70	35,98
Arduino Sensor kit	1	38,70	25,00
Cámara USB alta definición	1	39,90	39,90
Tarjeta de expansión con batería de Litio para Raspberry Pi	1	16,99	16,99
Batería Lipo 11.1v 2200mAh	1	13,99	13,99
Indicador Tester de baterías Lipo	1	8,90	8,90
Cargador baterías Lipo	1	21,90	21,90
Chasis vehículo Radiocontrol	1	54	54
Instancia Amazon Web Service	2 meses	5/mes	10

*CAPÍTULO 10. PRESUPUESTO*

---

<b>Total materiales:</b>	265.36 €
<b>Mano de obra:</b>	350Horas * 50 €/Hora = 17500 €
<b>I.V.A. %:</b>	21 %
<b>Total presupuesto:</b>	21496,08 €

# Capítulo 11

## Conclusiones

La elaboración de este trabajo ha resultado muy gratificante a nivel personal. Uno de los motivos principales ha sido la necesidad de trabajar en numerosas áreas de conocimiento entre las que encontramos, por un lado la programación, ya que ha sido necesaria la programación del microcontrolador que incorpora la placa Arduino, concretamente el ATmega2560. Por otra parte se ha realizado la programación de la placa Raspberry Pi, módulo encargado de la comunicación por puerto serie con la placa Arduino y la transmisión de los datos al servidor.

Por otro lado, la elaboración del vehículo robótico me ha permitido ampliar conocimientos en áreas más relacionadas con la electrónica, un área bastante desconocida para mí. Donde se ha trabajado, entre otras áreas, en lo referente a la transmisión de señales por puerto serie, la emisión de pulsos PWM junto con la utilización de multitud de sensores.

Entre los elementos desarrollados podemos destacar:

- La elaboración de un vehículo robótico haciendo uso de una Raspberry Pi 3 Model B.
- Programación del microcontrolador Atmega2560 alojado en una placa Arduino.
- Comunicación por puerto serie entre la placa Arduino y Raspberry Pi
- Realización de conexiones entre elementos y montaje del vehículo.
- Transmisión de gran cantidad de datos entre cliente servidor y servidor cliente. Streaming de vídeo y audio, emisión de comandos entre otros datos.
- Incorporación de algún sistema para el geoposicionamiento y escaneado de habitaciones, obstáculos, etc.

Pienso que el presente trabajo puede ser de gran utilidad permitiendo liberar a los trabajadores de aquellas tareas especialmente peligrosas para la salud y la integridad

física en diversas situaciones como exploración de zonas peligrosas o de difícil acceso, búsqueda de personas, etc ya que el vehículo desarrollado permite introducirlo de manera remota en zonas inaccesibles o peligrosas para las personas.

Una vez presentado podré continuar añadiendo mejoras y muchas otras cosas que tengo pensadas y que, posiblemente, se realicen a modo proyecto personal y ocio.

## 11.1. Mejoras futuras

La aplicación puede mejorarse en diversos aspectos. A continuación, se citan algunas de las mejoras que pueden llevarse a cabo:

- Incrementar el número de sensores en el sistema que permitan captar nuevos parámetros o situaciones.
- Dotar del sistema de la posibilidad de operar con más canales de conectividad distintas al WiFi como 4G, bluetooth, etc.
- Incorporar una cámara con posibilidad de visión 360 grados con un sistema de servomotores operable por el usuario.
- Añadir autenticación por un sistema de certificados por clave pública y privada que garantice que la conectividad entre dispositivo robótico y usuario es la correcta.

# Anexos

Anexos con los diferentes elementos software utilizados en el proyecto junto con las instrucciones para la instalación.

## .1. Arduino, entorno de desarrollo

El entorno de desarrollo (IDE) ofrecido por Arduino es el que nos permite escribir el programa y cargarlo en la placa, siendo posible su carga de dos maneras:

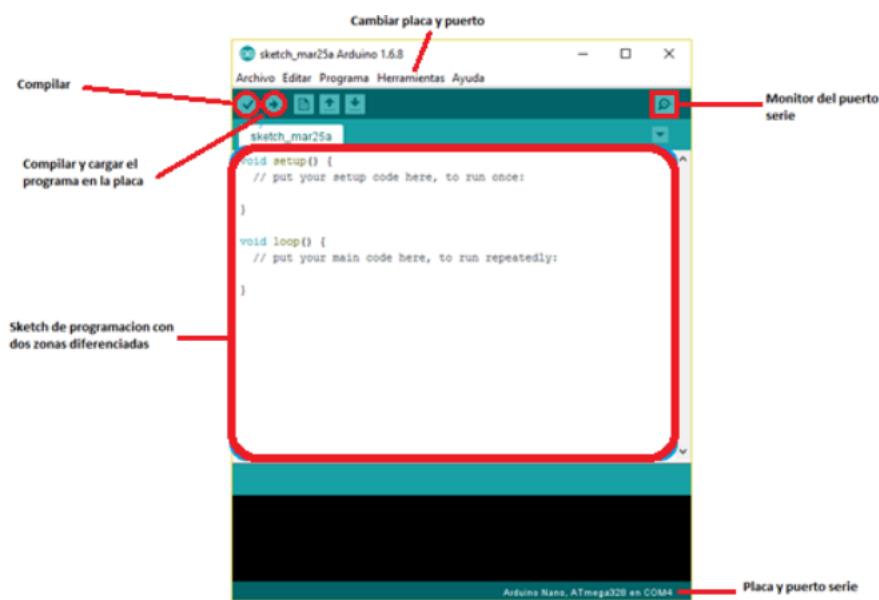


Figura 1: Entorno de desarrollo de Arduino.

El software de desarrollo de Arduino es publicado bajo una licencia libre y gratuita, la cual se puede descargar desde la página oficial incluyendo los drivers necesarios para cualquier placa Arduino, así como las librerías necesarias para su programación.

1. Si deseamos trabajar sin conexión, debemos descargar e instalar la versión más reciente de Arduino desde la página Arduino<sup>1</sup>.

<sup>1</sup><https://www.arduino.cc/en/Main/Software>

- Si tenemos una conexión de internet fiable, podemos usar el IDE online ya que siempre tendremos la última versión más actualizada del IDE, además de tener disponible nuestros códigos en la nube y poder acceder a su contenido a través de cualquier dispositivo y todo eso sin la necesidad de instalar actualizaciones nuevas. A continuación, Link para acceder al IDE online <sup>2</sup>.

### .1.1. Descarga e instalación

Para descargar el IDE en nuestro sistema operativo solo tendremos que seguir unos sencillos pasos:

- Descargar el programa gratuito Arduino IDE haciendo clic en el enlace anterior o dirigir a la web después Software.



Figura 2: Enlace para la descarga del software de Arduino.

- A continuación, podemos elegir entre Arduino web, pudiendo trabajar sin necesidad de descargar ningún software u optar por descargar Arduino IDE. Para su instalación solo tenemos que elegir nuestro sistema operativo y seguir las instrucciones de instalación.



Figura 3: Enlace de acceso al IDE de Arduino en su versión web.

## .2. Fritzing

Es una iniciativa de hardware de código abierto que hace que la electrónica sea accesible como material creativo para todos, que nos permita hacer diagramas, circuitos electrónicos y montajes, también sirve para hacer circuitos impresos PCB. También nos permite documentar nuestros prototipos y compartirlos con otros.

<sup>2</sup> <https://create.arduino.cc/>

## .2.1. Descarga e instalación

Para descargar Fritzing, accesible en el enlace inferior, donde tan solo debemos seleccionar nuestro sistema operativo y hacer click en Download.

<http://fritzing.org/download/?donation=0>

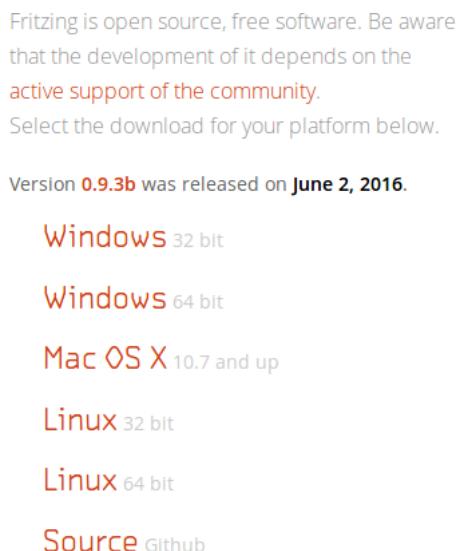


Figura 4: Sistemas operativos disponibles para Fritzing.

Para su instalación en versión linux debemos seguir los siguientes pasos:  
Nos dirigimos al directorio donde hemos realizado la descarga:

```
1 cd ~/Downloads
```

Posteriormente descomprimimos el archivo con el siguiente comando:

```
1 unzip -xzf fritzing-0.9.3b.linux.AMD64.tar.bz2
```

Nos posicionamos en el directorio de trabajo de Fritzing:

```
1 cd /home/wali/Downloads/fritzing-0.9.3b.linux.AMD64/
```

Ejecutamos Fritzing:

```
1 ./Fritzing
```

O:

```
1 sh Fritzing
```

## .3. Instalación de Node.js

Instalación de los prerequisitos:

```
1 sudo apt-get install python-software-properties python g++ make
```

Si está utilizando Ubuntu 16.04, necesitará hacer los siguiente:

```
1 sudo apt-get install software-properties-common
```

Añadimos el repositorio:

```
1 sudo add-apt-repository ppa:chris-lea/node.js
```

Actualizamos la lista de paquetes:

```
1 sudo apt-get update
```

Instalación de Node.js:

```
1 sudo apt-get install nodejs
```

## .4. Instalación del control de versiones Git

Para seguir esta guía es necesario disponer de una máquina con Ubuntu 14.04.

La forma más sencilla de tener Git instalado y configurado para su utilización es mediante el uso de los repositorios predeterminados de Ubuntu. Este es el método más rápido, pero, por contra puede ser que la versión disponible no sea la más reciente. Si necesita la última versión, deberá seguir los pasos para compilar Git desde el origen.

Si no necesitamos disponer de la última versión podemos instalarlo desde el repositorio de Ubuntu. Para ello introducimos los siguientes comandos en una terminal:

```
1 sudo apt-get update  
2 sudo apt-get install git
```

Esto descargará e instalará Git en el sistema. A continuación se describe los pasos para su configuración.

### .4.1. Configuración

Una vez que disponemos de Git instalado, necesitamos realizar una serie de pasos para añadir nuestros datos de acceso de nuestro repositorio.

La forma más sencilla de hacerlo es a través del comando *git config* proporcionando nuestro nombre y dirección de correo electrónico. Esto es debido a que Git incorpora esta información en cada commit que hacemos. Por ejemplo, si nuestro nombre es *RobotUI* y nuestro email *email@robotui.com*, los comandos de configuración serían los

siguientes:

```
1 git config --global user.name "RobotUI"  
2 git config --global user.email "email@robotui.com"
```

Finalmente podemos comprobar todos los valores de configuración establecidos escribiendo:

```
1 git config --list
```

Existen muchas más opciones configurables pero estos son los dos parámetros esenciales necesarios.



# Bibliografía

- [1] Amazon Web Services. Amazon Web Services. <https://www.aws.amazon.com/>. Visitado el 15-08-2018.
- [2] Andrew Lombardi. *WebSocket: Lightweight Client-Server Communications*. O'Reilly Media, 2012.
- [3] Antonio Serna Ruiz Francisco Antonio Ros García Juan Carlos Rico Noguera. *Guía Práctica de Sensores*. Creaciones Copyright, 2015.
- [4] A Willey Brand. *Amazon Web Services for Dummies*. Bernard Golden, 2017.
- [5] Mike Cantelon, Alex R. Young, Marc Harter, T.J. Holowaychuk, and Nathan Rajlich. *Node.js in Action*. Manning Publications, 2017.
- [6] Douglas Crockford. *JavaScript: The Good Parts*. O'Reilly, 2008.
- [7] Diego Aranda. *Electrónica: plataformas Arduino y Raspberry Pi*. Manuales USERS, 2015.
- [8] Eben Upton and Gareth Halfacree. *Raspberry Pi User Guide*. Raspberry Pi, 2013. <http://www.cs.unca.edu/~bruce/Fall14/360/RPiUsersGuide.pdf>, visitado el 12-08-2018.
- [9] Frantisek Korbel. *FFmpeg Basics: Multimedia handling with a fast audio and video encoder*. CreateSpace, 2015.
- [10] Git Hub. Git Hub. <https://github.com>. Visitado el 03-07-2018.
- [11] Pablo Hinojoda and JJ Merelo. *Aprende Git: ... y, de camino, GitHub*. Editorial Reviews, 2015.
- [12] Hau-Shiue Juang and Kai-Yew Lum. Design and Control of a Two-Wheel Self-Balancing Robot using the Arduino Microcontroller Board. Technical report, IEEE International Conference on Control and Automation, 5 2013.
- [13] Kimmo Karvinen Tero Karvinen. *Make: Getting Started with Sensors*. Make Media Books, 2014.
- [14] Nilson Mori Lazarin and Carlos Eduardo Pantoja. A Robotic-agent Platform For Embedding Software Agents usin Raspberry Pi and Arduino Boards. Technical report, UnED Nova Friburgo, 6 2011.

- [15] Leonel G. Corona Ramírez Griselda S. Abarca Jiménez Jesús Mares Carreño. *Sensores y actuadores, Aplicaciones con Arduino*. Grupo Editorial Patria, 2014.
- [16] D. Grahame Holmes Thomas A. Lipo. *Pulse Width Modulation for Power Converters*. Power Engineering, 2012.
- [17] Irl Nathan Mike McNeil. *Sails.js in Action*. Manning Publications, 2017.
- [18] Ministerio de Administraciones Públicas, Gobierno de España. Métrica v3. [http://administracionelectronica.gob.es/pae/Home/pae\\_Documentacion/pae\\_Metodolog/pae\\_Metrica\\_v3.html#.WQ79Wic1GkB](http://administracionelectronica.gob.es/pae/Home/pae_Documentacion/pae_Metodolog/pae_Metrica_v3.html#.WQ79Wic1GkB). Visitado el 10-08-2018.
- [19] Irl Nathan. Activityoverlord, an application to learn sails.js. <https://github.com/irlnathan/activityoverlord>. Visitado el 19-07-2018.
- [20] Official documentation. Node JS Documentation. <https://nodejs.org/es/docs/>. Visitado el 02-08-2018.
- [21] Official documentation. Sails JS Documentation. <https://sailsjs.com/documentation/reference>. Visitado el 14-08-2018.
- [22] Rohit Rai. *Socket.IO Real-Time Web Application Development*. Packt, 2013.
- [23] Sails.js. Sails.js | Realtime MVC Framework for Node.js. <http://sailsjs.com/>. Visitado el 27-08-2017.
- [24] Sandro Pasquali. *Deploying Node.js*. Packt Publishing, 2015.
- [25] Simon Monk. *Programming Arduino Getting Started with Sketches*. Mc Graw Hill, 2012. <http://index-of.es/Varios-2/Programming%20Arduino.pdf>, visitado el 10-08-2018.
- [26] Lakshminarasimhan Srinivasan, Julian Scharnagl, and Klaus Schilling. Analysis of WebSockets as the New Age Protocol for Remote Robot Tele-operation. Technical report, University of Wuerzburg, Department of Robotics and Telematics, 11 2013.
- [27] Lakshminarasimhan Srinivasan, Julian Scharnagl, Zhihao Xu, Nicolas Faerber, Dinesh K. Babu, and Klaus Schilling. Design and Development of a Robotic Tele-operation System using Duplex WebSockets suitable for Variable Bandwidth Networks. Technical report, University of Wuerzburg, Department of Robotics and Telematics, 11 2013.
- [28] Stefan Kottwitz. *LaTeX Beginner's Guide*. Packt Publishing, 2011.
- [29] Dr. Ovidiu Vermesan and Dr. Peter Friedss. *Internet of Things - From Research and Innovation to Market Deployment*. River Publishers Aalborg, 2014.
- [30] Wikibooks. The Book of LaTeX. <http://en.wikibooks.org/wiki/LaTeX>. Visitado el 08-07-2018.

- [31] Nazirah Ahmad Zaini, Norliza Zaini, Mohd Fuad Abdul Latip, and Nabilah Hamzah. Remote Monitoring System based on a Wi-Fi Controlled Car Using Raspberry Pi. Technical report, Universiti Teknologi MARA (UiTM) Shah Alam, Malaysia, Faculty of Electrical Engineering, 12 2016.



# GNU Documentation Free License

Version 1.3, 3 November 2008

Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc.

<<http://fsf.org/>>

Everyone is permitted to copy and distribute verbatim copies of this license document,  
but changing it is not allowed.

## Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document “free” in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non-commercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of “copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

## 1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The “**Document**”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as “**you**”. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A “**Modified Version**” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A “**Secondary Section**” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The “**Invariant Sections**” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none. The “**Cover Texts**” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A “**Transparent**” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not “Transparent” is called “**Opaque**”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The “**Title Page**” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

The “**publisher**” means any person or entity that distributes copies of the Document to the public.

A section “**Entitled XYZ**” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “**Acknowledgements**”, “**Dedications**”, “**Endorsements**”, or “**History**”.) To “**Preserve**

**the Title**" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

## **2. VERBATIM COPYING**

You may copy and distribute the Document in any medium, either commercially or non-commercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

## **3. COPYING IN QUANTITY**

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

## **4. MODIFICATIONS**

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the

“History” section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

- K. For any section Entitled “Acknowledgements” or “Dedications”, Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section Entitled “Endorsements”. Such a section may not be included in the Modified Version.
- N. Do not retitle any existing section to be Entitled “Endorsements” or to conflict in title with any Invariant Section.
- O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section Entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

## **5. COMBINING DOCUMENTS**

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgements”, and any sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements”.

## **6. COLLECTIONS OF DOCUMENTS**

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects. You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

## **7. AGGREGATION WITH INDEPENDENT WORKS**

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

## **8. TRANSLATION**

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and

disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled “Acknowledgements”, “Dedications”, or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

## **9. TERMINATION**

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

## **10. FUTURE REVISIONS OF THIS LICENSE**

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License “or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy’s public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

## **11. RELICENSING**

“Massive Multiauthor Collaboration Site” (or “MMC Site”) means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for

anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A “Massive Multiauthor Collaboration” (or “MMC”) contained in the site means any set of copyrightable works thus published on the MMC site.

“CC-BY-SA” means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

“Incorporate” means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is “eligible for relicensing” if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

## **ADDENDUM: How to use this License for your documents**

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright © YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled “GNU Free Documentation License”.

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the “with ... Texts.” line with this:

with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.