



ESCUELA SUPERIOR DE INGENIERÍA

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

SensorRS, Multisensor Robot System

Manuel López Urbina
Director: Arturo Morgado Estévez

Cádiz, 30 de agosto de 2018



ESCUELA SUPERIOR DE INGENIERÍA

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

RobotUI

Asistente de diseño de interfaz de control, seguimiento y sharing de robots en tiempo real

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

Cádiz, 30 de agosto 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 yo haya podido culminar mi ingeniería.

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

SensorRS, Multi-sensor Robot System, 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 dicha placa incorpora encargándose, posteriormente, de transmitir dicha información a la placa Raspberry Pi para su procesamiento y envío al servidor web para su seguimiento y control del vehículo.

La intención principal de este proyecto es la de permitir que usuarios dispongan de un vehículo robótico para el acceso y 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 sistema de telemetría.

Dicho sistema es configurado en la aplicación web RobotUI para permitir su 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 dispositivo robótico 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 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, de fácil construcción 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
Índice general	I
Índice de figuras	v
1. Introducción	1
1.1. Objetivos	5
1.2. Acerca de este documento	6
2. Conceptos básicos	9
2.1. Telemetría	9
2.1.1. Aplicaciones	10
2.2. Sensor	10
2.2.1. Clasificación de errores de medición	10
2.2.2. Tipos de sensores	11
2.2.3. Características	13
2.3. Transmisión y comunicación	15
2.4. Socket	15
2.5. WebSocket	16
2.6. La arquitectura TCP/IP y el modelo OSI	16
2.7. Streaming	18
2.8. Comunicación serie	19
2.8.1. Características	19
2.9. PWM	20
3. Estado del arte y herramientas utilizadas	21
3.1. Estado del arte	21
3.1.1. Introducción	21
3.1.2. Referencias	21
3.2. Tecnologías software utilizadas	22
3.2.1. L ^A T _E X	22
3.2.2. WebStorm	23
3.2.3. Github	23
3.2.4. Git	23
3.2.5. Amazon Web Services (AWS)	24
3.2.6. Node js	24

3.2.7. Npm	25
3.2.8. SocketIO	25
3.2.9. FFmpeg	26
3.2.10. Bootstrap	27
3.2.11. JQuery	27
3.3. Tecnologías hardware y materiales utilizados	27
3.3.1. Raspberry Pi Model B	28
3.3.2. Arduino	29
3.3.3. Arduino sensor kit	31
3.3.4. Controlador de motores doble puente H - L298N	32
3.3.5. Batería LiPo	33
3.3.6. Tarjeta de expansión con batería de Litio para Raspberry Pi	34
3.3.7. Cámara USB de alta definición	34
4. Requisitos	37
4.1. Requerimientos hardware	37
4.1.1. Análisis y selección de componentes electrónicos	38
4.2. Requerimientos software	40
4.3. Especificación	41
4.3.1. Requisitos funcionales	41
4.3.2. Requisitos no funcionales	42
5. Sensores	43
6. Desarrollo software	45
6.1. Metodología de desarrollo	45
6.2. Recolección de requisitos	46
6.2.1. Requisitos funcionales	46
6.2.2. Requisitos no funcionales	46
6.3. Diagrama de casos de uso	46
7. Comunicaciones	47
7.1. Introducción	47
7.2. Conexión y suscripción	49
7.3. Desconexión	55
7.4. Envío de comandos al robot	58
7.5. Captura de datos del robot	59
8. Análisis de requisitos	61
8.1. Montaje	61
8.1.1. Chasis	61
8.1.2. Tracción	61
8.1.3. Interconexión de elementos	63
8.1.4. Driver motores	64
8.1.5. Alimentación	67
8.1.6. Interconexión entre módulos y fijación	72
8.1.7. Esquema de conexiones	72

8.2. Control	74
8.3. Software de control	76
8.3.1. Entrada/Salida	77
8.3.2. Comunicaciones	77
9. Pruebas	89
9.1. Plan de pruebas	89
10. Organización temporal	91
10.1. Planificación temporal de tareas	94
10.1.1. Hito 1: Planificación y análisis	94
10.1.2. Hito 1: Definición de requisitos	95
10.1.3. Hito 2: Montaje del vehículo	95
10.1.4. Hito 3: Programación del vehículo SensorRS	95
10.1.5. Hito 4: Mejoras en la aplicación de control RobotUI	95
10.1.6. Hito 5: Documentación	96
10.2. Diagrama de Gantt	96
11. Guía de Usuario	99
11.0.1. Objetivo de esta guía	100
11.0.2. Dirigido a	100
11.0.3. Obtener SensorRS	100
11.1. Uso de SensorRS	100
11.1.1. Configuración	100
11.1.2. Programa tu robot	101
12. Comentarios finales	107
12.1. Presupuesto	108
12.2. Conclusiones	109
12.3. Mejoras futuras	110
Anexos	111
.1. Entorno de desarrollo	111
.2. Instalación de Node.js	111
.3. Instalación del control de versiones Git	112
.3.1. Configuración	112
Bibliografía	115
GNU Documentation Free License	117

Índice de figuras

1.1. Logo SensorRS ¹	3
1.2. Imagen del vehículo SensorRS ²	4
1.3. Página principal de RobotUI.	5
2.1. Representación de capas o niveles OSI y TCP/IP.	17
2.2. Representación de los sockets como una interfaz de la capa de transporte del protocolo TCP/IP.	18
3.2. Imagen de una Raspberry Pi 3 Model B	29
3.3. Placa Arduino MEGA 2560	30
3.4. Características Arduino MEGA 2560	31
3.5. Pack de sensores para Arduino	32
3.6. Imagen de la Controlador de motores doble puente H - L298N utilizada.	33
3.7. Imagen de la batería LiPo utilizada.	33
3.8. Imagen de la tarjeta de expansión con batería de Litio utilizado.	34
3.9. Imagen de la cámara USB utilizada.	35
3.1. Instancia RobotUI en AWS	36
4.1. Esquema GPIO de una Raspberry Pi Model B+.	39
4.2. Placa Arduino Mega donde se visualiza la disposición de sus pines E/S. .	40
6.1. Subsistemas existentes en el proyecto junto con el modelo de ciclo de vida utilizado para su desarrollo.	46
7.1. Peticiones entre un navegador y un servidor HTTP con y sin el empleo de Websockets.	47
7.2. Representación de una sala compuesta por dos clientes.	48
7.3. Página de gestión de usuarios actualizable en tiempo real.	50
7.4. Esquema representativo del flujo de datos originado por un usuario controlador de un robot.	58
7.5. Esquema representativo del flujo de datos originado por un usuario tras la captura de datos procedentes del robot.	59
8.1. Servomotor utilizado.	62
8.2. Funcionamiento del puente H.	66
8.3. Pines de entrada/salida del módulo L298N empleado.	67
8.4. Conjunto Raspberry Pi y módulo de expansión de alimentación.	69
8.5. Batería LiPo que alimenta los motores.	69

8.6. Adaptador de entrada USB a protoboard.	70
8.7. Adaptador de entrada USB a protoboard.	70
8.8. Vista del conector J1.	71
8.9. Vista del conector J2 y J3.	71
8.10. Vista del conector J4 y J5.	72
8.11. Cables de interconexión para protoboard.	72
8.12. Esquema de conexiones del robot de pruebas.	73
8.13. Vista superior del vehículo.	73
8.14. Vista superior del vehículo.	74
8.15. Diagrama de casos de uso para la interacción con el robot.	75
8.16. Autómata representativo de los diferentes estados del robot.	76
8.17. Canales de comunicación abiertos por el robot.	78
8.18. Diagrama de bloques del robot controlado por WiFi.	81
 10.1. Panel de actividades - Trello	91
10.2. Descomposición de las tareas implicadas en la construcción del vehículo robótico SensorRS (Primera Parte).	93
10.3. Descomposición de las tareas implicadas en el desarrollo del proyecto (Segunda parte).	94
10.4. Diagrama de Gantt 1. Desarrollo del proyecto.	97
10.5. Diagrama de Gantt 2. Desarrollo del proyecto.	98
 11.1. Página principal RobotUI.	101
11.2. Panel informativo de un robot donde se aprecia su identificador.	102
11.3. Robot a la espera de conexión entrante.	105
 1. Entorno de desarrollo de Arduino.	111

Capítulo 1

Introducción

La robótica es una rama de la ingeniería, la cual se ocupa del diseño, construcción, operación y uso de robots¹, 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 sensor² 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

¹Robot: 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.

²En 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 típicos de las zonas de desastre.
- Capacidad de manipular y utilizar un surtido diverso de herramientas diseñadas para los seres humanos.
- Capacidad para ser manejado por los seres humanos que hayan tenido poca o ninguna formación robótica.
- Autonomía 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 lo que es lo mismo, un sistema de telemetría³.

La telemetría es una tecnología que está en pleno auge al permitir recibir información de un entorno sin necesidad de estar en él presente. La información que se deseé medir para posteriormente transmitirla puede ser muy variada, según la situación. Desde el punto de vista de la comunicación, la información que se transmite es 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.

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

³En el capítulo de conceptos básicos puede acceder a un apartado específico destinado a la telemetría 2.1

CAPÍTULO 1. INTRODUCCIÓN

elaboración de este proyecto que unifica ambos campos anteriormente citados.

Así surgió *SensorRC*, *Multi-sensor Robot Sysmtem*.



Figura 1.1: Logo SensorRS ⁴.

SensorRS, Multi-sensor Robot System (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 eleborará 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*⁵, 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 RobotUI⁶, 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.

⁴Logotipo de *SensorRS*, Multi-sensor Robot Sysmtem.

⁵ 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.

⁶ 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>



Figura 1.2: Imagen del vehículo SensorRS ⁷.

⁷Vehículo desarrollado con la finalidad de integrarlo en la aplicación RobotUI. Su desarrollo y características quedan descritas en el capítulo ??.

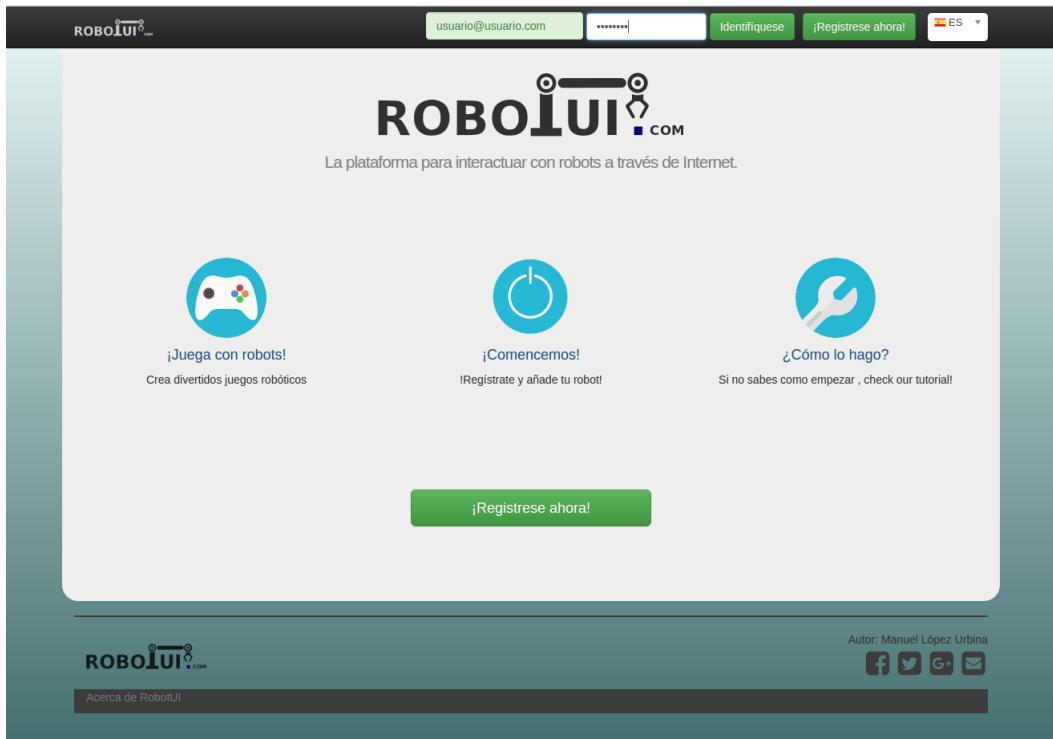


Figura 1.3: Página principal de RobotUI.

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 del entorno más próximo en el que se encuentra.

En lo referente al área de la programación, más concretamente con la programación de microcontroladores⁸ permitirá crear un sistema fácilmente escalable donde

⁸Un 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/-

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 el no disponer de grandes conocimientos en la materia sean un impedimento a la hora de comenzar a desarrollar sus ideas.

1.2. 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 ??, Construcción del vehículo, se recogen aquellos aspectos referentes al montaje, configuración y programación de un robot de pruebas para la demostración y uso de la aplicación desarrollada en el presente proyecto.
- En el capítulo 9, Pruebas, se detallan las pruebas a las que se ha sometido el sistema.
- En el capítulo 10, 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.

salida.

- En el capítulo 11, 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 12, Comentarios finales, se hace mención de las conclusiones obtenidas tras la realización del proyecto además de las posibles mejoras aplicables y presupuesto.
- 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 que permite la medición remota de magnitudes físicas y el posterior envío de la información hacia el operador del sistema. La palabra telemetría procede de las palabras griegas *tele* (tele), que quiere decir a distancia, y la palabra *(metron)*, que quiere decir medida.

El envío de información hacia el operador en un sistema de telemetría se realiza típicamente mediante comunicación inalámbrica, aunque también se puede realizar por otros medios (teléfono, redes de computadoras, enlace de fibra óptica, etcétera). Los sistemas de telemetría reciben las instrucciones y los datos necesarios para operar desde el Centro de Control.

Aunque el término se refiere a mecanismos de transferencia de datos inalámbricos (por ejemplo, usando sistemas de radio, ultrasonidos o infrarrojos), también abarca datos transferidos a otros medios tales como una red telefónica o de computadora, enlace óptico u otras comunicaciones por cable como portadoras de líneas eléctricas. Muchos sistemas modernos de telemetría aprovechan el bajo costo y la ubicuidad de las redes GSM mediante el uso de SMS para recibir y transmitir datos de telemetría.

2.1.1. Aplicaciones

La telemetría se utiliza en grandes sistemas, tales como naves espaciales, plantas químicas, redes de suministro eléctrico, redes de suministro de gas entre otras empresas de provisión de servicios públicos, debido a que facilita la monitorización automática y el registro de las mediciones, así como 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 naves espaciales y satélites.

La telemetría se utiliza en infinidad de campos, tales como la exploración científica con naves tripuladas o no (submarinos, aviones de reconocimiento y satélites), diversos tipos de competición (por ejemplo, Fórmula 1 y MotoGP), o la operación de modelos matemáticos destinados a dar sustento a la operación de embalses. En las fábricas, oficinas y residencias, el monitoreo del uso de energía de cada sección o equipo y los fenómenos derivados (como la temperatura) en un punto de control por telemetría facilita la coordinación para un uso más eficiente de la energía.

2.2. Sensor

En su definición más amplia, un sensor es un dispositivo, módulo o subsistema 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.

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). Algunos sensores también pueden afectar lo que miden; por ejemplo, un termómetro a temperatura ambiente insertado en una taza de líquido caliente enfriará el líquido mientras el líquido calienta el termómetro. Los sensores generalmente están diseñados para tener un pequeño efecto en lo que se mide; hacer el sensor más pequeño a menudo mejora esto y puede presentar otras ventajas. El progreso tecnológico permite fabricar cada vez más sensores a escala microscópica como microsensores que utilizan la tecnología MEMS. En la mayoría de los casos, un microsensor alcanza una velocidad y sensibilidad significativamente mayores en comparación con los enfoques macroscópicos

2.2.1. Clasificación de errores de medición

Un buen sensor obedece a las siguientes reglas:

- Es sensible a la propiedad medida.

- Es insensible a cualquier otra propiedad que pueda encontrarse en su aplicación.
- No influye en la propiedad medida.

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.

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.

2.2.2. Tipos de sensores

Listado de los diferentes tipos de sensores junto con algunos ejemplos:

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

Acústico:

- Microfono

Sensor de acidez:

- ISFET

Luz:

- Fotodiode
- 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.

2.2.3. Características

El transductor 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 transductores 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 transductores se pueden agrupar en dos grandes bloques:

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.

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.

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 pasabajos.

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 protocolos¹, 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/IP², provista usualmen-

¹Protocolo: reglamento o serie de instrucciones que se fijan por tradición o por convenio.

²TCP/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.

te 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.

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-duplex³ 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 IETF⁴ 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

Proviene de los nombres de dos protocolos importantes incluidos en el conjunto TCP/IP, es decir, del protocolo TCP y del protocolo IP.

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

⁴Internet 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.

CAPÍTULO 2. CONCEPTOS BÁSICOS LA ARQUITECTURA TCP/IP Y EL MODELO OSI

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.

En la siguiente figura 2.1 se aprecian los niveles o capas de los modelos OSI y TCP/IP.



Figura 2.1: Representación de capas o niveles OSI y TCP/IP.

Los sockets dentro del modelo TCP/IP se pueden ver como una interfaz con la capa de transporte.

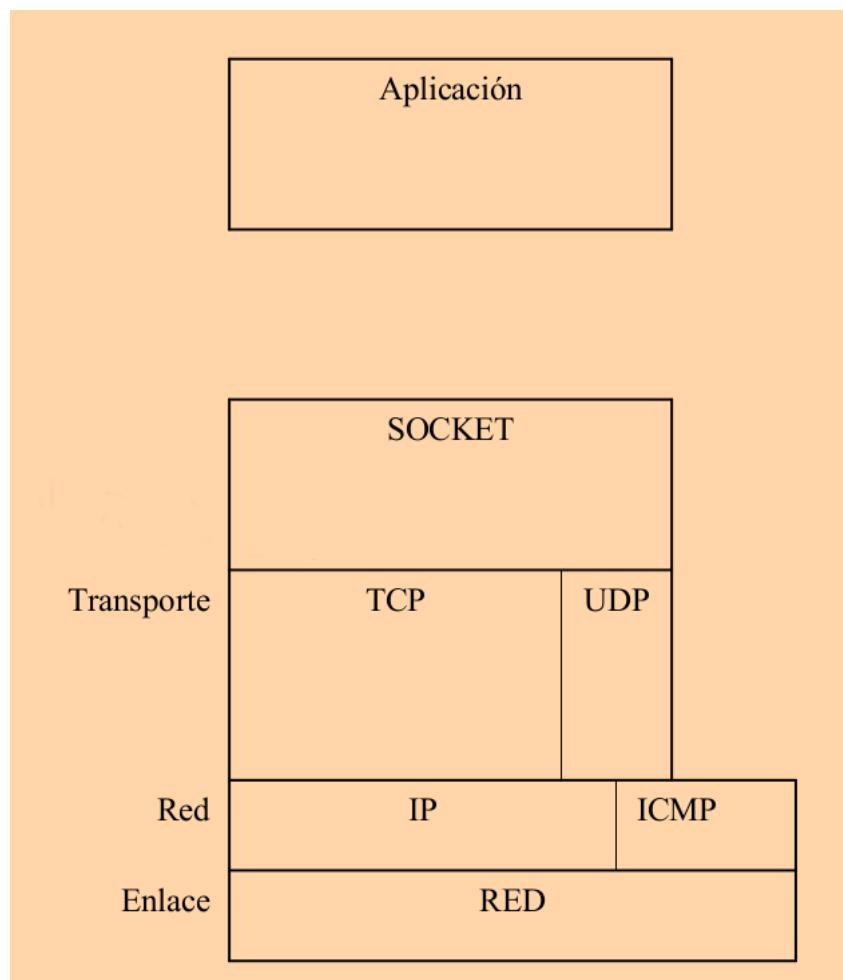


Figura 2.2: Representación de los sockets como una interfaz de la capa de transporte del protocolo TCP/IP.

2.7. Streaming

La retransmisión (en inglés streaming, también denominado transmisión) es la distribución digital de contenido multimedia a través de una red de computadoras, de manera que el usuario utiliza el producto a la vez que se es descargado. La palabra retransmisión se refiere a una corriente continua que fluye sin interrupción, habitualmente audio o vídeo, aplicándose la difusión de vídeo en el presente proyecto.

Este tipo de tecnología funciona mediante un búfer 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.

La retransmisión requiere de una conexión por lo menos de igual ancho de banda que la tasa de transmisión del servicio. La retransmisión de vídeo por Internet se popularizó

a fines de la década de 2000, cuando la contratación del suficiente ancho de banda para utilizar estos servicios en el hogar se hizo lo suficientemente barato.

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.

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
- I²C
- 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. Además, PWM es uno de los dos principales algoritmos utilizados en los cargadores de baterías solares fotovoltaicas, el otro es el seguimiento del punto de máxima potencia.

El valor promedio de voltaje (y corriente) alimentado a la carga se controla al encender y apagar el interruptor entre suministro y carga a una velocidad rápida. Cuanto más tiempo esté encendido el interruptor en comparación con los períodos de desconexión, mayor será la potencia total suministrada a la carga.

La frecuencia de conmutación de PWM debe ser mucho más alta que la que afectaría a la carga (el dispositivo que usa la potencia), lo que significa que la forma de onda resultante percibida por la carga debe ser lo más suave posible. La velocidad (o frecuencia) a la que debe cambiar la fuente de alimentación puede variar mucho según la carga y la aplicación, por ejemplo:

El cambio debe hacerse varias veces por minuto en una estufa eléctrica; 120 Hz en un atenuador de lámpara; entre unos pocos kilohercios (kHz) y decenas de kHz para un accionamiento de motor; y en las decenas o cientos de kHz en amplificadores de audio y fuentes de alimentación de computadoras.

El término ciclo de trabajo describe la proporción de tiempo “conectado” al intervalo regular o “período” de tiempo; un ciclo de trabajo bajo corresponde a la baja potencia, ya que la energía está apagada la mayor parte del tiempo. El ciclo de trabajo se expresa en porcentaje, 100 % siendo completamente activado. 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

REDACTAR ESTADO DEL ARTE

En la actualidad podemos observar cómo la robótica cada vez se encuentra más presente en nuestras vidas cotidianas tanto como para facilitarnos ciertas tareas como para entretenimiento. 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 multitud de proyectos novedosos y originales donde sus autores describen su desarrollo y por regla general lo acompañan de un vídeo donde se demuestra su funcionamiento.

Existen también comunidades que organizan competiciones de drones y/o robots, comunidades educativas asociadas a alguna tecnología concreta como pueden ser Arduino o Raspberry Pi, etcétera, donde nuevamente todas las descripciones de los diferentes proyectos se realizan con la documentación oportuna y en la mayoría de los casos acompañadas de un vídeo demostrativo con la inexistencia de interacción directa por parte del usuario.

Todo esto en lo referente a un contexto cotidiano en la 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.

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* [19], en el que se describe el desarrollo de un sistema teleoperable por 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* [18].

Por otra parte, el artículo *Remote Monitoring System based on a Wi-Fi Controlled Car Using Raspberry Pi* [23], describe un sistema de monitorización construido sobre una placa Raspberry Pi de las mismas características que la empleada para el robot SensorRS.

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 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^AT_EX

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

L^AT_EX 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 [20] y [22].

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. Github



Web: <https://about.github.com/> [7]

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.

3.2.4. Git



Web: <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 [8]. La cual hace un recorrido por

todas las posibilidades que brinda esta herramienta.

3.2.5. Amazon Web Services (AWS)



Web: <https://aws.amazon.com/es> [4]

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.

3.2.6. Node.js



Web: <https://nodejs.org/es/>

documentación: <https://nodejs.org/es/docs/> [13]

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 [3] 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 [13].

3.2.7. 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.

3.2.8. 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.

3.2.9. 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.

3.2.10. 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.

3.2.11. JQuery



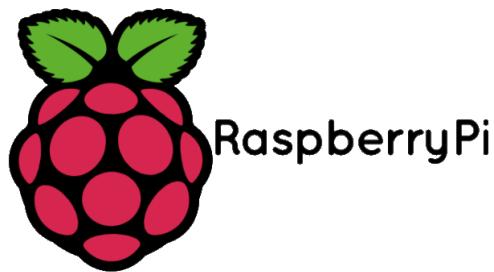
Web: <https://jquery.com/>

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 3.2: 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 3.3: Placa Arduino MEGA 2560

Para el presente proyecto 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 3.4: 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 referente a sensores [5](#).

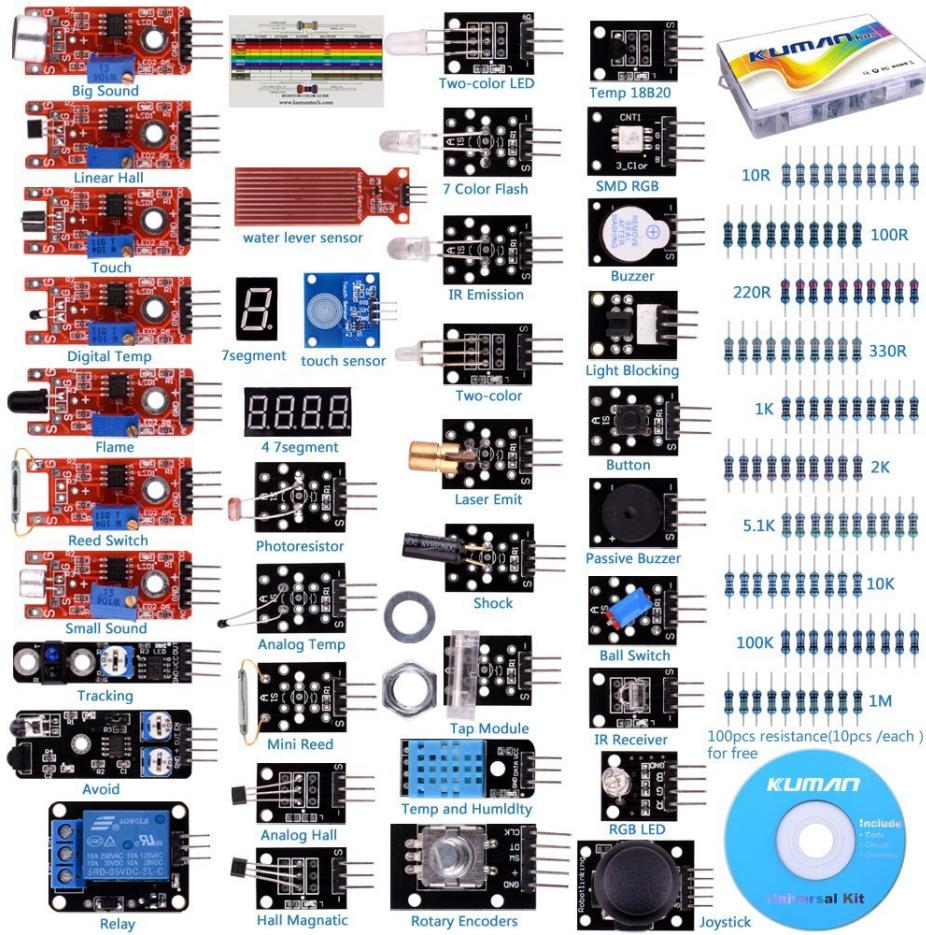


Figura 3.5: Pack de sensores para Arduino

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 motor¹.

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.

¹Podrá acceder a información más detallada de la presente controladora en la sección correspondiente al capítulo

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 proyecto.

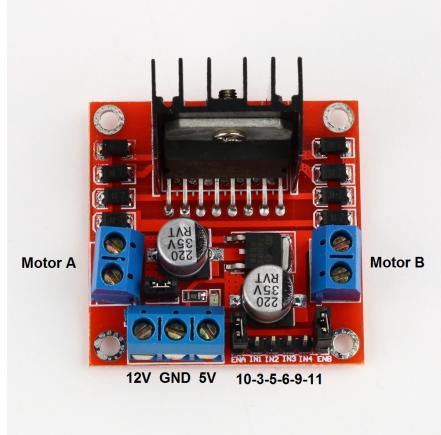


Figura 3.6: Imagen de la Controlador de motores doble puente H - L298N utilizada.

3.3.5. Batería LiPo

Para alimentar los motores y su controladora se ha empleado una batería LiPo de 1000mAh a 3,7V. 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. Siendo ideales para este tipo de usos.



Figura 3.7: Imagen de la batería LiPo utilizada.

3.3.6. 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.

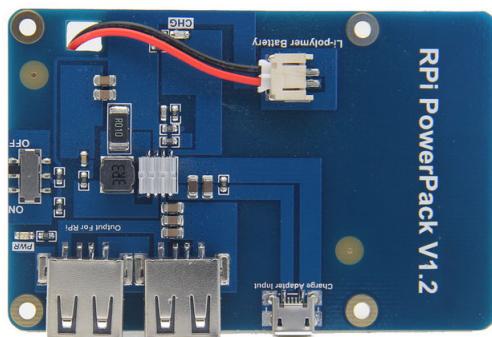


Figura 3.8: Imagen de la tarjeta de expansión con batería de Litio utilizado.

3.3.7. 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 3.9: 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. TECNOLOGÍAS HARDWARE

CAPÍTULO 3. ARTE Y TECNOLOGÍAS

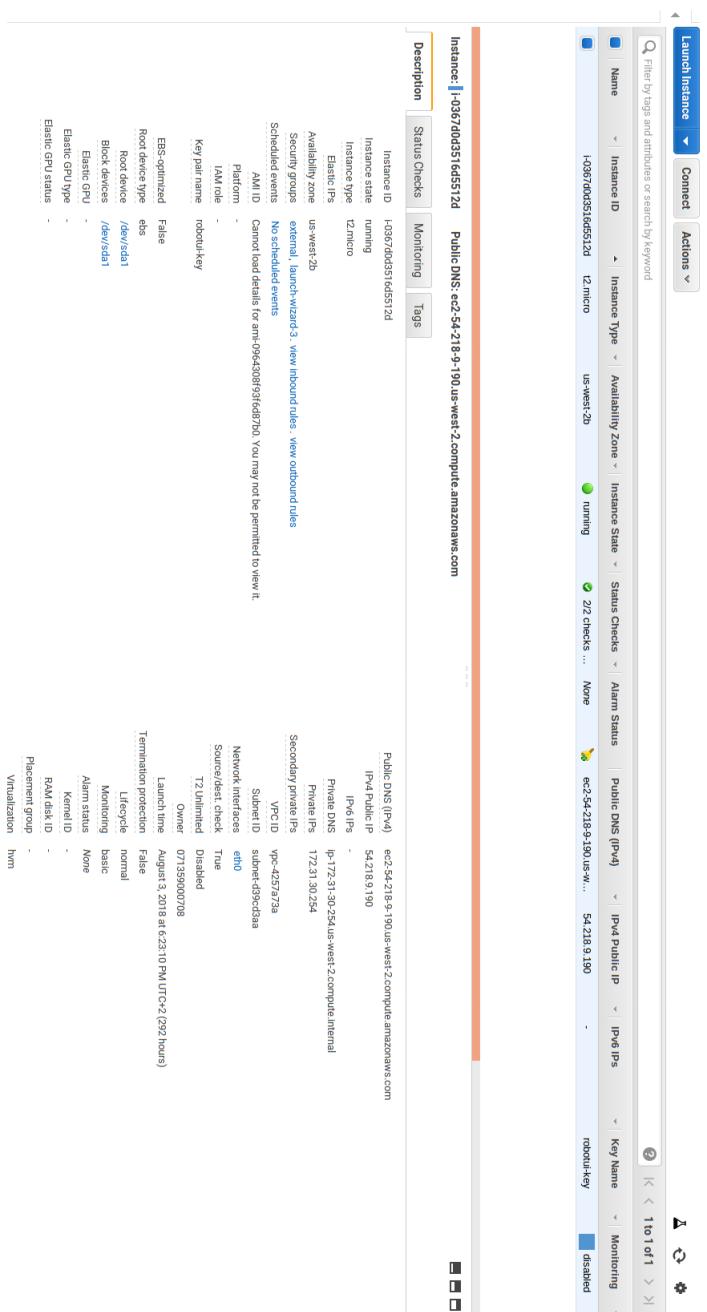


Figura 3.1: Instancia RobotUI en AWS

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 incial 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 porpulsor 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 IOT¹. 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.

¹ 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

El punto fuerte de Arduino no es su potencia de cálculo, ni la memoria de la placa, ni la frecuencia del procesador. Entonces, ¿Por qué debemos considerar dichas placas para utilizarlas en nuestros proyectos? El punto fuerte de Arduino está en 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.

Un Arduino Mega dispone 54 pines de entrada salida digital, de los cuales quince pueden ser utilizados como salidas PWM y controlar con ellos la velocidad de motores, por citar alguno de los ejemplos. 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 su favor es la facilidad de prototipado que ofrece y, por supuesto, los Shields o mochilas de expansión, que ofrecen dotar a nuestra placa desde conectividad Wi-Fi, GPS, conectividad por radio de largo alcance, displays táctiles, etc. Y muchas de ellas con un bajo coste.

Por el contrario la Raspberry Pi puede presumir de músculo y de potencia de cálculo comparada con las placas Arduino como memoria y capacidades multimedia, que van desde la reproducción de video en HD, pasando por una salida de audio (de no demasiada calidad, todo hay que decirlo, aunque hay alternativas), así como una salida de vídeo compuesto. Y, pese a no contar con las capacidades de interconexión de Arduino y con todos sus shields, sí que contamos con (cada vez más) placas de expansión en forma de shields. Todo ello 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.

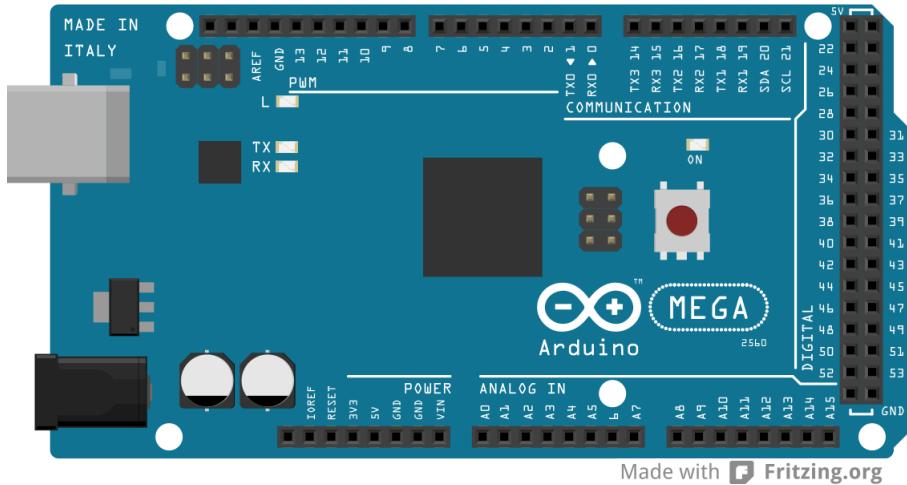


Figura 4.2: Placa Arduino Mega donde se visualiza la disposición de sus pines E/S.

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 base de datos estará implementada en un lenguaje objeto no relacional como MongoDB.
- La aplicación estará realizada en el lenguaje de programación Python.
- 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.

Capítulo 5

Sensores

En el presente capítulo se realizará una descripción de los diferentes sensores utilizados detallando sus características, esquema de conexión y código utilizado.

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 web y una metodología de desarrollo en cascada para el desarrollo de la parte software referente al robot de pruebas.

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 otro lado, para el desarrollo del vehículo de pruebas y por su simplicidad, se ha optado por un desarrollo en cascada. El modelo de desarrollo en cascada resulta adecuado en situaciones en las que:

- Se dispone de unos requisitos claros y precisos.
- El sistema a desarrollar es de pequeña envergadura.
- Las tecnologías utilizadas son conocidas por los desarrolladores.

Siendo precisamente éstas las características del proyecto del vehículo a desarrollar puesto que se trata de un desarrollo de pequeño tamaño y las herramientas empleadas ya me resultaban conocidas tras la realización de otros trabajos previos.

Por tanto el proyecto queda distribuido en los siguientes subsistemas:

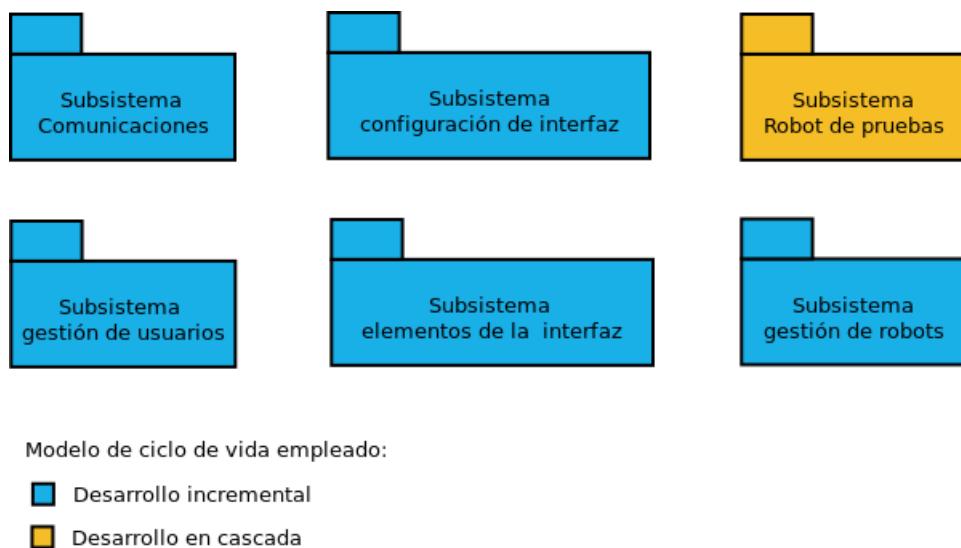


Figura 6.1: Subsistemas existentes en el proyecto junto con el modelo de ciclo de vida utilizado para su desarrollo.

6.2. Recolección de requisitos

6.2.1. Requisitos funcionales

6.2.2. Requisitos no funcionales

6.3. Diagrama de casos de uso

Capítulo 7

Comunicaciones

En el presente capítulo comenzaremos con una introducción sobre el funcionamiento y los fundamentos teóricos sobre cómo se gestionan las diferentes conexiones y eventos de cualquier aplicación Sails para, posteriormente, centrarnos específicamente en la descripción de un caso práctico desarrollado en el presente proyecto con la finalidad de comprender mejor su funcionamiento y poder afianzar conocimientos.

7.1. Introducción

En esta sección comenzaremos destacando aquellos aspectos teóricos sobre cómo Sails, framework Node JS utilizado en el desarrollo, trabaja con Websocket y Socket.io.

Como sabemos, un servidor HTTP no puede enviar datos a menos que un cliente los haya solicitado mediante una petición. Los Websockets[1], en cambio, presentan la particularidad de que permiten que un servidor envíe datos a un cliente sin la necesidad de que éstos sean solicitados, al menos de una manera inmediata. Estas solicitudes, realizadas con anterioridad, se formalizan mediante suscripciones, concepto que iremos constantemente haciendo referencia a lo largo del presente capítulo y que debemos recordar.

Anteriormente comentamos que el servidor HTTP no puede enviar datos a menos que el cliente lo haya solicitado y los Websockets permiten que un servidor envíe datos no solicitados una vez que se hace una conexión inicial (suscripción) desde el lado del

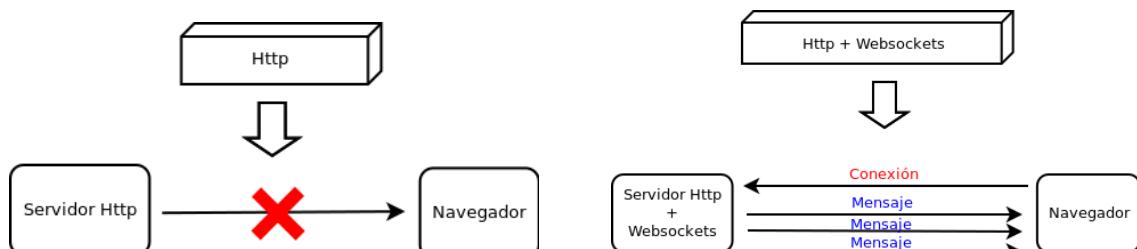


Figura 7.1: Peticiones entre un navegador y un servidor HTTP con y sin el empleo de Websockets.

cliente. Una aplicación Sails típica queda dividida en dos partes. La primera, en el lado del servidor, consta de un servidor HTTP junto con un nodo central que actúa como un servidor de sockets.

En segundo lugar, en el lado del cliente, se dispone de los diferentes códigos HTML y funciones JavaScript para realizar las conexiones con el servidor. Estas conexiones cliente-servidor permiten que cualquier otro cliente conectado pueda enviar mensajes al servidor para que a su vez éstos sean emitidos a los clientes que se encuentren conectados en la aplicación en ese instante.

Además cada socket posee un identificador único que identifica de manera inequívoca el cliente o, en nuestro caso el navegador que está accediendo a la página.

Destacar también el concepto de salas o rooms de Socket.io. Las salas nos permite agrupar los sockets de modo que en lugar de mensajes que son enviados a todos los sockets conectados, se puede enviar mensajes sólo a los sockets que están asociados con una sala. De esta podremos mandar mensajes específicos para un cliente concreto, todos ellos o un conjunto de los mismos.

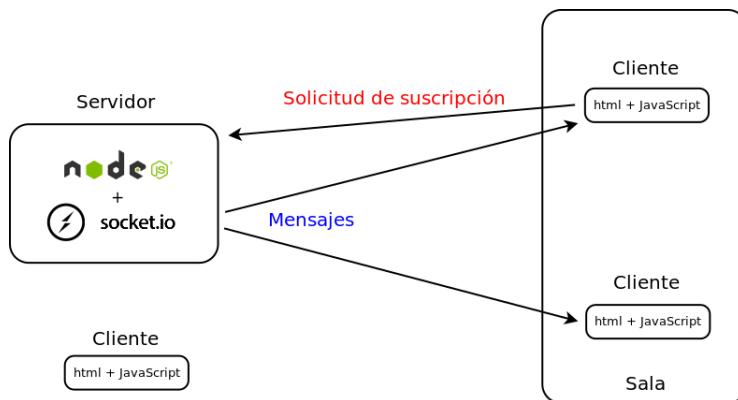


Figura 7.2: Representación de una sala compuesta por dos clientes.

De todo lo anterior podemos deducir que Sails es un framework especialmente potente, el cual proporciona infinidad de posibilidades a desarrollar. A continuación se cita un fragmento tal y como podemos extraer de la página oficial de Sails:

Sails.js facilita el desarrollo de aplicaciones Node.js empresariales. Ha sido diseñado para imitar el patrón MVC de frameworks como Ruby on Rails, pero con soporte para los requisitos de aplicaciones modernas: data-driven APIs con una arquitectura escalable y service-oriented. Es especialmente bueno para el desarrollo de chats, cuadros de mando en tiempo real o juegos multijugadores.

En este caso menciona diferentes aplicaciones tipo que pueden desarrollarse con Sails, desde un chat, cuadros de mando en tiempo real o juegos multijugadores. RobotUI

resulta como una combinación de las anteriores puesto que incorpora cuadros de mandos, sistema de mensajería y también es considerado como un juego ya que también está enfocado al entretenimiento.

7.2. Conexión y suscripción

Tras la comprensión de los fundamentos descritos en el punto 7.1 donde se recogen los principios de comunicación con Websockets pasamos a trasladarlos a un caso práctico describiendo una de las funcionalidades del proyecto donde éstos son aplicados de tal manera que se facilite su comprensión¹.

Como sabemos, Sails nos ayuda a incorporar funcionalidades en tiempo real a nuestra aplicación web gestionando una serie de eventos aplicándolos sobre los modelos. Para el caso a describir nos centraremos en el modelo *Usuario* de RobotUI. Uno de los atributos de este modelo es un booleano *online*, el cual representa si un usuario se encuentra logueado en el sistema o no. El objetivo es saber cuando el usuario cambia el estado para poder proporcionar una actualización en tiempo real de la página de gestión de usuarios 7.3 de manera que cuando usuario inicie sesión o salga de la aplicación se alterne entre las imágenes online:  y offline:  sin necesidad de refrescar la página manualmente.

¹ Referencias bibliográficas consultadas que recogen casos prácticos acompañados de explicaciones y que han resultado de especial utilidad para el estudio y comprensión del funcionamiento de una aplicación con Websockets: [15] y [12] y que han sido trasladados al presente proyecto.

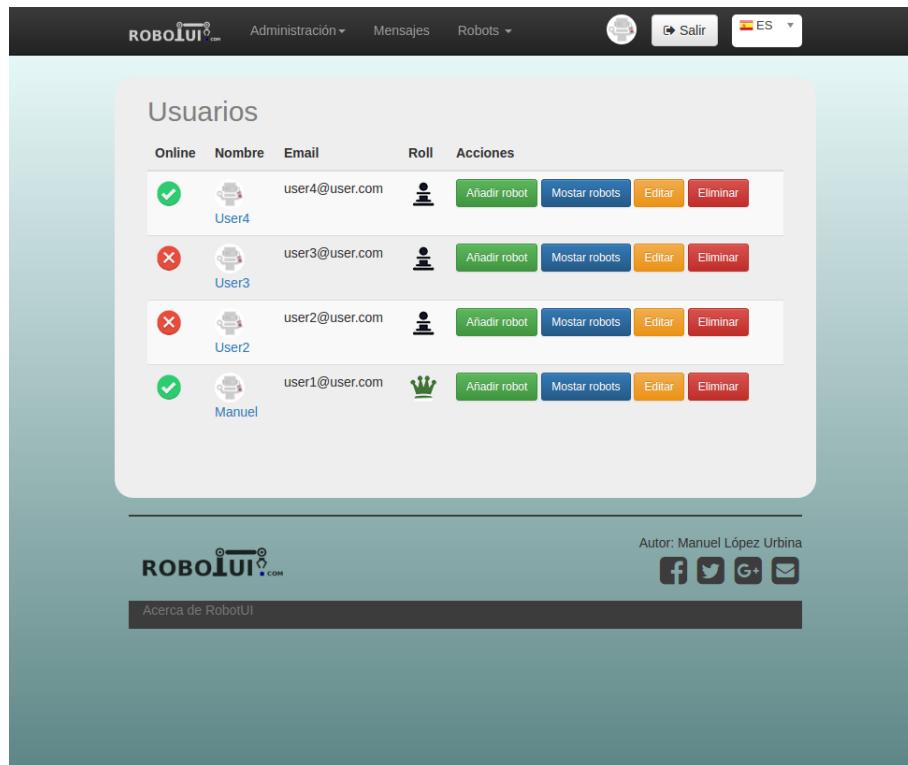


Figura 7.3: Página de gestión de usuarios actualizable en tiempo real.

Para capturar los eventos correspondientes al modelo *Usuario* o cualquier otro debemos realizar una suscripción. Sails incorpora dos modalidades de suscripción. suscripción a clase y suscripción a modelo [10].

¿Cuál es la diferencia entre suscribirse a una “clase” frente a la suscripción a una “instancia” de un modelo? Al suscribirse a la “clase” del modelo, el socket podrá escuchar la creación de nuevas instancias de modelo mediante un método denominado `publishCreate()`. Mientras que la suscripción a “instancia” permite al socket escuchar los cambios de modelos a través de los métodos `publishUpdate` y/o `publishDestroy` de una instancia o conjunto de instancias en concreto.

Por tanto, centrándonos en el ejemplo de los usuarios, debemos realizar la suscripción mediante la definición de una acción en el controlador de usuario. La acción se denominará `user_subscribe`.

En dicha acción podremos acceder al socket solicitante vía `req.socket`. Así que, prime ramente se realiza una suscripción a la sala de “clase” del usuario mediante `User.watch` pasando `req.socket` como argumento. De esta manera nos hemos suscrito a la clase del modelo de usuario.

Seguidamente se realiza la suscripción a las instancias del modelo de usuario mediante el método `User.subscribe` pasando como parámetro el socket solicitante obtenido a través de `req.socket` seguido de un segundo argumento, que se corresponde con las ins-

tancias existentes de usuarios. Las instancias de los usuarios existentes son recuperadas mediante un método de búsqueda para que, posteriormente, su salida sea pasada al método de suscripción como segundo argumento. Con esto ya nos hemos suscrito a las salas de instancias del modelo de usuario.

A continuación mostramos el código que realiza la suscripción al modelo Usuario en sus dos modalidades, suscripción a clase de modelo y suscripción a instancias de modelo descritas anteriormente:

```

1 user_subscribe: function (req, res, next) {
2   if (req.isSocket) {
3     //Update, destroy...
4     User.find(function foundUsers(err, users) {
5       if (err) return res.badRequest(err);
6
7       User.subscribe(req.socket, users);
8     });
9
10
11   //Create
12   User.watch(req);
13   console.log('User ' + req.session.User.id + 'with socket id ' +
14     sails.sockets.id(req) + ' is now subscribed to the model
15     class \'User\'');
16   } else {
17     res.view();
18   }
19 }
```

Ahora bien, ya nos encontramos suscritos a un modelo, pero... ¿Qué ocurre con publishCreate, publishUpdate y publishDestroy?, ¿Dónde se realiza la llamada a estos métodos?

Para ello solamente debemos llamar al método en la acción deseada, por ejemplo, al crear una sesión, tan solo debemos actualizar el usuario como *online* y mandar el mensaje correspondiente mediante el método publishUpdate:

```

1
2   //Cambio de estado a online
3   User.update(user.id, {online: true, longitude: long, latitude:
4     lat}, function (err){
5     if (err) return next(err);
6
7     //Informar a otros clientes (sockets abiertos) que el
8     //usuario esta logueado
9     User.publishUpdate(user.id, {
10       loggedIn: true,
11       id: user.id
12     );
13   });
14 }
```

Tras ello se notificará a todos los clientes que han sido suscritos a las instancias del modelo Usuario de que un determinado usuario ha cambiado su estado.

Del mismo modo para *publishCreate* o *publishDestroy*.

¿Y desde el lado del cliente? ¿Cómo realizamos las llamadas a las diferentes acciones de suscripción y cómo capturamos los eventos?

Para la realización de las suscripciones y captura de eventos las siguientes funciones ejecutables en el cliente resultan de vital importancia.

Primeramente tenemos la función *savesocket* la cual realiza la llamada al método *saveSocketID* del controlador *Session*. Dicho método crea un registro en base de datos ligando el identificador del socket abierto con el usuario que se encuentra identificado en el sistema, de tal manera que siempre sabremos qué socket corresponde con qué usuario. Este almacenamiento se debe a que al realizarse una nueva conexión de un usuario en el sistema el servidor debe conocer y registrar a dicho cliente con la finalidad de seguir su comportamiento, sus acciones y movimientos por toda la aplicación.

Dicho código JavaScript se encuentra localizado en el layout principal de la aplicación de tal manera que independientemente de qué vista resulte cargada siempre se ejecutará el siguiente fragmento de código realizando el almacenamiento del socket abierto por el usuario:

```

1
2
3 <script type="text/javascript">
4
5     $.when(savesocket()).done(function() {
6         if (typeof subscribeAndListen == 'function') {
7             subscribeAndListen();
8         }
9         listenMessages();
10    });
11
12     //Funcion para tener en todo momento almacenado en la tabla session
13     //los sockets conectados junto con el usuario al que pertenece
14     function savesocket() {
15         io.socket.get("/session/saveSocketID");
16     }
</script>
```

Función *saveSocketID*, la cual es llamada en cliente y que se encuentra implementada en el controlador *session* en el lado del servidor. Esta función realiza el registro de un nuevo cliente en la base de datos tras recibir el identificador correspondiente al nuevo socket creado.

El código inferior muestra la implementación en el lado del servidor de la función *saveSocketID*:

```

1
2
3 //Almacenamiento en la base de datos la sesión de cada usuario de la
4 //pagina
5 saveSocketID: function(req, res) {
6   if (!req.isSocket) return res.badRequest();
7
8   var socketId = sails.sockets.id(req);
9   // => "BetX2G-2889Bg22xi-jy"
10
11   var sessionObj = {
12     socket_id: socketId,
13     user_id: req.session.User != undefined ? req.session.User.id :
14       null
15   };
16
17   Session.create(sessionObj).exec( function (err, session) {
18     if (err) return res.badRequest();
19
20     console.log('\n.....');
21     console.log('Conectando a Sails js...');
22     console.log('Cliente conectado - id del socket: ' + socketId);
23     console.log('.....');
24
25     return;
26   });
27 }

```

Una vez tenemos almacenado el par socket-usuario en base de datos buscamos la función *subscribeAndListen*, la cual será específica según la vista en la que nos encontremos, por ejemplo, si nos encontramos en el panel de administración de usuarios, la función *subscribeAndListen* será específica para cada funcionalidad que queramos inicializar o a qué eventos nos deseamos suscribir. Si nos encontramos en la vista de índice de robots, la función se suscribirá a los eventos de conexión y desconexión de robots. Si estamos visualizando el funcionamiento de un robot la función se suscribirá a la sesión abierta para ese robot y capturará los diferentes comandos o datos transmitidos y respuestas del robot en cuestión.

En el código inferior mostramos la función *subscribeAndListen* que implementa la suscripción a los eventos correspondiente a la conexión, desconexión y borrado de usuarios del panel de administración descrito en el apartado 7.1. Esta función está localizada en el *partial*² que implementa la vista correspondiente con el citado panel:

```

1
2
3 <script type="text/javascript">
4

```

² Fragmentos del código HTML generados para romper el proceso de renderizado en bloques más manejables. Con un *partial*, puede establecer el código para representar por ejemplo una tabla HTML la cual es llamada en múltiples vistas.

```

5   function subscribeAndListen() {
6     io.socket.get('/user/user_subscribe');
7
8     io.socket.on('user', function (obj) {
9       if (obj.verb == 'created') {
10         var user = obj.data;
11         $.ajax({
12           url: '/user/render',
13           type: 'GET',
14           data: {id: user.id},
15           success: function(data){
16             $('#user_list').append(data);
17             toastr.info('<%= i18n("user_created") %>', 'RobotUI')
18           }
19         });
20       }
21
22       if (obj.verb == 'updated') {
23         var data = obj.data;
24         change_img_state(data.id, data.loggedIn, '<=%
25           i18n("connect")%>', '<=% i18n("disconnect")%>');
26
27         if (data.id != '<%= req.session.User.id %>') {
28           if (data.loggedIn) {
29             toastr.info('<%= i18n("user_connected") %>', 'RobotUI');
30           } else {
31             toastr.info('<%= i18n("user_disconnected") %>', 'RobotUI');
32           }
33           console.log('User has been updated to online:' +
34             data.loggedIn);
35         }
36       }
37
38       if (obj.verb == 'destroyed') {
39         $('#user_' + obj.id).remove();
40         toastr.info('<%= i18n("user_destroyed") %>', 'RobotUI');
41       }
42     });
43   }
44
45 </script>
```

Por otro lado se realiza la llamada a la función *listenMessages* ya que independientemente de la vista en la que nos encontremos siempre nos mantendremos a la escucha de los mensajes recibidos a nuestra bandeja de entrada por parte de otros usuarios.

```

1 //Evento a la espera de recibir mensajes y notificar al usuario
2 function listenMessages(){
3   io.socket.on('user', function messageReceived(message) {
4     switch (message.verb) {
5       case 'messaged':
6         var pathname = window.location.pathname;
```

```

8         if(pathname == '/message/index'){
9             location.reload();
10            }else{
11                new_msg_num_update('<%= i18n('messages') %>');
12            }
13            toastr.info('<%= i18n('new_message') %>' + '<a'
14                href="/message/index" > <%= i18n('open_here') %> </a> , '
15                'RobotUI');
16                break;
17            default:
18                break;
19        }
19    });

```

7.3. Desconexión

En este apartado describiremos los puntos de mayor relevancia a la hora de la desconexión de alguna de las diferentes comunicaciones establecidas comenzando con la desconexión de un usuario.

Cuando un usuario realiza una desconexión, bien deslogueándose de la página o cerrando una de las ventanas abiertas, automáticamente se lanza una llamada a la función *afterDisconnect* localizada en el fichero *config/Socket.js* del servidor. Función que será ejecutada cada vez que un socket es desconectado del sistema.

Dicha función primeramente localiza qué sesión en la base de relacionada está relacionada con identificador del socket desconectado. Si este socket estaba usando algún robot, éste se libera cambiando el estado del robot a *libre*. Y se informa a todos los sockets abiertos que el robot queda libreado y accesible al resto de usuarios.

```

1 Robot.update({id: session.robot_id}, {busy: false}, function
2     robotUpdated(err) {
3         if (err) return next(err);
4
5         //Informar a otros clientes (sockets abiertos) que el robot queda
6         //liberado
7         Robot.publishUpdate(session.robot_id, {
8             busy: false,
9             id: session.robot_id
10        });
10    });

```

A continuación, se comprueba si el usuario no dispone de más sockets abiertos. Si fuera el caso de que no los disponga, entonces se procede a cambiar su estado a desconectado y se emite o se informa al resto de clientes (sockets abiertos) que el usuario ya no se encuentra en el sistema. Llamada al método *User.publishUpdate*:

```
1 User.publishUpdate(session.user_id, {
2   loggedIn: false,
3   id: session.user_id
4 });
```

Se comprueba si el usuario desconectado se encontraba en alguna *room*. Pongamos como ejemplo que el usuario ha abandonado la visualización de un robot por lo queda se debe informar a todos los sockets integrantes de esa *room* que un usuario con un identificador determinado la ha abandonado con la función *sails.sockets.broadcast*. A continuación se muestra el fragmento de código:

```
1 //Emite a cada room que que se encuentre la sesión que un usuario la
2 //ha abandonado
3 session.rooms.forEach(function (room) {
4   sails.sockets.broadcast(room.room_name, {type: 'exit', msg:
5     {user_id: session.user_id}});
6});
```

Finalmente se destruye la sesión:

```
1 Session.destroy(session.id, function sessionDestroyed(err) {
2   if (err) return cb();
3   return cb();
4});
```

A continuación mostramos el código completo de la función *afterDisconnect*:

```
1
2 afterDisconnect: function(session, socket, cb) {
3   console.log('Disconnected - id del socket: ' + socket.id);
4
5   //Session del socket cerrado,
6   Session.findOne({socket_id:
7     socket.id}).populate('rooms').exec(function (err, session) {
8     if (err) return cb();
9     if (!session) return cb();
10
11     //Comprobar si el soscket estaba usando algun robot para
12     //liberarlo:
13     if (session.robot_id) {
14       console.log('Socked was using a robot');
15
16       Robot.update({id: session.robot_id}, {busy: false}, function
17         robotUpdated(err) {
18           if (err) return next(err);
19
20           //Informar a otros clientes (sockets abiertos) que el robot
21           //quedó liberado
22           Robot.publishUpdate(session.robot_id, {
23             busy: false,
24             id: session.robot_id
25           });
26         });
27     }
28   });
29
30   cb();
31 },
```

```

21      });
22    });
23  }
24
25
26 //Emite a cada room que que se encuentre la sesión que un
27 //usuario la ha abandonado ->
28 session.rooms.forEach(function (room) {
29   sails.sockets.broadcast(room.room_name, {type: 'exit', msg:
30     {user_id: session.user_id}});
31 });
32
33 //Comprobar si el usuario tiene más sockets abiertos:
34 Session.count({user_id: session.user_id}).exec(function
35   countUserSessions(error, n_sessions) {
36     console.log('There are ' + n_sessions + ' users ' +
37       session.user_id);
38
39     //Cambiemos el usuario a offline si solo tenía una ventana o
40     //conexión abierta.
41     if (n_sessions == 1) {
42       User.update(session.user_id, {online: false}, function (err)
43         {
44           if (err) return cb(err);
45
46           //Informar a otros clientes (sockets abiertos) que el
47           //usuario ya NO se encuentra logueado
48           User.publishUpdate(session.user_id, {
49             loggedIn: false,
50             id: session.user_id
51           });
52         });
53     }
54   });
55 });

```

En este punto ya conocemos cómo la aplicación RobotUI gestiona las diferentes conexiones y cómo se realiza el almacenamiento en base de datos de la información necesaria para determinar qué sockets se corresponde con qué usuarios en todo momento. Todo ello además ha sido explicado para la gestión de conexiones y desconexiones de usuarios notificando los cambios de estados entre *online* y *offline*.

Para la conexión y desconexión de los robots se aplica la metodología anterior de manera análoga. Pero... en el caso particular de los robots. ¿Cómo se realizan las comunicaciones entre el cliente (navegador) y el robot?, ¿Cómo es controlado un robot?, ¿Cómo se difunden los comandos lanzados a un robot al resto de usuarios espectadores?. En

los sucesivos puntos iremos describiendo cómo se ha implementado estos comportamientos.

7.4. Envío de comandos al robot

Cuando un usuario ha recibido la notificación de que un determinado dispositivo se encuentra disponible, entonces procede a acceder a su interfaz de control realizándose una conexión con el mismo. Una vez establecida dicha conexión el usuario puede proceder al lanzamiento de comandos para el control del robot. En ese momento se realiza un procedimiento para el envío de la información que consta de tres puntos:

1. Envío del comando indicado al robot.
2. Envío de notificación al servidor del comando lanzado.
3. Envío de la notificación por parte del servidor a todos los clientes suscritos al robot de que un comando ha sido enviado.

EL gráfico 7.5 muestra el flujo de datos generado cada vez que un comando es enviado al robot:

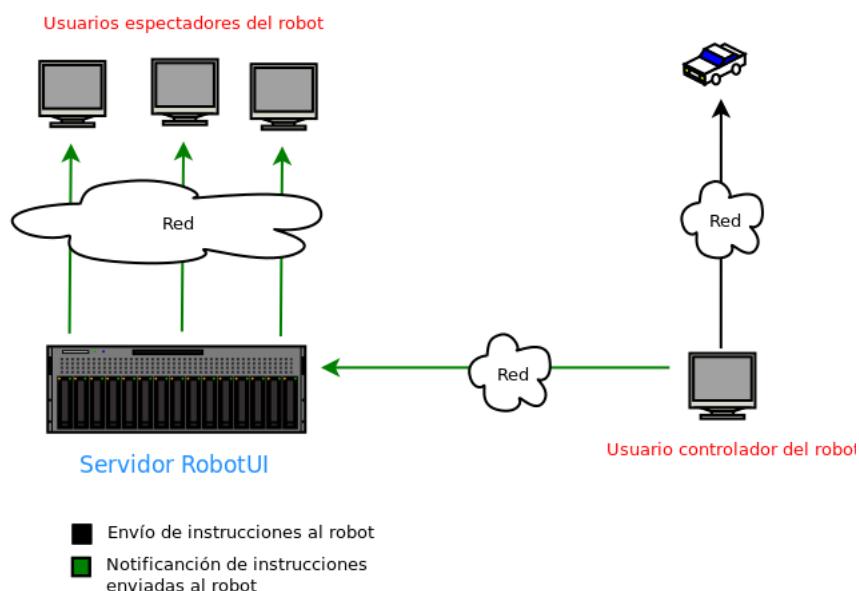


Figura 7.4: Esquema representativo del flujo de datos originado por un usuario controlador de un robot.

El código inferior realiza el envío del comando al robot para su interpretación y al servidor para su posterior difusión a los clientes suscritos:

```

1 //Presionar boton accion!
2 $('[id^=button_]').click(function (e) {
3   e.preventDefault();
4   var button_pressed_id = this.id.replace('button_custom_', '');
5   var code = document.getElementById(this.id).value
6
7   //Comunicacion al servidor del comando para difundir a los
8   //usuarios espectadores
9   io.socket.get('/interface/emit_action/', {robot: '<%=robot.id
10 %>', id: button_pressed_id, msg: code })
11
12 if (code != ''){
13   $('#chat').prepend('_' + code + '<br/>');
14   //Mandar instruccion al robot
15   socket.emit('action', code);
16   console.log('Emitting comand: ' + code );
17 }
});
```

7.5. Captura de datos del robot

Ahora bien, ya sabemos el procedimiento realizado a la hora de lanzar comandos al robot pero, esta comunicación tendría poco sentido si no capturamos sus respuestas. Estas respuestas pueden ser los datos correspondientes a los diferentes frames obtenidos por la cámara de vídeo del robot o aquellos datos correspondientes a la medición de los diferentes sensores de los que pueda disponer el robot.

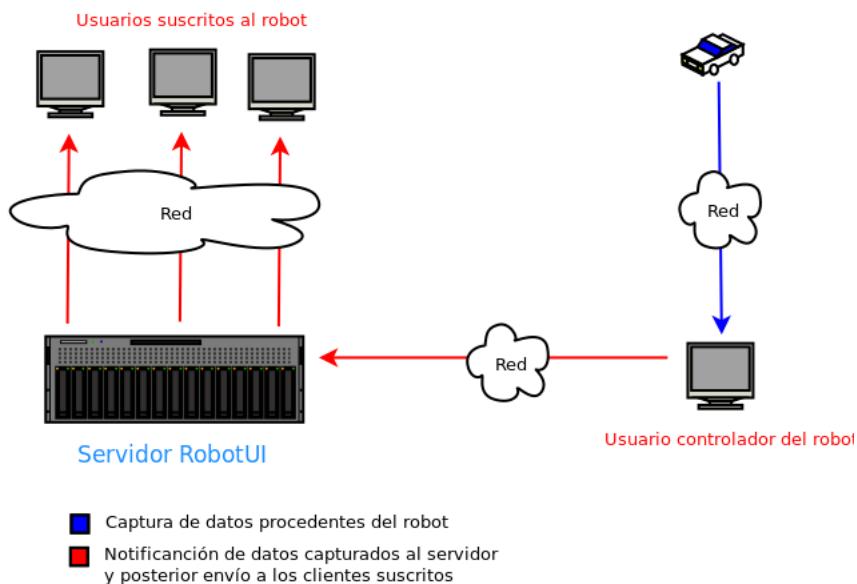


Figura 7.5: Esquema representativo del flujo de datos originado por un usuario tras la captura de datos procedentes del robot.

El código inferior muestra cómo se realiza la captura de los frames de vídeos procedentes del robot. Dicho código envía al servidor el frame para su posterior difusión con los clientes suscritos a robot y “dibuja” el frame capturado en el canvas de la interfaz de control del robot:

```
1 //Captura de video
2 socket.on('<%= video.event_name %>', function(data) {
3     io.socket.get('/interface/emit_video/', {robot: '<%=robot.id %>',
4         id: '<%= video.id %>', msg: data });
5     draw_image_to_canvas('play-<%= video.id %>', data)
}) ;
```

Código de la acción *emit_video* localizada en el servidor y llamada desde el cliente (código superior), la cual emite el frame capturado a todos los clientes que se encuentran suscritos al robot (broadcast a todos los clientes suscritos a la sala cuyo nombre se corresponde con el identificador del robot):

```
1 //Emision de las acciones a los visitantes de una interfaz
2 emit_video: function(req, res){
3     if (!req.isSocket) return res.badRequest();
4     sails.sockets.broadcast(req.param('robot'), {type: 'video', id:
5         req.param('id'), msg: req.param('msg')});
},
```

Capítulo 8

Análisis de requisitos

8.1. 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.

8.1.1. Chasis

El chasis consiste en una estructura interna que sostiene, aporta rigidez y da forma a un vehículo objeto en su construcción y uso. Es análogo al esqueleto de un animal. Para el caso que nos compete, consta de un armazón que integra entre sí y sujetta tanto los componentes mecánicos, como el grupo motopropulsor y la suspensión de las ruedas, motor 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 dispone de unas redas con tacos de considerable tamaño acompañadas y suspensiones. Este chasis debe facilitar el desplazamiento, mantenerlo en equilibrio en todo momento y que le permita cambiar de dirección fácilmente.

8.1.2. Tracción

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 [8.1.5](#) se encuentra información referente a las mismas.

8.1.2.1. 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 motor 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.

8.1.2.2. Servomotores

Los servomotores son dispositivos capaces de llevar el motor a posiciones angulares específicas al enviar una señal codificada. Mientras esta señal exista en la entrada, el motor mantendrá la posición angular del engranaje. 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 8.1: Servomotor utilizado.

Un servomotor está conformado por un motor, un circuito de control, un potenciómetro y un conjunto de engranajes. También potencia proporcional para cargas mecánicas. Un servo, por consiguiente, tiene un consumo de energía reducido.

La corriente que requiere depende del tamaño del servo. Normalmente el fabricante indica cuál es la corriente que consume. La corriente depende principalmente del par, y puede exceder un amperio si el servo está enclavado.

En otras palabras, un servomotor es un motor especial al que se ha añadido un sistema de control (tarjeta electrónica), 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 como aviones, barcos, helicópteros y trenes para controlar de manera eficaz los sistemas motores y los de dirección.

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.

8.1.3. 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+¹, 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 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.

¹ 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 [5] elaborado por uno sus creadores.

8.1.4. 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 puede suministrar a los motores es, en teoría, 2A por salida (hasta 3A de pico) y una tensión de alimentación de 3V a 35V. Sin embargo, el L298N tiene una eficiencia 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.

Estas pérdidas se disipan en forma de calor lo que se traduce 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 incorpora protecciones contra efectos que pueden producirse al manejar motores de corriente continua. Dispone de protecciones contra sobre intensidad, sobre temperatura, y diodos de protección contra corrientes inducidas.

El módulo seleccionado, el cual incorpora el controlador L298N, cuenta con todos los componentes necesarios para funcionar sin necesidad de elementos adicionales, entre ellos diodos de protección y 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.

En la figura 8.3 se muestra el módulo empleado con sus diferentes pines al detalle.

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.

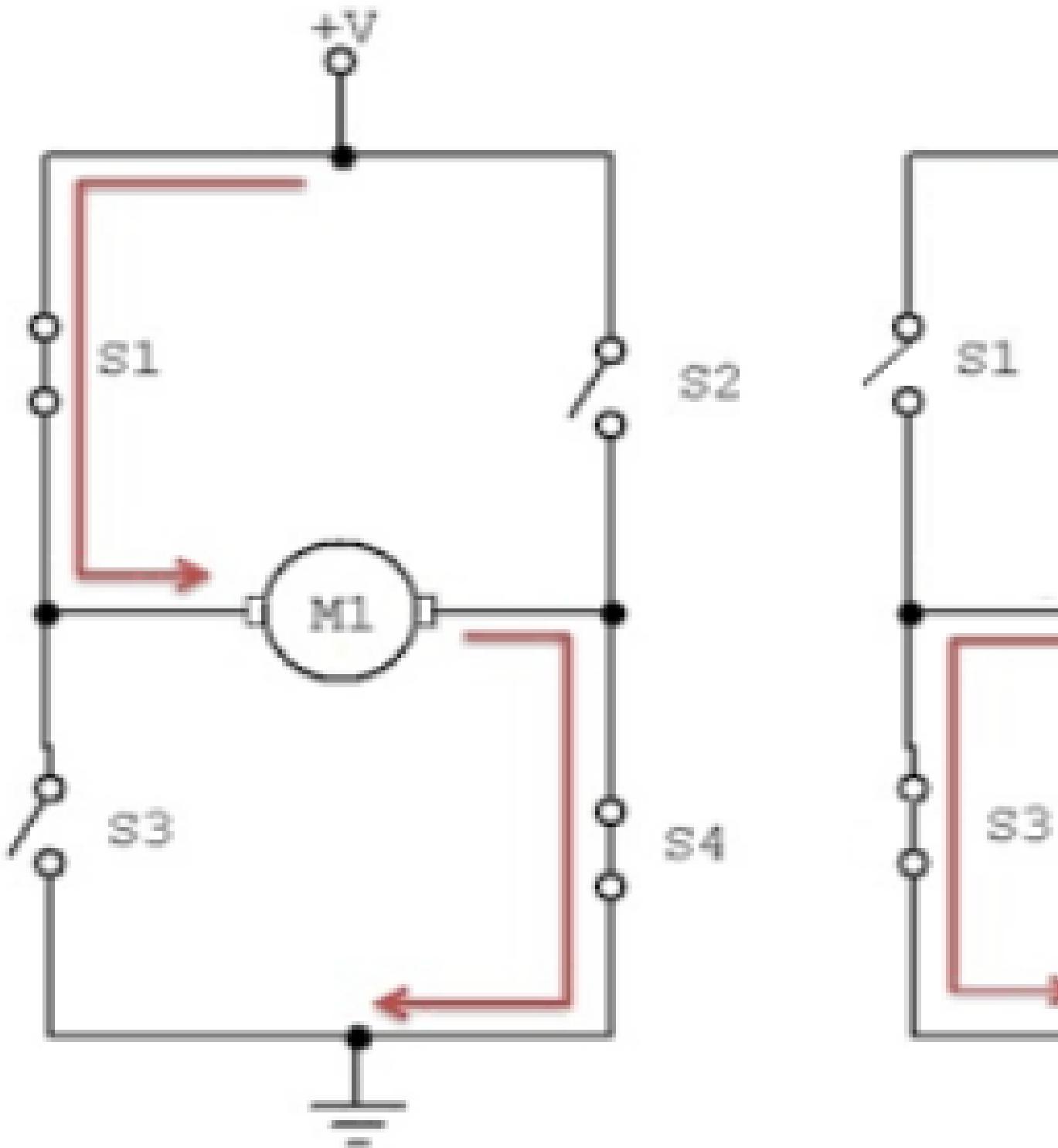


Figura 8.2: Funcionamiento del puente H.

El funcionamiento de un puente H es bastante simple. 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. Cuando se abren los interruptores S1 y S4 y se cierran S2

y S3, se invierte la tensión en bornes del motor, por lo que este gira en sentido contrario.

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

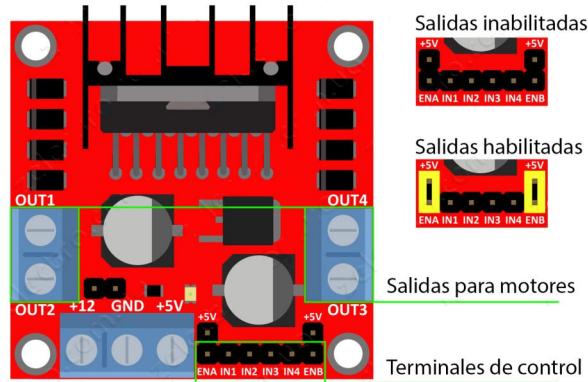


Figura 8.3: Pines de entrada/salida del módulo L298N empleado.

Out1, Out2, Out3 y Out4; son las salidas directas al motor, los motores Nema17 tienen 4 cables, los cuales se tienen que conectar aquí por orden. Si no se conectasen en orden, el funcionamiento del motor paso a paso no sería el adecuado. 2. IN1, IN2, IN3 y IN4; son los pines de entrada, se conectan a pines digitales de Arduino para poder controlar el motor. 3. ENA y ENB; son los enable , estos pines los dejaremos conectados con un jumper con los pines superiores de +5 V . Los ENA y ENB no se usaran en este proyecto ya que su función es para el control de motores DC. 4. La alimentación del driver es variable, ya que tiene un regulador de tensión de 5v. 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, pero recomendamos que el consumo no sea mayor a 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. [7] En este caso se utilizará una fuente de 12 V manteniendo el jumper activo, y así alimentar la parte de control del driver .

8.1.5. Alimentación

INTRO ALIMENTACION

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.

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.

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 separamos 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 [3.3.6](#).

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



Figura 8.4: Conjunto Raspberry Pi y módulo de expansión de alimentación.

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



Figura 8.5: Batería LiPo que alimenta los motores.

8.1.5.1. 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.

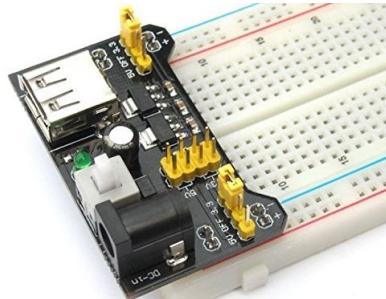


Figura 8.6: Adaptador de entrada USB a protoboard.

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 8.7 puede verse el esquema eléctrico del módulo.

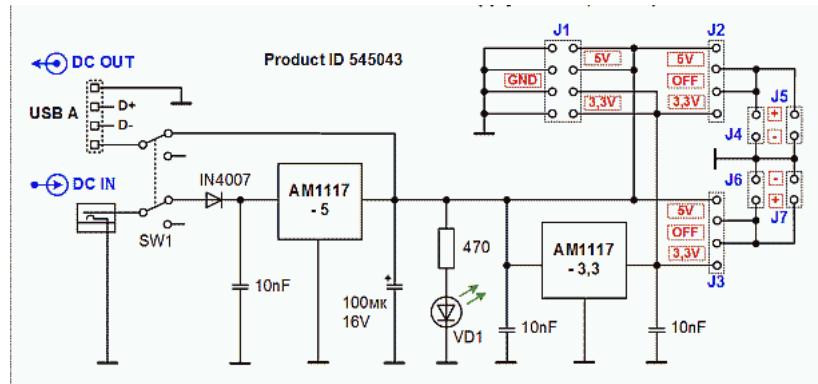


Figura 8.7: Adaptador de entrada USB a protoboard.

Siguiendo el esquema 8.7 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 pinos 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 ² 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 8.8: 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 jumper³ entre los pines en función de la tensión deseada.



Figura 8.9: Vista del conector J2 y J3.

²El diodo Zener es un diodo de silicio fuertemente dopado que se ha construido para que funcione en las zonas de rupturas, recibe ese nombre por su inventor Clarence Melvin Zener. El diodo Zener es la parte esencial de los reguladores de tensión casi constantes con independencia de que se presenten grandes variaciones de la tensión de red, de la resistencia de carga y temperatura.

³En electrónica y en particular en informática, un jumper o saltador es un elemento que permite cerrar el circuito eléctrico del que forma parte dos conexiones. Esto puede hacerse mediante soldadura (se derrite suficiente estaño para cerrar el circuito), soldando un cable o alambre entre ambos puntos o, lo más frecuente, conectando dos pines en hilera o paralelo mediante una pieza de plástico que protege el material conductor que cierra el circuito. Los más habituales tienen tamaños de 2,54 mm, 2 mm y 1,27 mm.

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

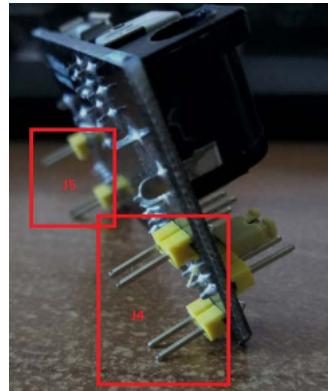


Figura 8.10: Vista del conector J4 y J5.

8.1.6. Interconexión entre módulos y fijación

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

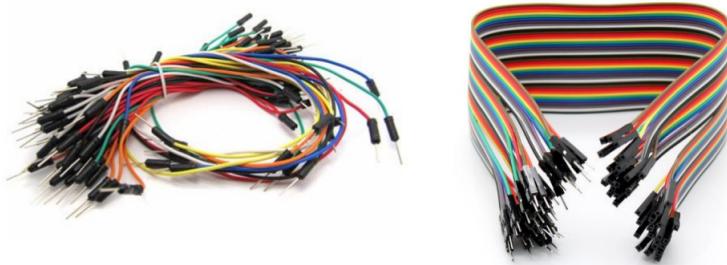


Figura 8.11: Cables de interconexión para protoboard.

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

8.1.7. Esquema de conexiones

El siguiente gráfico 8.12 muestra las conexiones de todo el conjunto:

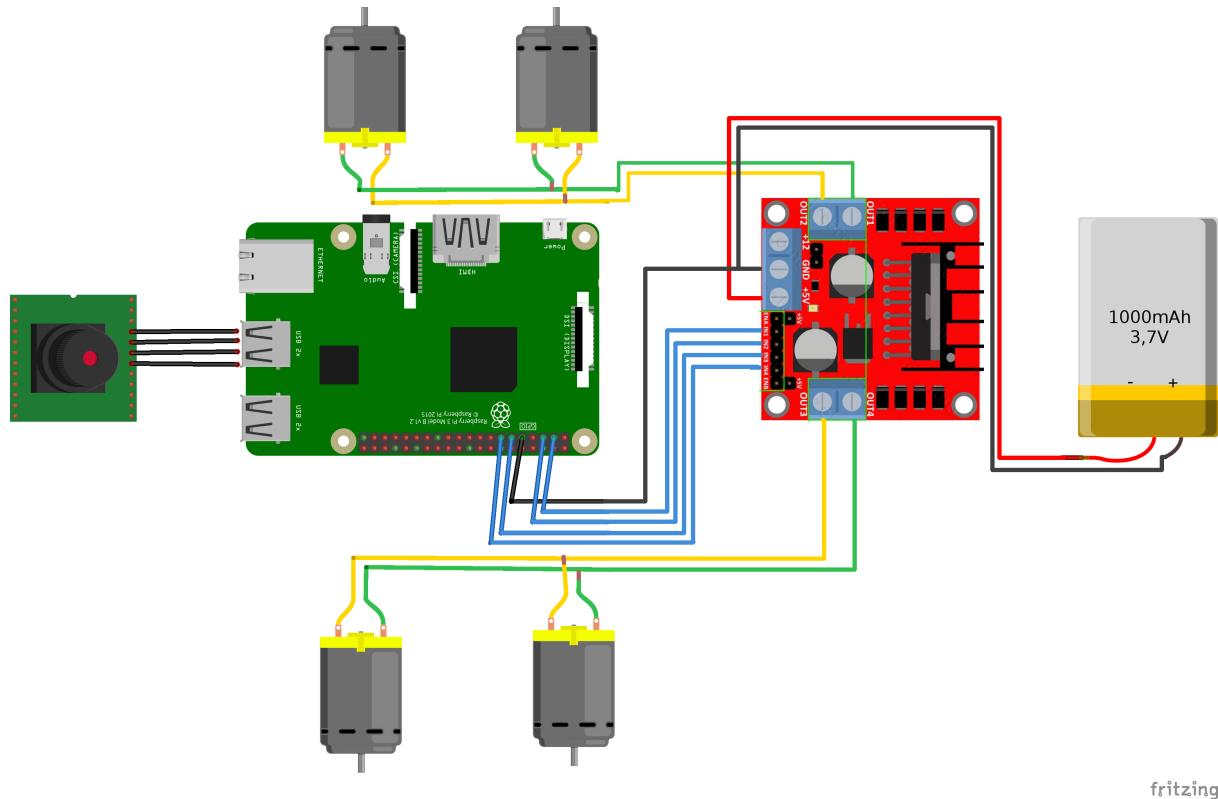


Figura 8.12: Esquema de conexiones del robot de pruebas.

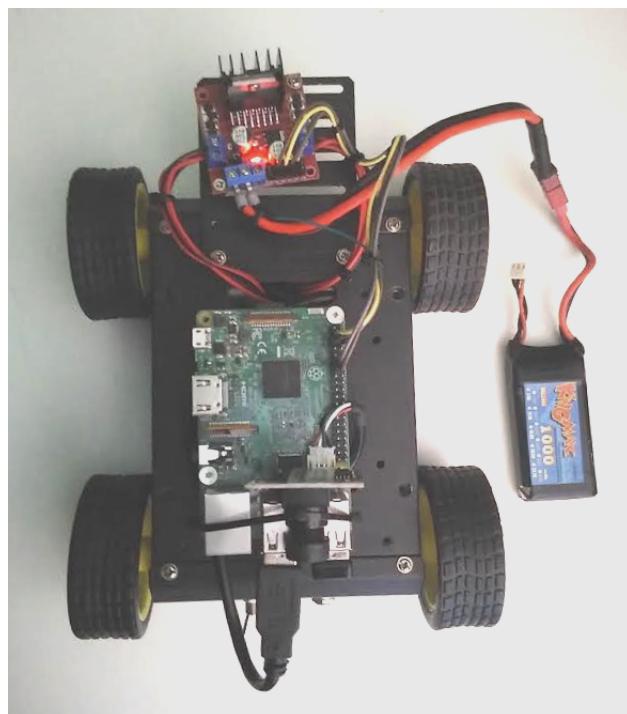


Figura 8.13: Vista superior del vehículo.

8.2. 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 8.14: Vista superior del vehículo.

8.2.0.1. 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.

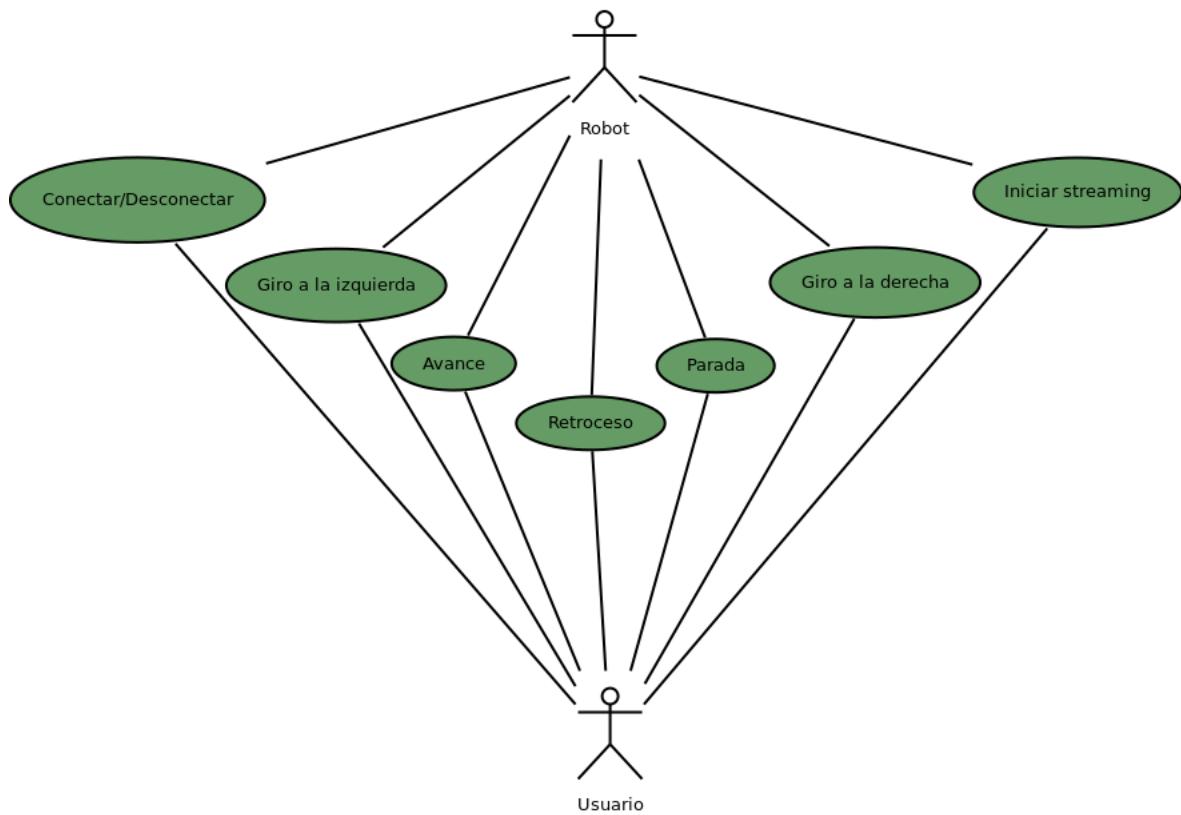


Figura 8.15: Diagrama de casos de uso para la interacción con el robot.

8.2.0.2. 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:

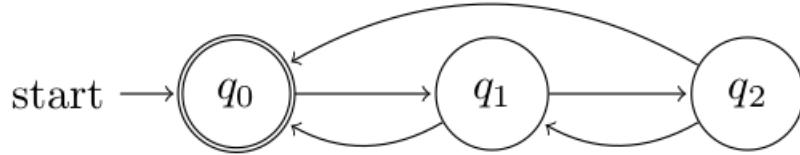


Figura 8.16: Autómata representativo de los diferentes estados del robot.

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.

8.3. Software de control

Para la programación del robot se ha empleado el lenguaje de programación JavaScript en un entorno de ejecución Node.js. A continuación se describirá aquellos aspectos más importantes referentes al código desarrollado para el control del robot.

Primeramente se ha realizado la carga de librerías necesarias, entre ellas encontramos:

- *pigpio*: Módulo para la comunicación y control de los pines GPIO.
- *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.
- *Arduino programming language* está basado en C++ y aunque la referencia para el lenguaje de programación de Arduino está en , también es posible usar comandos estandar de C++ en la programación de Arduino.

8.3.1. Entrada/Salida

En segundo lugar se han definido los diferentes pines GPIO a utilizar y si serán empleados como pines de entrada o de salida:

GPIO	Modo	Control
2	OUTPUT	Motores lado izquierdo
3	OUTPUT	Motores lado izquierdo
17	OUTPUT	Motores lado derecho
27	OUTPUT	Motores lado derecho

Tabla 8.1: Configuración establecida para los puertos GPIO.

Si para el vehículo que deseemos programar resultan necesarios más pines para la utilización de servos, sensores o cualquier otro elemento, 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>. Para el caso de este proyecto, la inicialización de los pines se ha realizado mediante las siguientes instrucciones:

```

1 // Carga del m dulo.
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});

```

8.3.2. Comunicaciones

En el punto anterior hemos visto las diferentes configuraciones de Entrada/Salida establecidas para el control de los diferentes motores y sensores. En el presente y sucesivos puntos describiremos los diferentes canales de comunicación abiertos y los flujos de información existentes. En la figura 8.17 se muestra un gráfico representativo de los diferentes flujos de datos existentes:

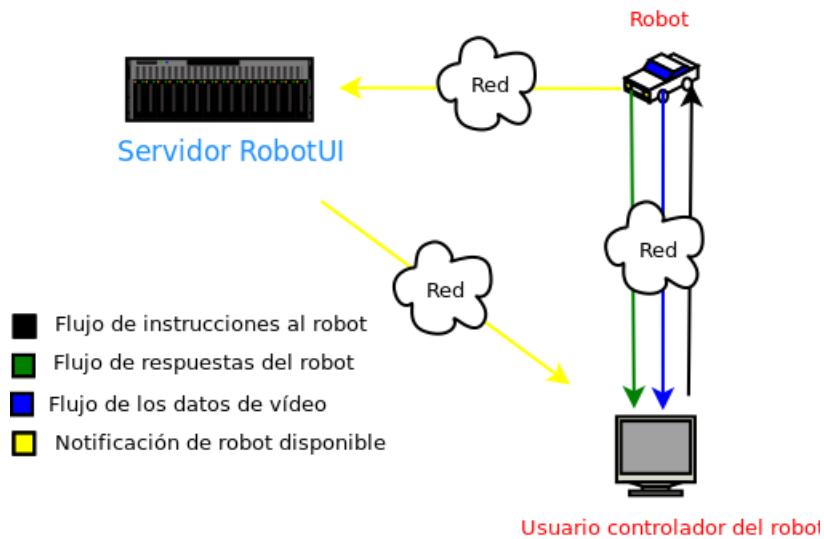


Figura 8.17: Canales de comunicación abiertos por el robot.

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 8.17.

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 8.17.

El usuario envía comandos al robot a través del canal abierto y las respuestas son enviadas a al cliente, transmisiones representados en verde para datos y en azul para el vídeo, figura 8.17.

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.

8.3.2.1. Sockets

Ahora bien, una vez definidos los pines que activarán nuestros motores y los canales de comunicación necesitamos que éstos sean activados cuando desde una red externa lo indiquemos 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 SDK⁴ 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 un robot que ha indicado a la aplicación web principal de 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;
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}
```

⁴Un 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.

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. La tabla 8.2 muestra las diferentes combinaciones de salidas y su acción correspondiente:

Acción	GPIO 2	GPIO 3	GPIO 17	GPIO 27
UP	1	0	1	0
DOWN	0	1	0	1
LEFT	1	0	0	0
RIGHT	0	0	1	0
STOP	0	0	0	0

Tabla 8.2: Combinaciones de salida para los puertos GPIO y su acción correspondiente.

A continuación se muestra el ejemplo desarrollado para la activación de los motores en sentido de giro y dirección según la tabla anterior:

```

1 // Escucha de comandos.
2 socket.on('action', function (data){
3
4     console.log('Comando recibido: ' + data);
5
6     switch(data) {
7         case 'UP':
8             gpio2.digitalWrite(1);
9             gpio3.digitalWrite(0);
10            gpio17.digitalWrite(1);
11            gpio27.digitalWrite(0);
12            console.log('UP');
13            break;
14
15        case 'RIGHT':
16            gpio2.digitalWrite(0);
17            gpio3.digitalWrite(0);
18            gpio17.digitalWrite(1);
19            gpio27.digitalWrite(0);
20            console.log('UP');
21            break;
22
23        case 'LEFT':
24            gpio2.digitalWrite(1);
25            gpio3.digitalWrite(0);
26            gpio17.digitalWrite(0);
27            gpio27.digitalWrite(0);
28            console.log('UP');
29            break;
30
31        case 'DOWN':

```

```

32     gpio2.digitalWrite(0);
33     gpio3.digitalWrite(1);
34     gpio17.digitalWrite(0);
35     gpio27.digitalWrite(1);
36     console.log('UP');
37     break;
38
39     case 'STOP':
40         gpio2.digitalWrite(0);
41         gpio3.digitalWrite(0);
42         gpio17.digitalWrite(0);
43         gpio27.digitalWrite(0);
44         console.log('UP');
45         break;
46
47     default:
48         console.log('command not found');
49     }
50 }
51 )

```

Diagrama de bloques del robot controlado vía WiFi:

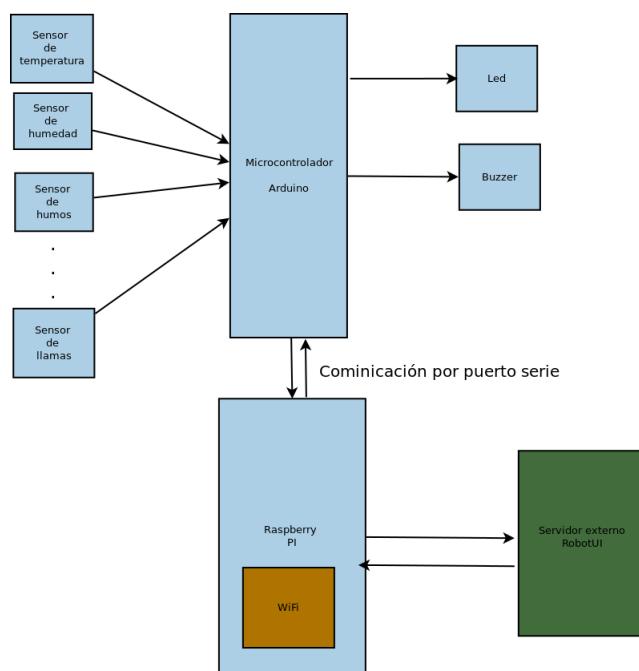


Figura 8.18: Diagrama de bloques del robot controlado por WiFi.

Componentes necesarios para el desarrollo del Robot controlado vía WiFi

- Microcontrolador
- Raspberry Pi con WiFi incorporado
- Sensor de temperatura

- Sensor de humedad.
- Sensor de sonidos de baja intensidad.
- Sensor de sonidos de elevada intensidad.
- Sensor de humos.
- Sensor de llamas.
- Otros sensores...
- Controladora de motores L289N.
- Cámara USB.
- Buzzer

8.3.2.2. Streaming de vídeo

Para realizar la transferencia de vídeo desde el robot hacia el cliente se ha empleado la librería FFmpeg⁵ 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 video4linux2⁶ (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 dispositivo está conectado al sistema, y tiene un nombre del tipo /dev/videoN, donde N es un

⁵ 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.

⁶ 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.

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 UNIX⁷.

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 buffer⁸ 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) {
3      //ffmpeg -f video4linux2 -i /dev/video0 -s 300x150 -f mjpeg pipe:1
4          -b:v 28k -bufsize 28k
```

⁷ 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.

⁸ 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.

```

5   if (running_camera == false){
6     console.log('Starting streaming....');
7     var args = ["-f", "video4linux2", "-i", "/dev/video0", "-s",
8       "300x150", "-f", "mpeg", "pipe:1", "-b:v 28k", "-bufsize 28k"]
9     ffmpeg_command = require('child_process').spawn("ffmpeg", args);
10    running_camera = true
11  }
12
13  ffmpeg_command.on('error', function(err, stdout, stderr) {
14    console.log("ffmpeg stdout:\n" + stdout);
15    console.log("ffmpeg stderr:\n" + stderr);
16    running_camera = false
17  });
18
19  ffmpeg_command.on('close', function (code) {
20    console.log('ffmpeg exited' + code );
21    running_camera = false
22  });
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}

```

8.3.2.3. Código de ejemplo completo

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

```

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   '59188631c8e94ba54f7a4bdc', online: true});
7
8 // Inicia servidor socket.io en el puerto 8085.
9 var io =io_client.listen(8085, { log: false });

```

```

10 // Carga de módulos necesarios.
11 var ffmpeg_command, running_camera = false, child_process =
12     require('child_process');
13
14 var Gpio = require('pigpio').Gpio;
15 // Pines utilizados. Motores izquierdos: 2 y 3, motores derechos: 17 y
16     27
17 var gpio2 = new Gpio(2, {mode: Gpio.OUTPUT}),
18     gpio3 = new Gpio(3, {mode: Gpio.OUTPUT}),
19     gpio17 = new Gpio(17, {mode: Gpio.OUTPUT}),
20     gpio27 = new Gpio(27, {mode: Gpio.OUTPUT});
21
22
23 console.log('Esperando conexión...');

24 var sockets = {};
25
26 io.sockets.on('connection', function (socket)
27 {
28
29     sockets[socket.id] = socket;
30     console.log("Clientes totales conectados: ",
31         Object.keys(sockets).length);
32
33     socket.on('disconnect', function() {
34         console.log('Adios !');
35         //stopStreaming(socket);
36     });
37
38     socket.on('start-stream', function() {
39         startStreaming(socket);
40     });
41     socket.emit('robotmsg', {msg: "Bienvenido !!!"});
42     console.log('emitiendo: ' + "Bienvenido !!!");
43
44     socket.on('action', function (data){
45
46         console.log('Comando recibido: ' + data);
47
48         switch(data) {
49             case 'UP':
50                 gpio2.digitalWrite(1);
51                 gpio3.digitalWrite(0);
52                 gpio17.digitalWrite(1);
53                 gpio27.digitalWrite(0);
54                 console.log('UP');
55                 break;
56
57             case 'RIGHT':
58                 gpio2.digitalWrite(0);
59                 gpio3.digitalWrite(0);
60                 gpio17.digitalWrite(1);
61                 gpio27.digitalWrite(0);

```

```

62         console.log('UP');
63         break;
64
65     case 'LEFT':
66         gpio2.digitalWrite(1);
67         gpio3.digitalWrite(0);
68         gpio17.digitalWrite(0);
69         gpio27.digitalWrite(0);
70         console.log('UP');
71         break;
72
73     case 'DOWN':
74         gpio2.digitalWrite(0);
75         gpio3.digitalWrite(1);
76         gpio17.digitalWrite(0);
77         gpio27.digitalWrite(1);
78         console.log('UP');
79         break;
80
81     case 'STOP':
82         gpio2.digitalWrite(0);
83         gpio3.digitalWrite(0);
84         gpio17.digitalWrite(0);
85         gpio27.digitalWrite(0);
86         console.log('UP');
87         break;
88
89     default:
90         console.log('command not found');
91     }
92
93 })
94 );
95
96 function stopStreaming(socket) {
97     delete sockets[socket.id];
98     // no more sockets, kill the stream
99     if (Object.keys(sockets).length == 0) {
100         if (ffmpeg_command){
101             ffmpeg_command.kill();
102             running_camera = false;
103             console.log('Stop streaming');
104         }
105     }
106 }
107
108 function startStreaming(socket) {
109     //ffmpeg -f video4linux2 -i /dev/video0 -s 300x150 -f mjpeg pipe:1
110     // -b:v 28k -bufsize 28k
111     if (running_camera == false){
112         console.log('Starting streaming....');
113         var args = ["-f", "video4linux2", "-i", "/dev/video0", "-s",
114             "300x150", "-f", "mjpeg", "pipe:1", "-b:v 28k", "-bufsize 28k"]
115         ffmpeg_command = child_process.spawn("ffmpeg", args);
116     }

```

```

115     running_camera = true
116 }
117
118 ffmpeg_command.on('error', function(err, stdout, stderr) {
119   console.log("ffmpeg stdout:\n" + stdout);
120   console.log("ffmpeg stderr:\n" + stderr);
121   running_camera = false
122 });
123
124
125 ffmpeg_command.on('close', function (code) {
126   console.log('ffmpeg exited' + code );
127   running_camera = false
128 });
129
130
131 ffmpeg_command.stderr.on('data', function (data) {
132   //console.log('stderr: ' + data);
133 });
134
135 ffmpeg_command.on('end', function() {
136   console.log('Fin');
137   running_camera = false
138 });
139
140 ffmpeg_command.stdout.on('data', function (data) {
141   //console.log('stdout: ' + data);
142   var frame = new Buffer(data).toString('base64');
143   socket.emit('canvas',frame);
144 });
145
146 }
```

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 9

Pruebas

La realización de pruebas de software, es una de las fases del ciclo del software y permiten la identificación de posibles fallos en la implementación, calidad o usabilidad de la aplicación.

9.1. Plan de pruebas

Debido a la arquitectura de RobotUI y a la metodología basada en desarrollo incremental que se ha seguido en el desarrollo, las diferentes partes han ido siendo probadas de forma independiente. Aun así, la aplicación ha sido sometida a las siguientes pruebas:

Pruebas de configuración:

- Se controla que el sistema de gestión de permisos funciona correctamente.

Pruebas de navegación:

- Se comprueba que se realiza una navegación correcta por toda la aplicación.

Pruebas de interfaz:

- Se comprueba que los estilos de la aplicación son uniformes y correctos.
- Se comprueba que se actualiza de forma satisfactoria la imagen de estado de un usuario, *online - offline*.
- Se comprueba que se actualiza de forma satisfactoria la imagen de estado de un robot, *libre - ocupado* y *online - offline*.
- Se comprueba que se actualiza de forma satisfactoria el panel de usuarios realizando el seguimiento de un robot, (entrada y salidas de usuarios de la sesión).

Pruebas de funcionamiento:

- Se controla el estado *online* u *offline* de un robot.
- Se controla el estado *libre* u *ocupado* de un robot.

- Se controla el estado *online* u *offline* de un usuario.
- Se realiza el control de un robot correctamente.
- Se realiza el seguimiento de un robot correctamente.
- Se realiza la configuración de una interfaz para un robot determinado.
- Se realiza el registro de un robot correctamente.

Capítulo 10

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.

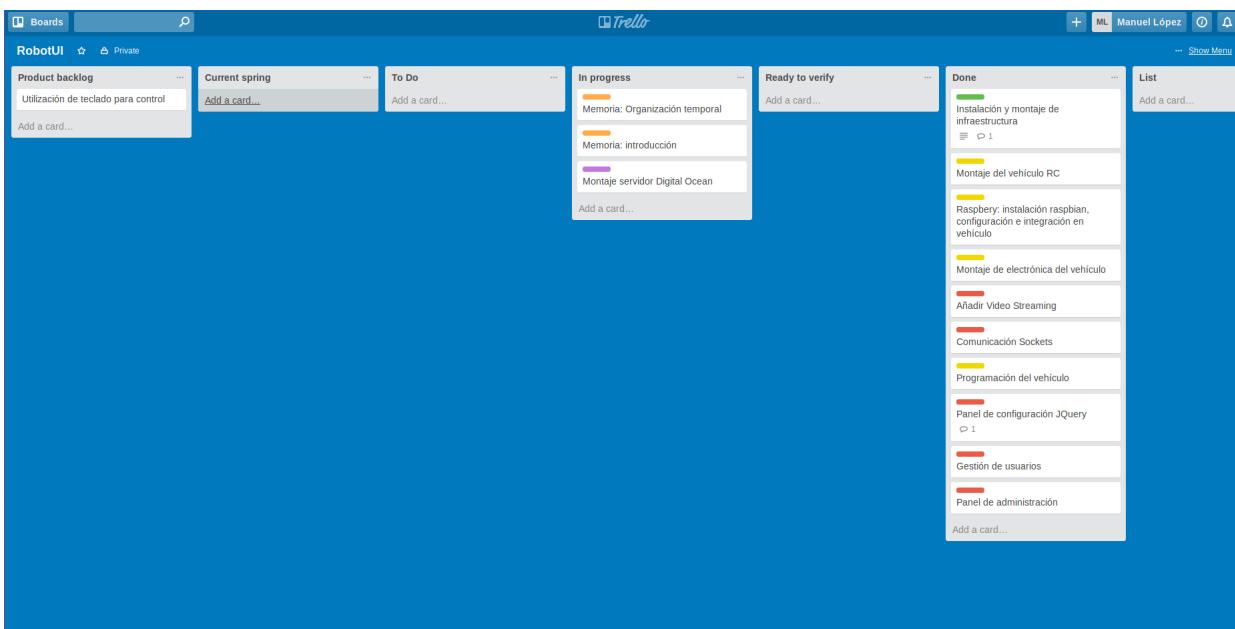


Figura 10.1: Panel de actividades - Trello

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 [10.2](#) y [10.3](#) muestran 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 10. ORGANIZACIÓN TEMPORAL

WBS	Nombre	Trabajo
1	▼ RobotUI	135d 3h
1.1	Conocimiento del proyecto	22d
1.2	Planificación y estudio del proyecto	5d
1.3	▼ Análisis de herramientas existentes	17d
1.3.1	Estudio de Node.js	5d
1.3.2	Estudio de Sails.js	5d
1.3.3	Estudio de Socket.io	2d
1.3.4	Códigos y pruebas	3d
1.3.5	Estudio de MongoDB	2d
1.4	▼ Definición de requisitos	5d 6h
1.4.1	Arquitectura BBDD análisis	1d
1.4.2	Diseño de diagrama UML	2d
1.4.3	Requisitos funcionales	1d
1.4.4	Requisitos no funcionales	6h 45min
1.4.5	Reunión de planificación con director del proyecto	1d
1.5	▼ Desarrollo aplicación	30d 4h
1.5.1	Sistema de autenticación - registro	2d
1.5.2	Alta de dispositivos robóticos	2d
1.5.3	Internacionalización	1d
1.5.4	Módulo de mensajes entre usuarios	3d
1.5.5	Implementación de políticas de permisos	4d
1.5.6	Reunión de seguimiento con director del proyecto	3d 3h
1.5.7	▼ Elementos de la interfaz	5d 5h
1.5.7.1	Acciones	1d 1h
1.5.7.2	Sliders	1d
1.5.7.3	Labels	1d 7h
1.5.7.4	Video	1d 4h
1.5.8	▼ Personalización de la interfaz	9d 4h
1.5.8.1	Personalización de acciones	1d 4h
1.5.8.2	Personalización de sliders	2d
1.5.8.3	Personalización de labels	2d
1.5.8.4	Edición de la interfaz	4d
1.6	▼ Módulo de comunicaciones	18d
1.6.1	▼ Conexión cliente - robot	4d 7h
1.6.1.1	Envío de órdenes al robot	2d
1.6.1.2	Captura de vídeo del robot	2d 7h
1.6.2	▼ Conexión cliente - servidor	3d
1.6.2.1	Envío de vídeo capturado al servidor	1d 4h
1.6.2.2	Envío de órdenes lanzadas al servidor	1d 4h
1.6.3	Desarrollo de tests funcionales	8d
1.6.4	Frontend - detalles de estilos	2d

Figura 10.2: Descomposición de las tareas implicadas en la construcción del vehículo robótico SensorRS (Primera Parte).

1.7	▼ Conexión servidor - cliente	4d
1.7.1	Difusión de vídeo a espectadores	2d
1.7.2	Difusión de comandos a espectadores	2d
1.8	▼ Montaje del vehículo	3d
1.8.1	Búsqueda de elementos hardware	2d
1.8.2	Montaje y conexiones	1d
1.9	▼ Programación del vehículo	4d 7h
1.9.1	Gpio	2d
1.9.2	Video streaming	1d 3h
1.9.3	Fase de pruebas	1d 4h
1.10	Despliegue de la aplicación en producción	1d
1.11	▼ Documentación	23d
1.11.1	Memoria	19d
1.11.2	Resumen	2d
1.11.3	Presentación	2d
1.12	Reunión de seguimiento con el director del proyecto	1d

Figura 10.3: Descomposición de las tareas implicadas en el desarrollo del proyecto (Segunda parte).

10.1. 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:

10.1.1. 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 eventos en tiempo real.

10.1.2. Hito 1: 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.

10.1.3. Hito 2: 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. Alimentación de todo el conjunto.

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

10.1.4. Hito 3: 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.

10.1.5. Hito 4: 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 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.

10.1.6. Hito 5: 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.

10.2. Diagrama de Gantt

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

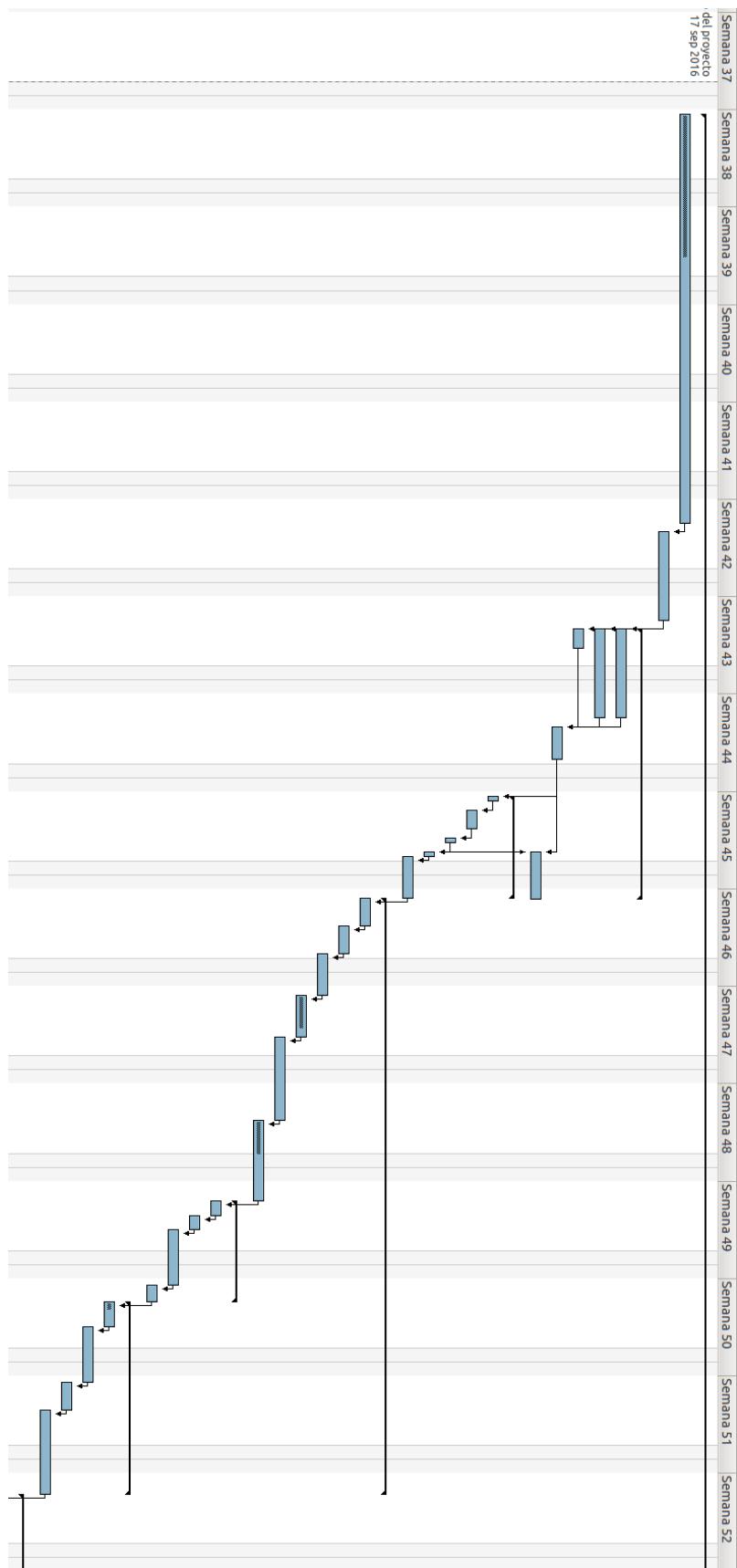


Figura 10.4: Diagrama de Gantt 1. Desarrollo del proyecto.

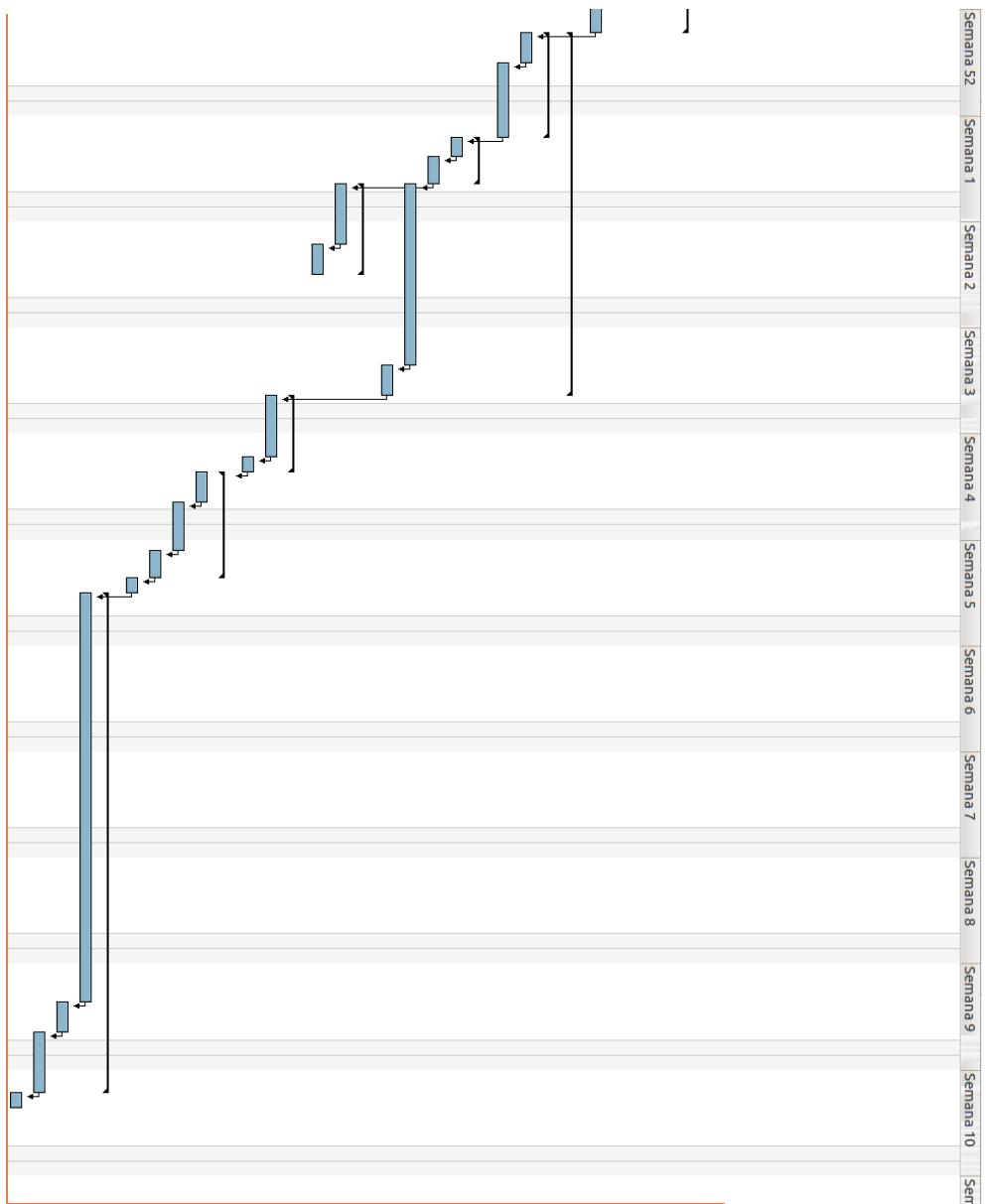


Figura 10.5: Diagrama de Gantt 2. Desarrollo del proyecto.

Capítulo 11

Guía de Usuario

La aplicación RobotUI ha sido creada por Manuel López Urbina para el proyecto fin de carrera de la titulación de Ingeniería Informática de la Universidad de Cádiz.

Resulta de vital importancia consultar esta guía antes y/o durante la utilización de los diferentes elementos tanto hardware, robot de pruebas desarrollado, como software del presente proyecto ya que le proporcionará una guía paso a paso en el manejo correcto de la aplicación.

La resolución recomendada para la aplicación debe ser superior o igual a 1024x768 (Estándar XGA), aunque es adaptable a cualquier resolución debido a su diseño web responsive.

RobotUI ha sido elaborado con un claro propósito; el de proporcionar a los usuarios un medio donde compartir sus dispositivos robóticos 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 de la aplicación, en este manual se describen los diferentes pasos a realizar para configurar sus dispositivos correctamente en el sistema y abrirlo a toda una comunidad de usuarios. Además de tener abierto el acceso a otros muchos dispositivos de otras personas.

11.0.1. Objetivo de esta guía

Esta guía tiene como objetivo proporcionar al usuario un soporte de ayuda e iniciación a la utilización de RobotUI.

Esta sección comprende:

- Introducción.
- Guía de acceso al código fuente de la aplicación.
- Guía de uso de la aplicación.
- Guía para la puesta en marcha y programación de un robot.

11.0.2. 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 de los diferentes dispositivos robóticos en sus dos modalidades disponibles, la de control y la de visualización.

11.0.3. 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
```

11.1. Uso de SensorRS

11.1.1. 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.com.

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.

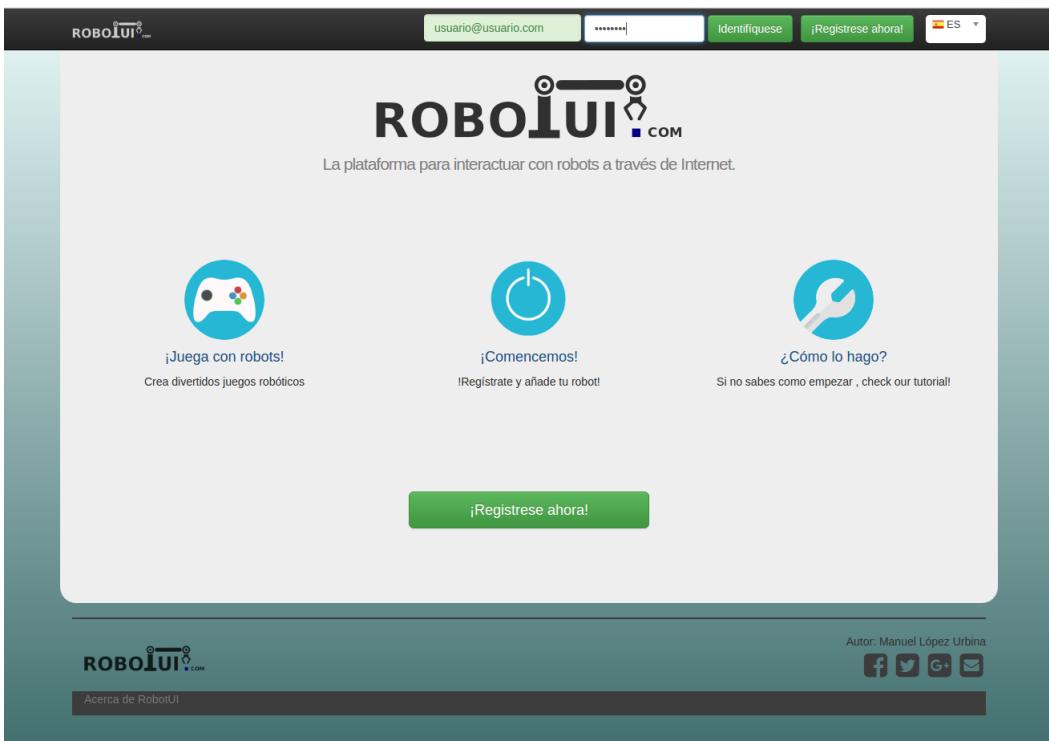


Figura 11.1: Página principal RobotUI.

11.1.2. 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, la cual emplea para la conexión de sensores y motores haciendo uso de su sistema de Entrada/Salida Gpio.

El sistema es compatible con cualquier dispositivo, no solo limitado a las 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.

Antes de proceder con la programación de nuestro robot debemos de realizar su creación en la aplicación RobotUI como queda descrito en el punto ?? y tomar nota del identificador único que la aplicación proporciona para dicho robot y que posteriormente necesitaremos.

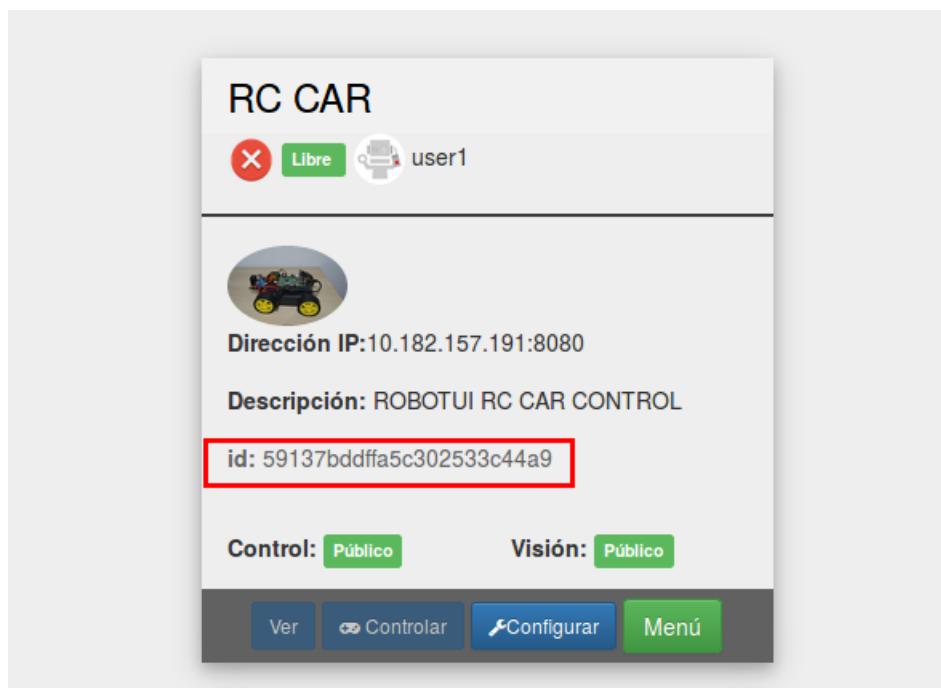


Figura 11.2: Panel informativo de un robot donde se aprecia su identificador.

Una vez creado el robot y configurada su interfaz, añadiendo los diferentes elementos disponibles como acciones, vídeo, etiquetas, etcétera, recogidos en el punto ??: 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 [11.1.2.1](#).
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*.

11.1.2.1. 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:

```

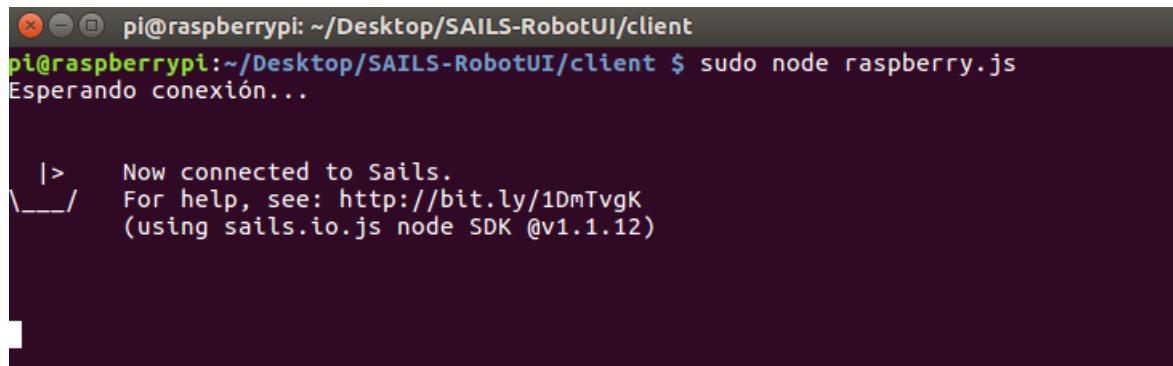
1   socket.on('action', function (data){
2     if (data == 'READ') {
3       var temp = gpio2.digitalRead(1);
4       socket.emit('robot_temp', {msg: temp});
5     }
6   });
7 
```

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 11.3: Robot a la espera de conexión entrante.

Capítulo 12

Comentarios finales

12.1. 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
Lipo batería (3.7v, 600mAh Lipo)	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
Horas de programación y montaje	108 350 horas	50/hora	17500

Total bruto: 17745,38 €

I.V.A. %: 21 %

Total presupuesto: 21471,91 €

12.2. Conclusiones

La elaboración de este proyecto 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 roboótico haciendo uso de una Raspberry Pi 3 Model B.
- Programación del micrnocontrolador 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 proyecto 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 peligrosa 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.

12.3. 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 otras posibilidades de conectividad distintas al WiFi como 4G, bluetooth, etc.
- Incorporar una cámara con posibilidad de visión 360 grados.
- 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. Entorno de desarrollo

El entorno de desarrollo es extremadamente sencillo ya que el programa se escribe en la zona del sketch y para compilarlo y cargarlo a cualquier placa solo es necesario pulsar un botón.

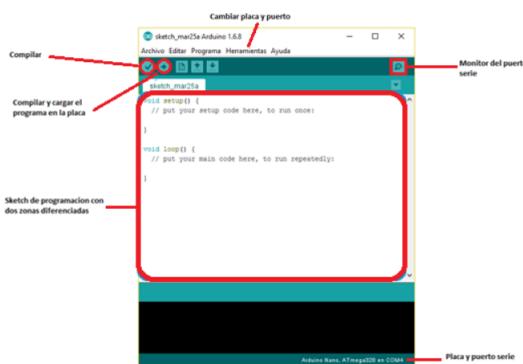


Figura 1: Entorno de desarrollo de Arduino.

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.

.2. 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
```

.3. 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.

.3.1. 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 esenciales necesarios.

Bibliografía

- [1] Andrew Lombardi. *WebSocket: Lightweight Client-Server Communications*. O'Reilly Media, 2012.
- [2] Mike Cantelon, Alex R. Young, Marc Harter, T.J. Holowaychuk, and Nathan Rajlich. *Node.js in Action*. Manning Publications, 2017.
- [3] Douglas Crockford. *JavaScript: The Good Parts*. O'Reilly, 2008.
- [4] Digital Ocean. Digital Ocean. <https://www.digitalocean.com/>. Visitado el 15-06-2017.
- [5] Eben Upton and Gareth Halfacree. *Raspberry Pi User Guide*. Rasoberry Pi, 2013. <http://www.cs.unca.edu/~bruce/Fall14/360/RPiUsersGuide.pdf>, visitado el 10-04-2017.
- [6] Frantisek Korbel. *FFmpeg Basics: Multimedia handling with a fast audio and video encoder*. CreateSpace, 2015.
- [7] Git Hub. Git Hub. <https://github.com>. Visitado el 03-07-2017.
- [8] Pablo Hinojoda and JJ Merelo. *Aprende Git: ... y, de camino, GitHub*. Editorial Reviews, 2015.
- [9] Kristina Chodorow. *MongoDB: The Definitive Guide, 2nd Edition*. O'Reilly Media, 2013.
- [10] Irl Nathan Mike McNeil. *Sails.js in Action*. Manning Publications, 2017.
- [11] Ministerio de Administraciones Pùblicas, Gobierno de Espaùa. Métrica v3. http://administracionelectronica.gob.es/pae/Home/pae_Documentacion/pae_Metodolog/pae_Metrica_v3.html#.WQ79Wic1GkB. Visitado el 10-01-2017.
- [12] Irl Nathan. Activityoverlord, an application to learn sails.js. <https://github.com/irlnathan/activityoverlord>. Visitado el 19-01-2017.
- [13] Official documentation. Node JS Documentation. <https://nodejs.org/es/docs/>. Visitado el 02-05-2017.
- [14] Official documentation. Sails JS Documentation. <https://sailsjs.com/documentation/reference>. Visitado el 14-03-2017.
- [15] Rohit Rai. *Socket.IO Real-Time Web Application Development*. Packt, 2013.

- [16] Sails.js. Sails.js | Realtime MVC Framework for Node.js. <http://sailsjs.com/>. Visitado el 27-06-2017.
- [17] Sandro Pasquali. *Deploying Node.js*. Packt Publishing, 2015.
- [18] 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.
- [19] 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.
- [20] Stefan Kottwitz. *LaTeX Beginner's Guide*. Packt Publishing, 2011.
- [21] Dr. Ovidiu Vermesan and Dr. Peter Friedss. *Internet of Things - From Research and Innovation to Market Deployment*. River Publishers Aalborg, 2014.
- [22] Wikibooks. The Book of LaTeX. <http://en.wikibooks.org/wiki/LaTeX>. Visitado el 08-06-2017.
- [23] 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.