



Grado en Ingeniería Telemática

Escuela Técnica Superior de Ingeniería de Telecomunicación

Curso académico 2015-2016

Trabajo Fin de Grado

TECNOLOGÍAS WEB EN PLATAFORMA ROBÓTICA JDEROBOT

Autor: Aitor Martínez Fernández

Tutor: José María Cañas Plaza

Madrid 2015

*A mis padres y hermano,
que siempre han estado a mi lado,
y siempre lo estarán.*

*A Laura, que siempre
está a mi lado apoyándome.*

*Y a mis amigos, por ser
verdaderamente mis amigos.*

Gracias.

Índice general

1. Introducción	10
1.1. Robótica	10
1.1.1. ROS	13
1.1.2. JdeRobot	13
1.2. Tecnologías Web	15
1.2.1. HTTP	16
1.2.2. HTML	17
1.2.3. JavaScript	19
1.3. Tecnologías web en middlewares robóticos	21
1.3.1. The Robot Management System	21
1.3.2. Surveillance 4.0 (URJC)	22
1.3.3. Surveillance 5.1 (URJC)	22
2. Objetivos y Metodología	26
2.1. Descripción del problema	26
2.2. Requisitos	27
2.3. Metodología y plan de trabajo	27
3. Plataforma de Desarrollo	30
3.1. Simulador Gazebo	30
3.2. JdeRobot	31
3.2.1. CameraServer + CameraView	31

ÍNDICE GENERAL	4
3.2.2. Openni1Server + RgbdViewer	32
3.2.3. Ardrone_server + uav_viewer	32
3.2.4. Kobuki_driver + kobukiViewer	33
3.3. HTML5	34
3.3.1. Canvas	35
3.3.2. WebWorker	36
3.3.3. WebSocket	37
3.4. CSS3	37
3.5. JavaScript6	38
3.5.1. Promise	38
3.6. WebGL	39
3.7. ThreeJS	40
3.8. JQuery	41
3.9. Bootstrap	42
3.10. ICE for Javascript	43
4. Diseño y programación	45
4.1. Diseño global	45
4.1.1. Servidores	47
4.1.2. Clientes	48
4.2. CameraViewJS	49
4.2.1. Conector	50
4.2.2. Núcleo	53
4.2.3. Interfaz gráfico	54
4.3. RgbdViewerJS	59
4.3.1. Núcleo	60
4.3.2. Interfaz gráfico	64
4.4. KobukiViewerJS	66
4.4.1. Conectores	68

ÍNDICE GENERAL	5
4.4.2. Núcleo	71
4.4.3. Interfaz gráfico	76
4.5. UavViewerJS	80
4.5.1. Conectores	82
4.5.2. Núcleo	83
4.5.3. Interfaz gráfico	86
4.6. IntrorobKobukiJS	90
4.6.1. Núcleo	90
4.6.2. Interfaz gráfico	93
4.7. IntrorobUavJS	93
4.7.1. Núcleo	94
4.7.2. Interfaz gráfico	95
4.8. JdeRobotWebClients	95
5. Experimentos	98
5.1. Experimentos con CameraViewJS	98
5.2. Experimentos con RgbdViewerJS	100
5.3. Experimentos con KobukiViewerJS	101
5.4. Experimentos con UavViewerJS	102
5.5. Experimento con IntrorobKobukiJS	103
5.6. Experimentos con IntorobUavJS	105
5.7. Rendimiento Temporal	109
5.7.1. Todos los elementos dentro de una red local	110
5.7.2. Servidor en el domicilio y los clientes en otras redes	111
5.7.3. Servidor en la universidad y los clientes en otras redes	111
6. Conclusiones	114
6.1. Conclusiones	114
6.2. Trabajos futuros	115
Bibliografía	117

Índice de figuras

1.1.	Baxter en una cadena de embalaje	11
1.2.	Roomba de iRobot (a) y Aibo de Sony (b)	11
1.3.	GoogleCar (a) y Robots de Amazon (b)	12
1.4.	Pepper (a) y Hotel robótico (b)	12
1.5.	Uav Viewer (a) e Introrob (b)	14
1.6.	Google Maps (a) y Facebook (b)	16
1.7.	El País (a) y Netflix (b)	16
1.8.	Comunicación cliente servidor en HTTP.	17
1.9.	Ejemplo de código HTML.	18
1.10.	Menú de RMS (a) y Cámara a través de RMS (b)	22
1.11.	interfaz (a) y arquitectura (b)	24
1.12.	interfaz (a) y arquitectura (b)	25
2.1.	Modelo en espiral	28
3.1.	Interfaz gráfica del simulador Gazebo.	31
3.2.	Ejemplo del uso del componente CameraServer.	32
3.3.	Ejemplo del uso del componente Openni1Server.	33
3.4.	Ejemplo del uso del componente Ardrone_server.	33
3.5.	Ejemplo del uso del Kobuki.	34
3.6.	Vídeo en Youtube.	35
3.7.	Elementos WebWorker.	36
3.8.	Vida de una Promise	39

3.9. Ejemplo mundo WebGL con Three.js.	40
3.10. Web de la NASA.	43
3.11. Esquema de funcionamiento de Ice-JS.	44
4.1. Arquitectura de JdeRobotWebClients	46
4.2. Niveles de los clientes	47
4.3. Arquitectura de CameraViewJS	49
4.4. Mensajes API.Camera.	52
4.5. Escritorio (a) y Móvil (b)	54
4.6. Header y Body (a) y Modal (b)	55
4.7. Arquitectura de RgbdViewerJS	59
4.8. Creación de la escena 3D	60
4.9. Modelo de cámara Pin Hole	61
4.10. Interfaz (a) y Modal (b)	64
4.11. Body de RgbdViewerJS	64
4.12. Visualización con cámaras de observación de frente (a) y a la derecha (b) .	67
4.13. Arquitectura de KobukiViewerJS	67
4.14. Mensajes API.Motors	70
4.15. Control (a) y Modelo 3D (b)	72
4.16. Láser en 2D (a) y 3D (b)	76
4.17. Interfaz (a) y Modal (b)	77
4.18. Body de KobukiViewerJS	77
4.19. sin (a) y con (b) Zoom	80
4.20. Arquitectura de UavViewerJS	81
4.21. Botones de funciones del drone	84
4.22. Modelo 3D (a) e indicadores de vuelo (b)	84
4.23. Interfaz (a) y Modal (b)	86
4.24. Body de UavViewerJS	86
4.25. Arquitectura de IntrorobKobukiJS	90

ÍNDICE DE FIGURAS

8

4.26. Interfaz de IntrorobKobukiJS	93
4.27. Arquitectura de IntrorobUavJS	94
4.28. Interfaz de IntrorobUavJS	96
4.29. Escritorio (a) y Móvil (b)	96
5.1. Equipos usados en este experimento y su función.	99
5.2. Cliente 1 (a) y Cliente 2 (b)	99
5.3. Equipos usados en estas pruebas y su función.	100
5.4. Ejecución de cliente	101
5.5. Equipos usados en este experimento y su función.	102
5.6. Simulado (a) y Real (b)	102
5.7. Equipos usados en este experimento y su función.	103
5.8. Simulado (a) y Real (b)	104
5.9. Presentación en Global Robot Expo	104
5.10. Imagen del experimento	106
5.11. Imagen del experimento	109
5.12. Montaje de todos los equipos en la misma red local	111
5.13. Montaje del servidor en casa y los clientes en otras redes	112
5.14. Montaje del servidor en la universidad y los clientes en otras redes	113

Índice de cuadros

3.1. Soporte de Canvas	36
3.2. Soporte para WebWorker	37
3.3. Soporte para WebSocket	37
3.4. Soporte para Promise	39
3.5. Comparación JQuery, JavaScript	42
5.1. Resultados de las pruebas	110

Capítulo 1

Introducción

Este Trabajo fin de grado se encuentra en la intersección entre dos campos, la robótica y las tecnologías web. En este capítulo vamos a introducir los conceptos de *middleware* robótico y aplicación web. A modo de ejemplo de presentan varias aplicaciones web que usan sendos *middlewares* robóticos. En particular las dos empleadas en el laboratorio de robótica de la URJC, que son el contexto inmediato de este TFG.

1.1. Robótica

La palabra robot proviene del término checo robota, cuyo significado es “servidumbre, trabajo forzado o esclavitud”. Principalmente usado para referirse a los siervos checoslovacos de la época feudal, con el paso de las épocas ha ido evolucionando hasta la actualidad que el término es usado por la juventud checa y eslovaca para referirse al trabajo aburrido o poco interesante. Según la Real Academia Española, el término robot se refiere a:

“Máquina o ingenio electrónico programable, capaz de manipular objetos y realizar operaciones antes reservadas sólo a las personas”

La robótica como hoy la conocemos, surge hacia la década de los 50, con la construcción de las primeras computadoras. Aunque no será hasta la década de los 70 con los primeros microprocesadores cuando comience su vertiginoso avance. En 1954 George Devol diseñó el primer mecanismo programable aplicado a la industria, el “Universal Automation o Unimation”, que sería el corazón del primer brazo robótico industrial. Los brazos robóticos son usados para tareas repetitivas pudiendo mantener siempre la misma precisión. Desde

entonces hasta ahora han avanzado mucho y ya no son sólo brazos robóticos, uno de los robots industriales más avanzados es Baxter (figura 1.1)

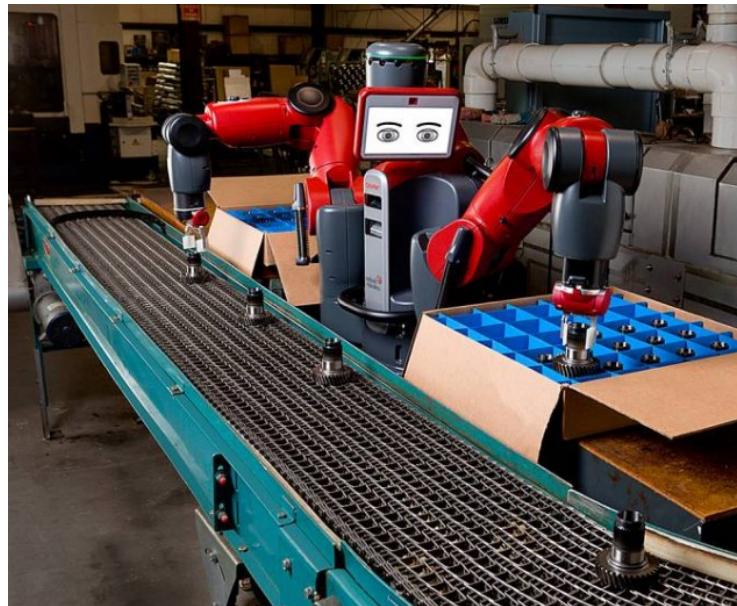


Figura 1.1: Baxter en una cadena de embalaje

Fuera de las fábricas se empezaron a usar en terrenos hostiles para el hombre, como las zonas radioactivas o en la exploración espacial. Además, actualmente se empieza a usar en otros entornos como la agricultura, el ocio y el entretenimiento. Por ejemplo, con robots como Aibo de Sony (figura 1.2(b)) y en los hogares Roomba (figura 1.2(a)) para la limpieza de la casa, en los coches, con las funciones como aparcar o incluso de conducir solos (figura 1.3(a)). También se usan en los almacenes grandes como los de Amazon para transportar los productos de un lado a otro (figura 1.3(b))



Figura 1.2: Roomba de iRobot (a) y Aibo de Sony (b)



Figura 1.3: GoogleCar (a) y Robots de Amazon (b)

Actualmente hay robots que entienden emociones e interactúan con personas para hacerlas sentir mejor, como es el caso de Pepper¹ (figura 1.4(a)) o incluso son capaces de realizar labores de recepcionista en un hotel en Japón² (figura 1.4(b)).



Figura 1.4: Pepper (a) y Hotel robótico (b)

Sin un *software* todo esto sería impensable. *Hardware* y *software* deben trabajar juntos, no pueden existir el uno sin el otro. Un robot no puede funcionar sin un *software* que le permita acceder a los datos de los sensores y actuadores, que le dote de inteligencia, le permita llevar a cabo algoritmos perceptivos y de toma de decisiones, en definitiva, sin un *software*, un robot no difiere mucho de un pisapapeles muy caro.

El *Middleware* es un *software* que asiste a una aplicación para interactuar o comunicarse con otras aplicaciones, o paquetes de programas, redes, hardware y/o sistemas operativos. Éste simplifica el trabajo de los programadores en la compleja tarea de generar las conexiones y sincronizaciones que son necesarias en los sistemas distribuidos. Abstactra de la complejidad y heterogeneidad de las redes de comunicaciones subyacentes, así como de los sistemas operativos y lenguajes de programación, proporcionando una API para la fácil programación y manejo de aplicaciones distribuidas.

En los últimos años se han asentado varios *middlewares* en el campo de la robótica, que simplifican la creación de aplicaciones en ese ámbito.

¹fuente: https://www.aldebaran.com/en/Launch_Sales_of_Pepper

²fuente: <http://cnnespanol.cnn.com/2015/07/18/inauguran-en-japon-el-primer-hotel-atendido-por-robots/>

1.1.1. ROS

ROS³ es un *middleware* para el desarrollo de *software* para robots, bajo licencia de código abierto y mantenido por OSRF⁴ (*Open Source Robotics Foundation*). Provee servicios típicos de un sistema operativo, como son abstracción de acceso al hardware, control de bajo nivel para dispositivos, mecanismos de paso de mensajes entre procesos y nodos (*Topics Services Actions*) y mantenimiento de paquetes. Se propone como una capa mediadora entre el robot y sistema operativo por un lado y el programador por otro. Es un *software* multi-plataforma aunque actualmente sólo la versión para Linux es considerada estable. Por el momento, los lenguajes de desarrollo soportados son C++, Python y LISP, aunque hay intención de soportar algunos más proporcionando librerías cliente que puedan acceder al API de ROS, como es el caso de `roslibjs` que da soporte para JavaScript

Su diseño va en la línea de ser lo más ligero posible y capaz de ser integrado o utilizado fácilmente por otros sistemas existentes, para fomentar la reutilización de *software* robótico. Sus librerías se han diseñado para ofrecer interfaces claros y limpios.

Otro punto a destacar es su sistema de gestión del *software*, que provee mecanismos para la distribución e integración de paquetes de *software* con funcionalidad determinada. Incluye la estructura de directorios que un paquete debe tener y cómo deben usarse, descripciones de la funcionalidad contenida, y detalles de alto nivel de cómo se comunica dicho *software* con otras piezas.

Se ha extendido tanto, como demuestran sus aproximadamente 9 millones de paquetes descargados de unas 70.000 IPs diferentes sólo en Mayo de 2015, que ha llegado a convertirse en estándar.

1.1.2. JdeRobot

JdeRobot [1] es un *middleware* para el desarrollo de aplicaciones orientadas a la robótica, domótica y visión artificial (ejemplos en la figura 1.5). La versión actual de JdeRobot es la 5.3. Esta plataforma está diseñada para facilitar la programación de *software* que dote de cierta inteligencia a hardwares tan distintos como cámaras, actuadores y en general robots de todo tipo. Los componentes están escritos en C++, Python, Java... y basadas en un entorno de componentes distribuidos. Cada aplicación está construida con la concurrencia de varios componentes asíncronos. Cada componente puede ejecutarse en una máquina distinta y para su interconexión se usa el *middleware* de comunicaciones ICE [19].

³Web: <http://www.ros.org/>

⁴<http://www.osrfoundation.org/>

JdeRobot simplifica el acceso a los dispositivos hardware desde el programa principal. Así, obtener una medida de un sensor u ordenar un movimiento a un motor es tan simple como llamar a una función local.

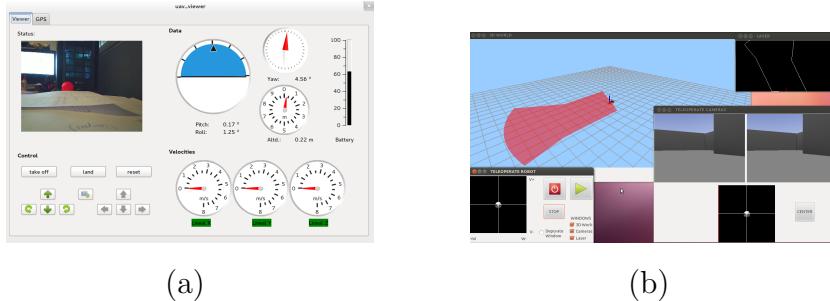


Figura 1.5: Uav Viewer (a) e Introrob (b)

Los componentes desarrollados por JdeRobot pueden conectarse a sensores y actuadores reales o simulados. Estas conexiones pueden ser locales (dentro de la misma máquina), a nivel de red local o usando Internet.

Actualmente se usa tanto en docencia como en investigación en la URJC y ha formado parte del *Google Summer of Code 2015*⁵, programa donde Google remunera a los estudiantes mayores de 18 años que completan un proyecto de programación de *software libre* durante ese periodo de verano.

ICE

ZeroC ICE [19][20] (*Internet Communications Engine*) es un *middleware* de comunicaciones, *software libre*, orientado al desarrollo de aplicaciones distribuidas que permite a los programadores centrarse en la lógica de la aplicación, haciendo transparentes detalles como abrir conexiones, retransmitir paquetes por la red, serialización, etc. Es compatible con lenguajes como C++, Java, Python, PHP o C#, haciendo posible que dos máquinas con procesos escritos en lenguajes distintos puedan comunicarse.

ICE ofrece mecanismos de RPC (llamadas a procedimientos remotos), tanto asíncronas como síncronas, control de hilos sin necesidad de preocuparnos por regiones críticas en cuanto a accesos a memoria, posibilidad de elegir entre TCP, UDP o SSL como protocolos de nivel de transporte, y un lenguaje propio llamado *slice* para establecer interfaces de comunicación. Con *slice* se pueden definir clases, métodos, tipos definidos por el usuario como diccionarios, secuencias o enumeraciones, herencias, excepciones, etc. Tras definir

⁵<https://www.google-melange.com/gsoc/org2/google/gsoc2015/jderobot>

una interfaz, es necesario traducirlo al lenguaje concreto de nuestra aplicación mediante unos compiladores llamados `ice2` “lenguaje” [21] ya incluidos en la instalación, por ejemplo, `ice2cpp`, `ice2python`.

La versión utilizada en este proyecto es la 3.5.1.

1.2. Tecnologías Web

La principal manera de acceder a Internet es mediante el navegador para ver páginas web y para ello se utiliza el protocolo HTTP. Las aplicaciones web tienen un lado cliente y un lado servidor. Los lenguajes más usados en el lado del cliente son HTML, para expresar el contenido de las páginas, JavaScript para interactuar con ellas y CSS para modificar su apariencia. En este punto vamos a tratar cada una de estas tecnologías. Una de las principales ventajas es que son multiplataforma y de la mano de los *smartphones* se han convertido en una verdadera revolución digital. Actualmente *World Wide Web Consortium* (W3C) está elaborando una API para permitir a las aplicaciones del navegador realizar llamadas de voz, chat de vídeo y uso compartido de archivos P2P sin plugins, llamada WebRTC⁶.

Internet es un conjunto descentralizado de redes de comunicación interconectadas que utilizan la familia de protocolos TCP/IP, lo cual garantiza que las redes físicas heterogéneas que la componen funcionen como una red lógica única de alcance mundial. Nació a partir de una red denominada ARPANET, diseñada y desarrollada en 1969 para el Departamento de Defensa de Estados Unidos, creada para mantener la comunicación entre computadoras en caso de guerra.

Estados Unidos fue capaz de desarrollar una red que funcionara (la antecesora de la actual Internet) y los usuarios académicos e investigadores que tenían acceso a ella empezaron a usarla constantemente. Los desarrolladores de Internet en Estados Unidos, el Reino Unido y Escandinavia, en respuesta a las presiones del mercado, empezaron a poner el *software* de IP (*Internet Protocol*) en todo tipo de computadoras, llegando a ser un estándar. Al mismo tiempo que Internet se consolidaba, muchas compañías y otras organizaciones empezaron a construir redes privadas usando los mismos protocolos de ARPAnet, por lo que los usuarios de una red podrían comunicarse con usuarios de otra.

Actualmente se usan para casi todo, desde buscar direcciones en un mapa (figura 1.6(a)), enterarse de las noticias viendo el periódico (figura 1.7(a)), relacionarse con otras personas

⁶<https://webrtc.org/>

(figura 1.6(b)) o incluso han cambiado la forma en la que vemos la televisión (figura 1.7(b)).

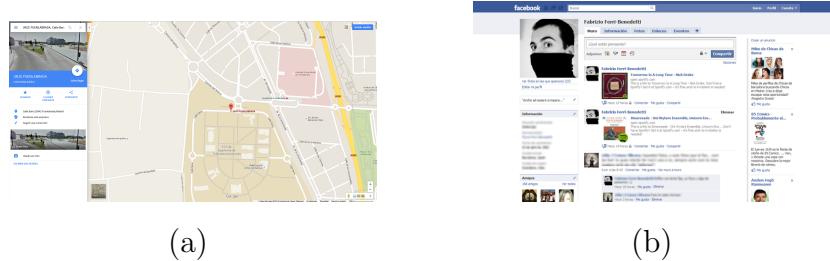


Figura 1.6: Google Maps (a) y Facebook (b)



Figura 1.7: El País (a) y Netflix (b)

1.2.1. HTTP

Este protocolo fue desarrollado por el *World Wide Web Consortium*⁷ y la *Internet Engineering Task Force*. En 1999 se publicó la versión 1.1 usada en la actualidad. HTTP define la sintaxis y la semántica que utilizan los elementos de *software* de la arquitectura web (clientes, servidores, proxies) para comunicarse. Es un protocolo orientado a transacciones y sigue el esquema petición-respuesta entre un cliente y un servidor (figura 1.8). Al cliente que efectúa la petición (un navegador web) se lo conoce como *user agent* (agente del usuario). A la información transmitida se le llama recurso y se identifica mediante un localizador uniforme de recursos (URL). El resultado de la ejecución de un programa, una consulta a una base de datos, la traducción automática de un documento, etc.

Es un protocolo sin estado, es decir, que no guarda ninguna información sobre conexiones anteriores.

⁷Web: <http://www.w3.org/>



Figura 1.8: Comunicación cliente servidor en HTTP.

1.2.2. HTML

HTML significa Lenguaje de Marcado para Hipertextos (*HyperText Markup Language*), y es el bloque de construcción más básico de una página web y se usa para crear y representar visualmente una página web. Determina el contenido de la página web, pero no su funcionalidad. Consta de etiquetas para delimitar los bloques (figura 1.9)

Es un estándar a cargo de la W3C⁸, organización dedicada a la estandarización de casi todas las tecnologías ligadas a la web, sobre todo en lo referente a su escritura e interpretación. Se considera el lenguaje web más importante siendo su invención crucial en la aparición, desarrollo y expansión de la *World Wide Web*. Es el estándar que se ha impuesto en la visualización de páginas web y es el que todos los navegadores actuales han adoptado.

A lo largo de sus diferentes versiones, se han incorporado y suprimido diversas características, con el fin de hacerlo más eficiente y facilitar el desarrollo de páginas web compatibles con distintos navegadores y plataformas (PC de escritorio, portátiles, teléfonos inteligentes, tabletas, etc.). No obstante, para interpretar correctamente una nueva versión de HTML, los desarrolladores de navegadores web deben incorporar estos cambios. Así mismo, las páginas escritas en una versión anterior de HTML deberían ser actualizadas o reescritas, lo que no siempre se cumple. Es por ello que ciertos navegadores aún mantienen la capacidad de interpretar páginas web de versiones HTML anteriores. Por estas razones, aún existen diferencias entre distintos navegadores y versiones al interpretar una misma página web.

⁸Web: <http://www.w3.org/>

```

1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>Example</title>
5     <link rel="stylesheet" href="styi
6   </head>
7   <body>
8     <h1>
9       <a href="/">Header</a>
10    </h1>
11    <nav>
12      <a href="one/">One</a>
13      <a href="two/">Two</a>
14      <a href="three/">Three</a>
15    </nav>

```

Figura 1.9: Ejemplo de código HTML.

El origen de HTML se remonta a 1980, cuando el físico Tim Berners-Lee, trabajador del CERN (Organización Europea para la Investigación Nuclear) propuso un nuevo sistema de "hipertexto" para compartir documentos. Tras finalizar el desarrollo, Tim Berners-Lee lo presentó a una convocatoria organizada para desarrollar un sistema de "hipertexto" para Internet. Después de unir sus fuerzas con el ingeniero de sistemas Robert Cailliau, presentaron la propuesta ganadora llamada WorldWideWeb (W3). El primer documento formal con la descripción de HTML se publicó en 1991 bajo el nombre HTML Tags (Etiquetas HTML).

La primera propuesta oficial para convertir HTML en un estándar se realizó en 1993 por parte del organismo IETF (*Internet Engineering Task Force*). Aunque no consiguieron convertirse en estándar oficial.

En 1995, el organismo IETF organiza un grupo de trabajo de HTML y consigue publicar, el 22 de septiembre de ese mismo año, el estándar HTML 2.0. A pesar de su nombre, HTML 2.0 es el primer estándar oficial de HTML.

A partir de 1996, los estándares de HTML los publica otro organismo de estandarización llamado W3C (*World Wide Web Consortium*⁹). La versión HTML 3.2 se publicó el 14 de Enero de 1997. Esta revisión incorpora *applets* de Java y texto que fluye alrededor de las imágenes.

HTML 4.0 se publicó el 24 de abril de 1998 y supone un gran salto desde las versiones anteriores. Entre sus novedades más destacadas se encuentran las hojas de estilos CSS, la posibilidad de incluir pequeños programas o *scripts* en las páginas web, mejora de la

⁹Web: <http://www.w3.org/>

accesibilidad de las páginas diseñadas, tablas complejas y mejoras en los formularios.

Desde la publicación de HTML 4.01, la actividad de estandarización de HTML se detuvo y el W3C se centró en el desarrollo del estándar XHTML. Por este motivo, en el año 2004, las empresas Apple, Mozilla y Opera mostraron su preocupación por la falta de interés del W3C en HTML y decidieron organizarse en una nueva asociación llamada WHATWG (*Web Hypertext Application Technology Working Group*).

Debido a la fuerza de las empresas que forman el grupo WHATWG y a la publicación de los borradores de HTML 5.0, en marzo de 2007 el W3C¹⁰ decidió retomar la actividad estandarizadora de HTML. La versión definitiva de la quinta revisión del estándar se publicó en octubre de 2014¹¹. Esta revisión incluye nuevas características como pueden ser la geolocalización (hasta la llegada de los *smartphones* no era necesaria) y soporte para vídeos y audios sin necesidad de plugins externos.

1.2.3. JavaScript

JavaScript (abreviado comúnmente “JS”) es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico. Se utiliza principalmente en su forma del lado del cliente (*client-side*), que es lo que usamos en este TFG y que describiremos en detalle en el capítulo 3, se ejecuta dentro del navegador web, que contiene el intérprete de JavaScript, permitiendo mejoras en la interfaz de usuario y páginas web dinámicas.

La idea de Javascript surgió cuando a principios de los 90 con unas aplicaciones web cada vez más complejas y una velocidad de navegación tan lenta, surgió la necesidad de un lenguaje de programación que se ejecutara en el navegador del usuario y Brendan Eich, un programador que trabajaba en Netscape, creó LiveScript.

Posteriormente, Netscape firmó una alianza con Sun Microsystems para el desarrollo del nuevo lenguaje de programación. Además, justo antes del lanzamiento Netscape decidió cambiar el nombre por el de JavaScript. La razón del cambio de nombre fue exclusivamente por marketing, ya que Java era la palabra de moda en el mundo informático y de Internet de la época.

La primera versión de JavaScript fue un completo éxito. Al mismo tiempo, Microsoft lanzó JScript con su navegador Internet Explorer 3, el cual era una copia de JavaScript

¹⁰Web: <http://www.w3.org/>

¹¹<http://www.w3.org/TR/2014/REC-html5-20141028/>.

al que le cambiaron el nombre para evitar problemas legales. Para evitar una guerra de tecnologías, Netscape decidió que lo mejor sería estandarizar el lenguaje JavaScript. De esta forma, en 1997 se envió la especificación JavaScript 1.1 al organismo ECMA¹² (*European Computer Manufacturers Association*).

ECMA creó el comité TC39 con el objetivo de “estandarizar de un lenguaje de *script* multiplataforma e independiente de cualquier empresa”. El primer estándar que creó el comité TC39 se denominó ECMA-262, en el que se definió por primera vez el lenguaje ECMAScript.

Al principio muchos desarrolladores renegaban del lenguaje porque el público al que va dirigido lo formaban publicadores de artículos y demás aficionados, entre otras razones. La llegada de AJAX (permite interacciones ligeras entre navegador y servidor web) devolvió JavaScript a la fama y atrajo la atención de muchos otros programadores. Como resultado de esto hubo una proliferación de un conjunto de entornos y librerías de ámbito general, mejorando las prácticas de programación con JavaScript, y aumentando el uso de JavaScript fuera de los navegadores web, como se ha visto con la proliferación de entornos JavaScript del lado del servidor. En enero de 2009, el proyecto CommonJS fue inaugurado con el objetivo de especificar una librería para uso de tareas comunes principalmente para el desarrollo fuera del navegador web. En mayo de 2009 surge Node.js¹³ que permite ejecutar aplicaciones JavaScript en el lado del servidor.

En Junio de 2015 se cerró y publicó el estándar ECMAScript¹⁴ 619 20 (última versión hasta la fecha) con un soporte irregular entre navegadores y que dota a JavaScript de características avanzadas que se echaban de menos y que son de uso habitual en otros lenguajes como, por ejemplo, módulos para organización del código, verdaderas clases para POO, expresiones de flecha, iteradores, generadores o promesas para programación asíncrona.

Más allá del lenguaje hay bloques de funcionalidad ya resuelta, bibliotecas en JavaScript que se verán con mas detalle en el capítulo 3.

JQuery[27] es una biblioteca de JavaScript, creada inicialmente por John Resig, que permite simplificar el uso de JavaScript. La primera versión de jQuery se presentó en Febrero de 2006. Esta librería sorprendió gratamente a la comunidad de desarrolladores web puesto que simplificaba en gran medida la programación de código JavaScript para interactuar, tanto con los elementos del DOM como para gestionar los diferentes eventos

¹²<http://www.ecma-international.org/>

¹³<https://nodejs.org/en/>

¹⁴<http://www.ecma-international.org/publications/standards/Ecma-262.htm>

que se producen en la visita de una página web. Desde su publicación, la librería ha evolucionado solucionando errores, optimizando su código y ampliando las funcionalidades ofrecidas. Actualmente está la versión 2.14 y cuenta con el apoyo de Google, Microsoft, IBM,...

WebGL[14] es una especificación estándar que está siendo desarrollada actualmente para mostrar gráficos en 3D en navegadores web. Técnicamente es un API para JavaScript que permite usar la implementación nativa de OpenGL ES 2.0 que será incorporada en los navegadores. WebGL es gestionado por el consorcio de tecnología sin ánimo de lucro Khronos Group¹⁵. WebGL creció desde los experimentos del canvas 3D comenzados Mozilla. El primero mostró un prototipo de Canvas 3D en 2006. A finales de 2007, tanto Mozilla como Opera habían hecho sus propias implementaciones separadas. Notables primeras aplicaciones de WebGL son Google Maps y Zygote Body.

1.3. Tecnologías web en middlewares robóticos

Este TFG se sitúa a caballo entre los dos campos mencionados, robótica y tecnologías web. Algunos trabajos previos existentes en esta intersección son los siguientes.

1.3.1. The Robot Management System

El RMS¹⁶ (Robot Management System) es una herramienta que permite controlar desde una página web robots que tengan el *middleware* ROS. RMS en sí se refiere a un sistema de gestión web escrito en PHP que está respaldado por una base de datos MySQL. El sistema utiliza un framework modelo-vista-controlador (MVC) a través del popular framework CakePHP¹⁷. RMS está escrito en forma independiente de la plataforma robot (el robot en sí) por lo que permite el control de una gran variedad de robots. Además de ser multiplataforma, RMS permite usuario básico y gestión de accesos, gestión de la interfaz, gestión de contenidos, y los estudios de los usuarios remotos. RMS se desarrolla como parte del esfuerzo por desarrollar herramienta de la “Robot Web Tools”¹⁸. En la Figura 1.10 hay un ejemplo de cómo se ve dicha aplicación.

¹⁵<https://www.khronos.org/>

¹⁶<http://wiki.ros.org/rms/>

¹⁷<http://cakephp.org/>

¹⁸<http://robotwebtools.org/>

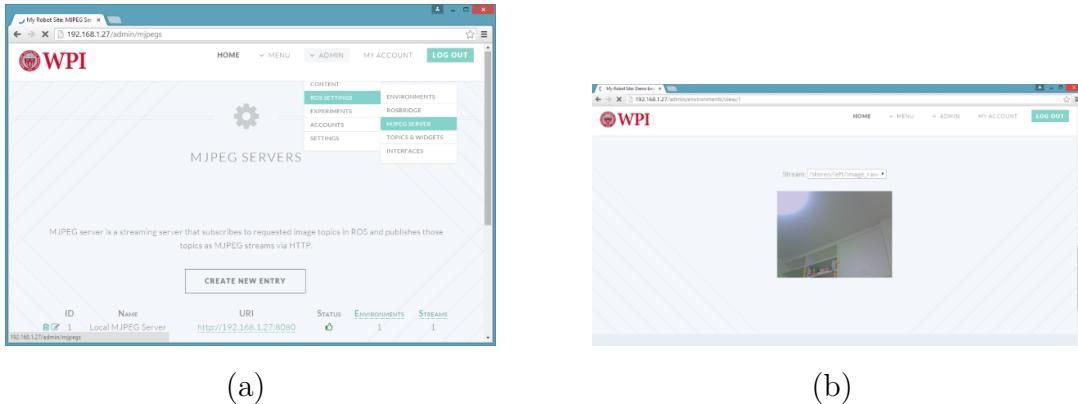


Figura 1.10: Menú de RMS (a) y Cámara a través de RMS (b)

1.3.2. Surveillance 4.0 (URJC)

Surveillance 4.0 [15] desarrollado por Daniel Castellano como su Proyecto Fin de Carrera. Esta aplicación contaba con varios sensores de distinto tipo (humedad, temperatura, gas, etc) que se conectaban inalámbricamente con un nodo central situado en una Raspberry Pi. La conexión inalámbrica se hacía mediante transmisores Zigbee¹⁹ con un protocolo propio llamado WHAP. El nodo central recibía los datos de los sensores y los mostraba mediante un servidor web que corría en la misma máquina. La aplicación web se desarrolló en Python usando el entorno de desarrollo web Django. En Surveillance 4.0, los valores de los sensores se guardaban en una base de datos que la aplicación web consultaba cuando era necesario. Además, esta versión incluía un *streaming* de vídeo utilizando el *software* de código abierto M-JPEG Streamer. En la figura 1.11 se puede ver la aplicación.

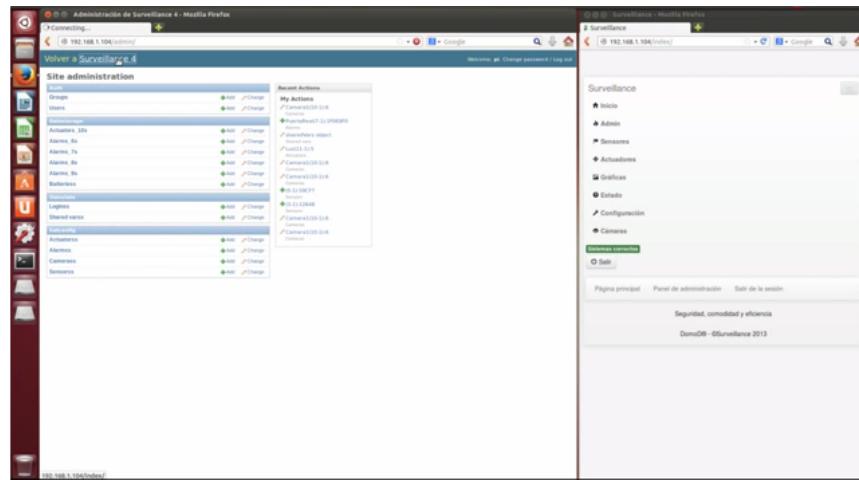
1.3.3. Surveillance 5.1 (URJC)

Surveillance 5.1 [16] desarrollado por Edgar Barrero como su Trabajo Fin de Grado. Esta aplicación obtenía un flujo de imágenes de una cámara web, un flujo de imágenes de profundidad de un sensor Kinect, además de datos de un sensor de humedad y de interaccionar con un actuador. La aplicación web se desarrolló en Ruby sobre Rails. En Surveillance 5.1, el servidor web se conectaba a los componentes de JdeRobot mediante sus interfaces ICE. La aplicación web refrescaba estos datos mediante peticiones AJAX. En la figura 1.12 se puede ver la aplicación.

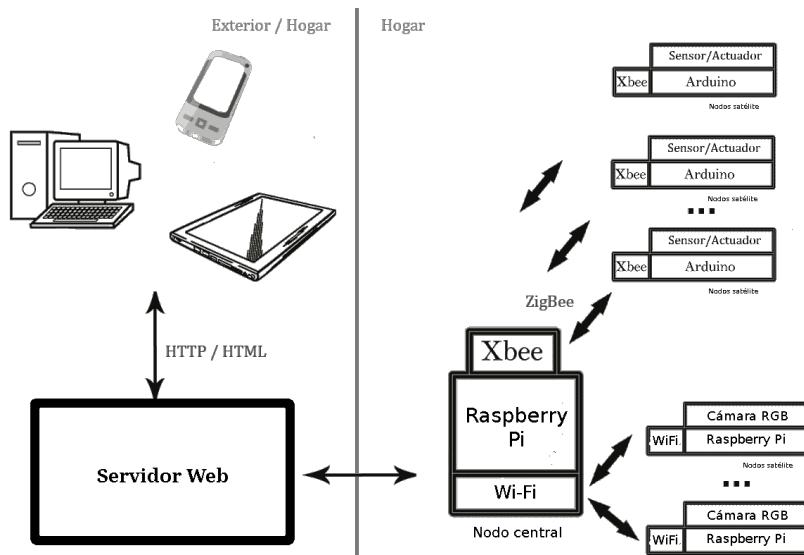
¹⁹<http://www.zigbee.org/>

En el Trabajo de Fin de Grado presentado en esta memoria se desarrollan seis clientes web: CameraViewJS, RGBDViewerJS, KobukiViewerJS, UavViewerJS, IntrorobKobukiJS e IntrorobUavJS, que son las versiones web de las herramientas de JdeRobot homónimas y que hablan directamente con los servidores que tiene JdeRobot para acceder a los sensores y robots a diferencia de los dos últimos antecedentes, que usan un servidor web como intermediario.

En el siguiente capítulo se presentan los objetivos de este TFG. En el capítulo de Plataforma de Desarrollo se hace un resumen de las tecnologías usadas en los clientes. En el capítulo de Diseño e implementación se profundiza en el diseño y programación de los clientes. Las pruebas se detallan en el capítulo de Experimentos. Por último, el capítulo de Conclusiones se hace un resumen de los objetivos logrados y presenta futuras líneas de trabajo.

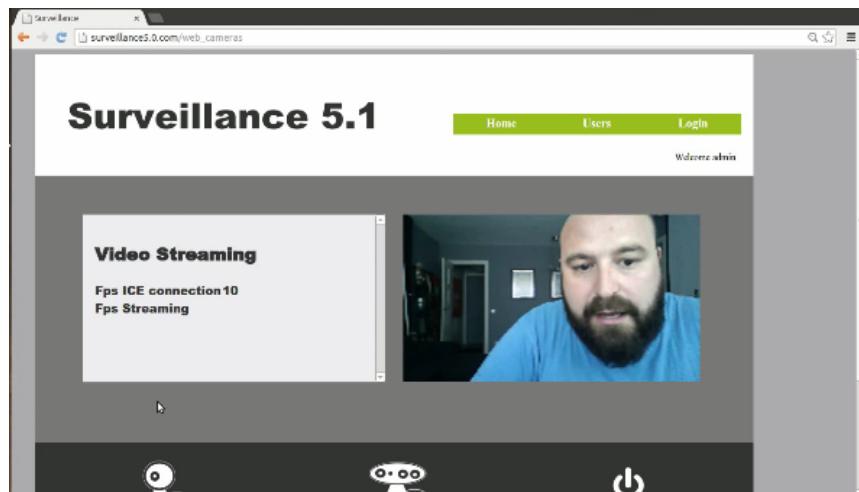


(a)

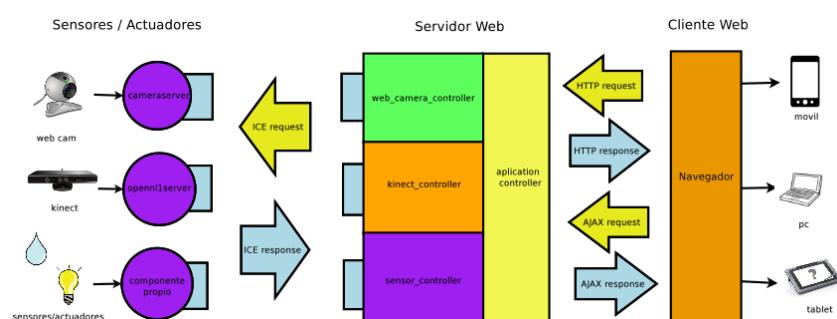


(b)

Figura 1.11: interfaz (a) y arquitectura (b)



(a)



(b)

Figura 1.12: interfaz (a) y arquitectura (b)

Capítulo 2

Objetivos y Metodología

Una vez presentado el contexto general sobre el que se asienta este trabajo, vamos a describir los objetivos concretos que pretendemos resolver con la realización de este TFG y los requisitos que han condicionado la solución desarrollada.

2.1. Descripción del problema

El objetivo general del TFG consiste en crear versiones web de seis herramientas de JdeRobot muy utilizadas y que actualmente están programadas en C++ o Python con su propio interfaz gráfico, que usa bibliotecas como QT o GTK y que sólo se puede ejecutar en Linux. Éstas versiones web van a ser multiplataforma (Linux, Android, IOS, Windows,...), van a usar el navegador web como interfaz gráfico y nos va a permitir acceder a los sensores y actuadores sin un servidor web intermedio.

Este objetivo final lo hemos dividido en cinco sub-objetivos:

1. *CameraViewJS*: Creación del cliente web similar a la herramienta CameraView para visualizar imágenes procedentes del servidor Camerastreamer.
2. *RGBDViewerJS*: Creación del cliente web similar a la herramienta RGBDViewer para visualizar datos de color y profundidad procedentes del servidor Openni1Server.
3. *KobukiViewerJS*: Creación de un teleoperador para poder ver manejar y ver los datos de los sensores de los robots Kobuki y Pioneer del laboratorio de robótica de la URJC.
Versión web de [KobukiViewer](#)

4. *UavViewerJS*: Creación del cliente web similar a la herramienta UavViewer para teleoperar drones tanto reales como simulados y ver los datos de sus sensores.
5. Creación dos herramientas que además de mostrar los datos sensoriales del robot y ofrecer su teleoperación, permite insertar código que gobierna el comportamiento autónomo de robots Kobuki y drones (IntrorobKobukiJS e IntrorobUavJS).

Además, se ha creado una página web para contener a todos los clientes.

2.2. Requisitos

Se deben satisfacer los siguientes requisitos:

- Se tiene que usar la última versión de JdeRobot, la 5.3.1.
- La comunicación con los servidores de sensores y actuadores debe ser en tiempo real.
- No habrá servidores web intermedios, a diferencia de los antecedentes descritos en las secciones 1.3.2 y 1.3.3.
- Debe ser lo suficientemente maduro para poder integrarse en el repositorio oficial de JdeRobot y ser usado por terceros fácilmente.

2.3. Metodología y plan de trabajo

Para el desarrollo de este TFG se ha seguido el método de desarrollo en espiral. Este sistema se basa en iteraciones sucesivas en la que cada bucle o iteración representa un conjunto de actividades. Estas actividades incluyen tal y como muestra la figura 2.1: análisis de los requerimientos del sistema, diseño del mismo, implementación y pruebas.

Éste TFG se ha ido documentando en un Mediawiki [17]. Este cuaderno de bitácora ha servido de apoyo a las reuniones con el tutor. Como complemento a este Mediawiki se ha utilizado un repositorio público svn [18] donde se ha guardado y está accesible todo el código fuente de este TFG.

El plan de trabajo, repartido en diferentes fases, con el fin de utilizar en cada fase nueva desarrollos ya creados en las anteriores, ha sido el siguiente:

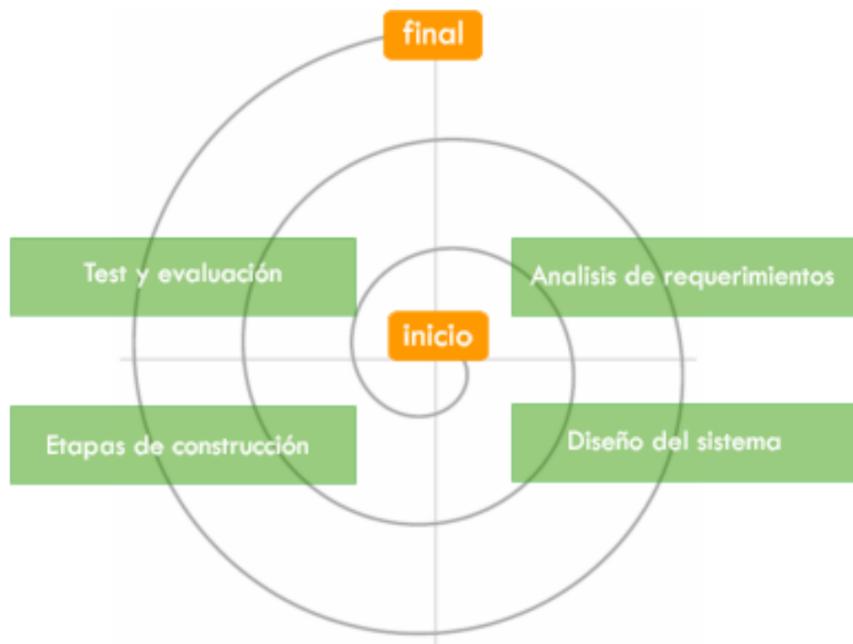


Figura 2.1: Modelo en espiral

- **Aprendizaje de JdeRobot:** Primer contacto con la plataforma con el objetivo de saber cómo funciona.
- **Familiarización con las tecnologías web a utilizar:** Tiene el objetivo de aprender lo necesario sobre HTML, JavaScript, CSS, WebGL, ThreeJS. Primeros contactos con ICE-JS, *middleware* utilizado para la comunicación entre los clientes que se van a crear y los servidores de JdeRobot.
- **Desarrollo de CameraViewJS:** En esta fase se crea el módulo que recibe el flujo de vídeo, que es la base del cliente.
- **Desarrollo de RGBDViewerJS:** Se crea un módulo para recibir el flujo de imágenes de distancia, juntando éste flujo con otro de vídeo (ambos procedentes de un Kinect) mediante lo aprendido de ThreeJS se crea una representación en 3D de los datos recibidos por el sensor.
- **KobukiViewerJS:** Se crea un módulo que permite interactuar con los motores de dicho robot mediante un control con ThreeJS. Se incluyen dos flujos de vídeo que representan cada una de las cámaras y se añade a la web. Además se crean los módulos para recibir la información láser y odometría y una representación 3D del robot moviéndose según los datos recibidos de la odometría y se muestran los datos

del láser.

- **UavViewerJS:** Se crea un módulo que mediante dos controles iguales a los del teleoperador anterior permite mover el UAV. Además se utiliza una representación de indicadores de un avión para los datos de posición recibidos, al igual que se crea una representación 3D.
- **Introrob:** A *KobukiViewerJS* y a *UavViewerJS* se les añade la opción de poder agregar comportamiento autónomo en vez de teleoperarlos.
- **Diseño final de la web:** Se adapta el diseño de la web para que sea vistoso y fácil de manejar e integre todos los desarrollos anteriores.

Además se han hecho reuniones periódicas con el tutor para tener un seguimiento adecuado del desarrollo del mismo, que están documentadas en la bitácora¹.

¹<http://jderobot.org/Aitormf-tfg>

Capítulo 3

Plataforma de Desarrollo

Una vez que hemos detallado los requisitos y objetivos de este proyecto, vamos a describir en este capítulo la infraestructura empleada y en la que nos hemos apoyado.

3.1. Simulador Gazebo

Gazebo[9] es un simulador de software libre que incluye multitud de modelos y motores de física virtualizada y mantenido por OSRF¹ (*Open Source Robotics Foundation*). Ofrece una interfaz gráfica y control sobre los objetos y el mundo (bastante realista) generado, además de la creación y modificación de actuadores y sensores personalizados. Permite probar algoritmos robóticos en mundos virtuales para madurarlos antes de llevarlos a robots reales, también permite impartir docencia en robótica sin tener los robots físicamente. Por ejemplo, se pueden crear vehículos con diferentes sensores o casas con las que interactuar. En la figura 3.1 se puede ver la interfaz gráfica de Gazebo. Ha sido elegido por DARPA para su DRC².

Nosotros lo hemos utilizado tanto para probar los teleoperadores (UavViewerJS, KobukiViewerJS) como los introrob (IntrorobUavJS, IntrorobKobuki) con robots y drones simulados hasta que han sido lo suficientemente maduros.

¹<http://www.osrfoundation.org/>

²<http://www.theroboticschallenge.org/>

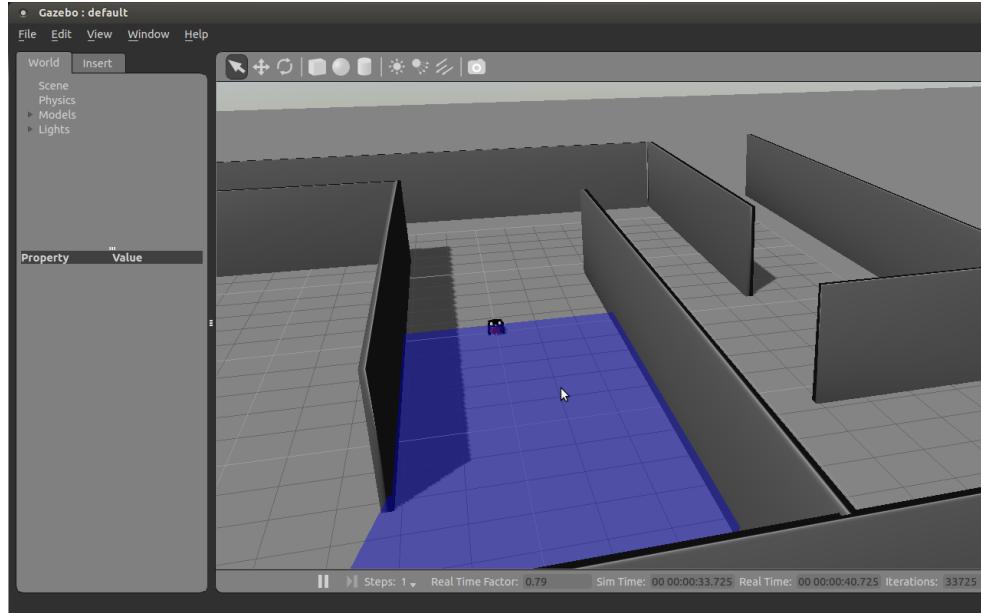


Figura 3.1: Interfaz gráfica del simulador Gazebo.

3.2. JdeRobot

A continuación se detallan los componentes de *JdeRobot*, plataforma ya introducida en el capítulo 1, utilizados en este TFG. Una de las ventajas que tiene es que el interfaz ICE del driver real y simulado es el mismo por lo que con cambiar la configuración se puede usar uno u otro sin tener que volver a compilar las aplicaciones.

En este TFG se ha utilizado la versión 5.3.1.

3.2.1. CameraServer + CameraView

Cameraserver[3] es un componente incluido en *JdeRobot* que ofrece *streaming* de vídeo de una o varias cámaras, tanto reales como simuladas. Usa GStreamer[5] internamente para manejar y procesar las fuentes de vídeo. Está escrito en C++ y ofrece una interfaz ICE con la que comunicarse. La figura 3.2 ofrece un esquema del funcionamiento de **cameraserver** junto a otro componente de *JdeRobot* que muestra el *streaming* de vídeo.

Cualquier programa que quiera conectarse con el componente **cameraserver** debe importar la interfaz **camera.ice**. Esta interfaz a su vez importa otra llamada **image.ice**. Esta última contiene un método llamado “**getImageData(String ImgFormat)**”. Al invocar este método se devuelve una estructura formada por una imagen en el formato indicado con **ImgFormat** y las características de la misma.

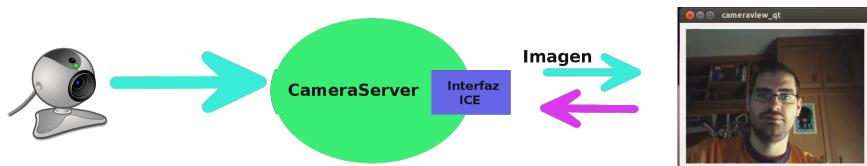


Figura 3.2: Ejemplo del uso del componente CameraServer.

Para la ejecución de este componente, es necesario proporcionar la ruta de su archivo de configuración. En este fichero de configuración se define la IP y puerto donde escucha el servidor, el número y nombre de las cámaras que tiene y la configuración de las mismas.

3.2.2. Openni1Server + RgbdViewer

`Openni1Server`[4] es otro componente incluido en JdeRobot. Está escrito en C++ y para su funcionamiento usa las librerías Opencv³ y Openni⁴ entre otras. Este componente tiene todos los drivers necesarios para acceder al sensor Kinect y ofrece tres tipos de datos: una imagen RGB, una imagen de profundidad y una nube de puntos. Al ofrecer tres tipos distintos de datos tiene tres interfaces ICE. En este TFG sólo se va a hacer uso de las interfaces de imágenes.

Para usar tanto la interfaz de imagen RGB como la de profundidad hacen falta los mismos interfaces ICE que en `CameraServer` ya que al tratarse de *streamings* de imágenes se ha unificado la interfaz. De nuevo la interfaz contiene un método llamado “`getImageData(String ImgFormat)`”. Al invocar este método se devuelve una estructura formada por una imagen en el formato indicado con `ImgFormat` y las características de la misma, en el caso de la imagen RGB `ImgFormat` es “`RGB8`” mientras que en la imagen de distancia es “`DEPTH8_16`”. En la figura 3.3 se refleja éste funcionamiento.

3.2.3. Ardrone_server + uav_viewer

`Ardrone_server`[7] es otro componente incluido en JdeRobot, escrito en C++. Este componente permite comunicarse con el drone ArDrone 2 + GPS de Parrot, ofreciendo varias interfaces tanto para recibir información como para enviar órdenes (Imagen, Pose3D, CmdVel, ...).

³<http://opencv.org/>

⁴<http://www.openni.tk/openni/>

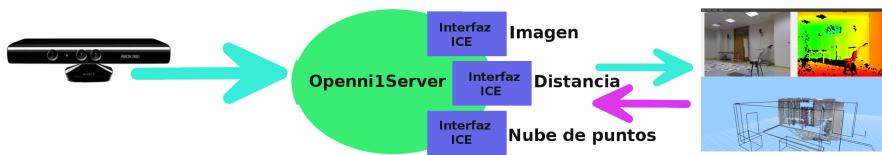


Figura 3.3: Ejemplo del uso del componente Openni1Server.

Primero se enciende el dron, se conecta la máquina a la red wifi suya y se ejecuta `Ardrone_server`. A través de suya, el cliente recibe tanto la información de la imagen (misma interfaz que `cameraserver`), como el Pose3D (posición y orientación del dron) y otras informaciones como pueden ser la batería restante por ejemplo (NavData). También se pueden enviar órdenes como la velocidad (CmdVel) u otras propias de aeronaves como despegar, aterrizar,...(`ArDroneExtra`). En la figura 3.4 se refleja éste funcionamiento.

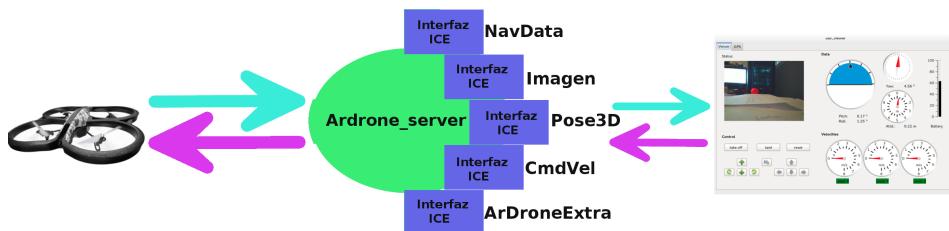


Figura 3.4: Ejemplo del uso del componente Ardrone_server.

3.2.4. Kobuki_driver + kobukiViewer

`Kobuki_driver`[8] es otro componente incluido en JdeRobot, escrito en C++. Este componente permite comunicarse con el robot Kobuki, ofreciendo varias interfaces tanto para recibir información como para enviar órdenes (Imagen, Pose3D, Motors, ...).

Primero se enciende el Kobuki, se conecta el robot por USB al PC y se ejecuta `kobuki_driver`. A través de suya, el cliente recibe tanto la información de la imagen (misma interfaz que `cameraserver`), como el Pose3D (posición y orientación del robot) y otras informaciones como pueden ser el láser por ejemplo (Laser). También se pueden enviar órdenes como la velocidad (Motors). En la figura 3.5 se refleja éste funcionamiento.

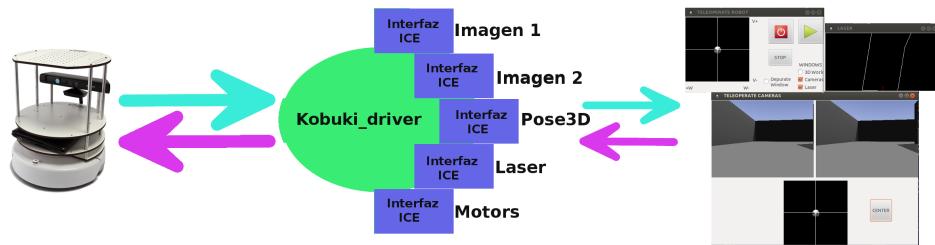


Figura 3.5: Ejemplo del uso del Kobuki.

3.3. HTML5

HTML5[10] (HyperText Markup Language, versión 5) es la quinta revisión importante del lenguaje básico de la *World Wide Web*, HTML. HTML5 especifica dos variantes de sintaxis para HTML: una «clásica», HTML (text/html), conocida como HTML5, y una variante XHTML conocida como sintaxis XHTML5 que deberá servirse con sintaxis XML (application/xhtml+xml). Esta es la primera vez que HTML y XHTML se han desarrollado en paralelo. La versión definitiva de la quinta revisión del estándar se publicó en octubre de 2014⁵.

HTML5 establece una serie de nuevos elementos y atributos que reflejan el uso típico de los sitios web modernos. El desarrollo de este lenguaje de marcado es regulado por el Consorcio W3C⁶.

Algunos de los nuevos elementos introducidos son: **video** (permite reproducir vídeos sin la necesidad de *plugins Flash*), **canvas** (Puede usarse para dibujar gráficos a través JavaScript), **header** (representa la cabecera de la web, donde normalmente está el menú), **progress** (permite mostrar una barra de progreso). En el siguiente código se puede ver la etiqueta video necesaria para el vídeo representado en la figura 3.6

```
<video style="width: 640px; height: 360px; left: 0px; top: 0px;
transform: none;" class="video-stream html5-main-video" src=
mediastream:https://www.youtube.com/54e5cca1-7aef-4865-ae1b-
baab9047b526"></video>
```

Nosotros usamos varios ingredientes asociados a HTML5 como son: **Canvas**, **WebWorkers** y **WebSockets**.

⁵Estándar: <http://www.w3.org/TR/html5/>

⁶web de W3C: <http://www.w3.org/>



Figura 3.6: Vídeo en Youtube.

3.3.1. Canvas

Consiste en una región dibujable definida en el código HTML con atributos de altura y ancho. El código JavaScript puede acceder a la zona a través de un conjunto completo de funciones similares a las de otras APIs comunes de dibujo 2D, permitiendo así que los gráficos sean generados dinámicamente. Algunos de los usos previstos incluyen construcción de gráficos, animaciones, juegos, y la composición de imágenes.

Canvas fue introducido inicialmente por Apple para su uso dentro de su propio componente de Mac OS X surgido en 2004, para empujar aplicaciones como *widgets* de *Dashboard* y el navegador Safari. Más tarde, en 2005, se adoptó en la versión 1.8 de los navegadores Firefox y Ópera en 2006. Fue estandarizado por el Grupo de Trabajo de Tecnología de Aplicación de hipertexto Web⁷ (WHATWG) sobre las nuevas especificaciones propuestas para tecnologías web de última generación. En la tabla 3.3.1 se puede ver desde qué versión de los diferentes navegadores se soporta Canvas.

Nosotros lo utilizamos tanto para visualizar las imágenes recibidas de las cámaras como para mostrar los modelos 3D de los robots.

⁷web de WHATWG: <https://whatwg.org/>

IE	Edge	Firefox	Chrome	Safari	Opera	IOS Safari	Android Browser	Chrome For Android
9	12	3.6	4	4	10.1	3.2	3	7

Cuadro 3.1: Soporte de Canvas

3.3.2. WebWorker

Uno de los mayores problemas que tienen las aplicaciones web es que JavaScript es un entorno de subproceso único, es decir, que no se pueden ejecutar varias secuencias de comandos al mismo tiempo. Los desarrolladores imitaban la “simultaneidad” utilizando técnicas como `setTimeout()` o `setInterval()` que lo único que hacen es programar la ejecución de una función, pero se sigue ejecutando sólo una función cada vez.

Los **WebWorkers** proveen un medio sencillo para que el contenido web ejecute *scripts* en hilos en segundo plano. El hilo *worker* puede realizar tareas sin interferir con la interfaz de usuario. No tienen acceso al DOM (Document Object Model es un conjunto de utilidades diseñadas para manipular documentos HTML de forma rápida y eficiente) ni a otras zonas sensibles para evitar problemas de concurrencia (en la figura 3.7 se pueden ver los elementos a los que tiene acceso cada hilo), por lo que el intercambio de información entre *Worker* e hilo principal se hace mediante paso de mensajes.

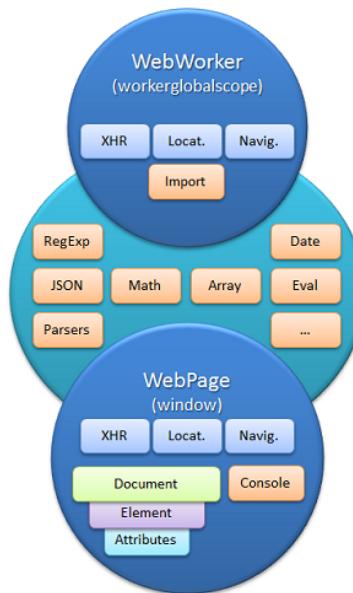


Figura 3.7: Elementos WebWorker.

En la tabla 3.3.2 se puede ver desde qué versión de los diferentes navegadores se soporta WebWorker.

IE	Edge	Firefox	Chrome	Safari	Opera	IOS Safari	Android Browser	Chrome For Android
10	12	3.5	4	4	11.5	5.1	4.4	47

Cuadro 3.2: Soporte para WebWorker

IE	Edge	Firefox	Chrome	Safari	Opera	IOS Safari	Android Browser	Chrome For Android
10	12	11	16	7	12.1	6.1	4.4	47

Cuadro 3.3: Soporte para WebSocket

3.3.3. WebSocket

WebSocket es una tecnología que proporciona un canal de comunicación bidireccional y *full-duplex* sobre un único *socket* TCP. Está diseñada para ser implementada en navegadores y servidores web, pero puede utilizarse por cualquier aplicación cliente/servidor. La API de *WebSocket* está siendo normalizada por el W3C⁸. 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).

En la tabla 3.3.3 se puede ver desde qué versión de los diferentes navegadores se soporta WebWorker.

3.4. CSS3

Hoja de estilo en cascada o CSS (*cascading style sheets*) es un lenguaje usado para definir y crear la presentación de un documento estructurado escrito en HTML. El *World Wide Web Consortium*⁹ es el encargado de formular la especificación de las hojas de estilo que servirán de estándar para los agentes de usuario o navegadores.

La idea que se encuentra detrás del desarrollo de CSS es separar la estructura de un documento de su presentación. La información de estilo puede ser definida en un documento

⁸<http://www.w3.org>

⁹web de W3C: <http://www.w3.org/>

separado o en el mismo documento HTML. En este último caso podrían definirse estilos generales con el elemento «style» o en cada etiqueta particular mediante el atributo «style».

CSS tiene una sintaxis muy sencilla, que usa unas cuantas palabras clave tomadas del inglés para especificar los nombres de varias propiedades de estilo, por ejemplo, *width* o *bottom-padding*.

A diferencia de las versiones anteriores, CSS3 está dividida en varios documentos separados, llamados “módulos”. Cada módulo añade nuevas funcionalidades a las definidas en CSS2, de manera que se preservan las anteriores para mantener la compatibilidad.

Nosotros lo usamos para modificar la apariencia de la página web a nuestro gusto.

3.5. JavaScript6

JavaScript6 o ECMAScript 2015 es la versión actual de la especificación del lenguaje ECMAScript conocida simplemente como “ES6”.

Agrega cambios significativos en la sintaxis para escribir aplicaciones complejas, incluyendo clases y módulos, definiéndolos sémanticamente en los mismos términos del modo estricto de la edición ECMAScript 5. Otras nuevas características incluyen iteradores *for/of loops*, generadores expresiones estilo Python, funciones de dirección, datos binarios, colecciones (mapas, sets, mapas débiles) y *proxies* (metaprogramación para objetos virtuales y *wrappers*).

El nuevo ingrediente de JavaScript6 que se utiliza en este TFG es el objeto *Promise*, que devuelve una promesa de tener un valor en algún momento en el futuro.

3.5.1. Promise

La interfaz *Promise* representa un *proxy* para un valor no necesariamente conocido cuando se crea la promesa. Permite asociar manejadores a un eventual éxito o fracaso de acciones asíncronas. Esto permite que los métodos asíncronos devuelven valores como métodos síncronos: en lugar del valor final, el método asíncrono devuelve una promesa de tener un valor en algún momento en el futuro.

Una Promesa puede estar en uno de estos estados:

- *Pending*: estado inicial, no cumplida o rechazada.

IE	Edge	Firefox	Chrome	Safari	Opera	IOS Safari	Android Browser	Chrome For Android
no soportado	12	29	33	7.1	20	8	4.4.4	47

Cuadro 3.4: Soporte para Promise

- *Fulfilled*: operación satisfactoria.
- *Rejected*: operación fallida.

Una promesa pendiente puede llegar a ser cumplida con un valor o rechazada con una razón. Cuando cualquiera de ellas sucede, los manejadores asociados se llaman mediante el método `then` de la promesa. (Si la promesa ya se ha cumplido o rechazado cuando se conecta un controlador correspondiente, el manejador será llamado, así que no hay condición de competencia entre el final de una operación asíncrona y sus manejadores). En la figura 3.8 pueden los estados y como se pasa a ellos.

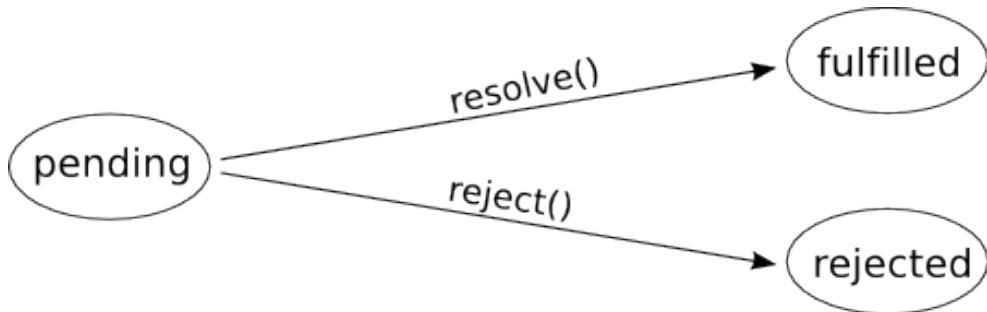


Figura 3.8: Vida de una Promise

En la tabla 3.5.1 se puede ver desde qué versión de los diferentes navegadores se soporta Promise.

3.6. WebGL

WebGL[14] es una especificación estándar, basada en OpenGL ES 2.0, que está siendo desarrollada actualmente para mostrar gráficos en 3D en navegadores web. Permite mostrar gráficos en 3D acelerados por hardware (GPU) en páginas web, sin la necesidad de *plugins* en cualquier plataforma que soporte OpenGL 2.0 u OpenGL ES 2.0.

WebGL carece de las rutinas matemáticas matriz eliminadas en OpenGL 3.0. Esta funcionalidad debe ser proporcionada por el usuario; este código necesario se complementa con frecuencia con una biblioteca de matriz tal como `glMatrix`, `TDL`, o `MJS`.

Como WebGL es una tecnología diseñada para trabajar directamente con la GPU (unidad de procesamiento gráfico) es difícil de codificar en comparación con otros estándares web más accesibles, por lo que ha surgido muchas bibliotecas JavaScript para resolver este problema: Curve3D, CubicVR, EnergizeGL, O3D, TDL, Three.js, X3DOM. BabylonJS...

Las más destacadas son Three.js, que es la más usada y con la que se trabaja en este proyecto, y BabylonJS que está desarrollada por Microsoft.

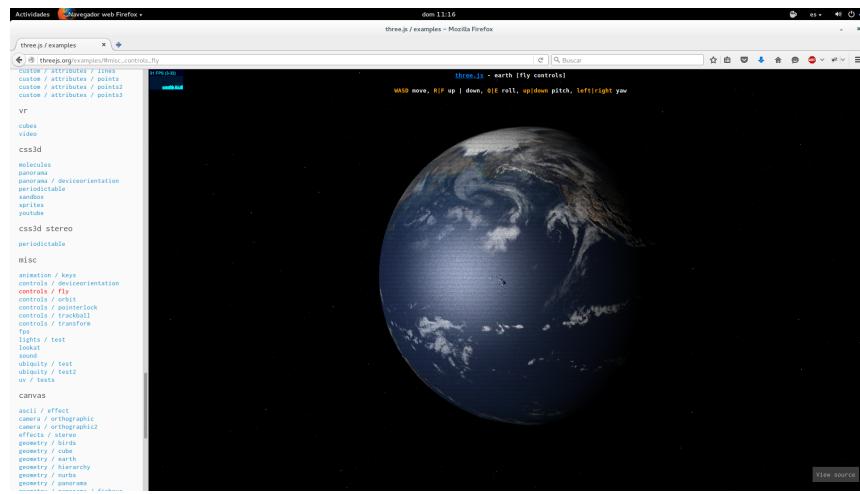


Figura 3.9: Ejemplo mundo WebGl con Three.js.

Nosotros la usamos para crear los modelos 3D de los robots.

3.7. ThreeJS

Three.js[24] es una biblioteca escrita en JavaScript para crear y mostrar gráficos animados por ordenador en 3D en un navegador Web y puede ser utilizada en conjunción con el elemento canvas de HTML5, SVG ó WebGL. Es la más usada para crear animaciones con WebGL (figura 3.9).

Características:

- Renderizadores: Canvas, SVG y WebGL.
- Efectos: anaglifo, bizo y la barrera de paralaje.
- Escenas: añadir y eliminar objetos en tiempo de ejecución; niebla.
- Cámaras: perspectiva y ortográfica; controladores: trackball, FPS, trayectoria y otras.

- Animación: armaduras, cinemática directa, cinemática inversa, *morphing* y fotogramas clave.
- Luces: ambiente, dirección, luz de puntos y espacios, sombras: emite y recibe.
- Materiales: Lambert, Phong, sombreado suave, texturas y otras.
- Shaders: el acceso a las capacidades del OpenGL Shading Language (GLSL): reflejos en la lente, pase profundo y una extensa biblioteca de post-procesamiento
- Objetos: mallas, partículas, sprites, líneas, cintas, huesos y otros.
- Geometría: plano, cubo, esfera, toroide, texto en 3D y otras; modificadores: torno, extrusión y tubo.
- Cargadores de datos: binario, imagen, JSON y escena.
- Utilidades: conjunto completo de funciones matemáticas en 3D, incluyendo tronco, matriz Quaternion, UVs y otras.

Incluso están añadiendo funciones para realidad virtual.

Nosotros la usamos para facilitar el acceso a WebGL.

3.8. JQuery

JQuery[27] es una biblioteca de JavaScript que permite simplificar la manera de interactuar con los documentos HTML, manipular el árbol DOM, manejar eventos, desarrollar animaciones y agregar interacción con la técnica AJAX a páginas web.

Es la biblioteca de JavaScript más utilizada y ofrece una serie de funcionalidades que de otra manera requerirían de mucho más código por lo que se ahorra tiempo y espacio.

Características:

- Selección de elementos DOM.
- Interactividad y modificaciones del árbol DOM, incluyendo soporte para CSS 1-3 y un *plugin* básico de XPath.
- Eventos.
- Manipulación de la hoja de estilos CSS.

JQuery	JavaScript
var divs = \$("div");	var divs = document.querySelectorAll ("div");
var newDiv = \$("<div/>");	var newDiv = document.createElement("div");
\$(“a”).click (function({ //code });	[].forEach.call (document.querySelectorAll (“a”),function (el){ el.addEventListener (“click”,function(){ //code }); });

Cuadro 3.5: Comparación JQuery, JavaScript

- Efectos y animaciones.
- Animaciones personalizadas.
- AJAX.
- Soporta extensiones.
- Utilidades varias como obtener información del navegador, operar con objetos y vectores, funciones para rutinas comunes, etc.
- Compatible con los navegadores Mozilla Firefox 2.0+, Internet Explorer 6+, Safari 3+, Opera 10.6+ y Google Chrome 8+.

La versión utilizada en el proyecto es la 1.11.3.

En la tabla 3.8 se puede ver una comparativa entre JQuery y los métodos por defecto de JavaScript.

3.9. Bootstrap

Bootstrap[28] es un *framework* o conjunto de herramientas de software libre 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.

Bootstrap fue desarrollado por Mark Otto y Jacob Thornton de Twitter, como un marco de trabajo para fomentar la consistencia a través de herramientas internas. Antes

de Bootstrap, se usaban varias librerías para el desarrollo de interfaces de usuario, las cuales guiaban a inconsistencias y a una carga de trabajo alta en su mantenimiento.

En agosto del 2011, Twitter liberó a Bootstrap como código abierto. En febrero del 2012, se convirtió en el proyecto de desarrollo más popular de GitHub y es usado por la NASA (figura 3.10), la universidad de Washington, la FIFA...

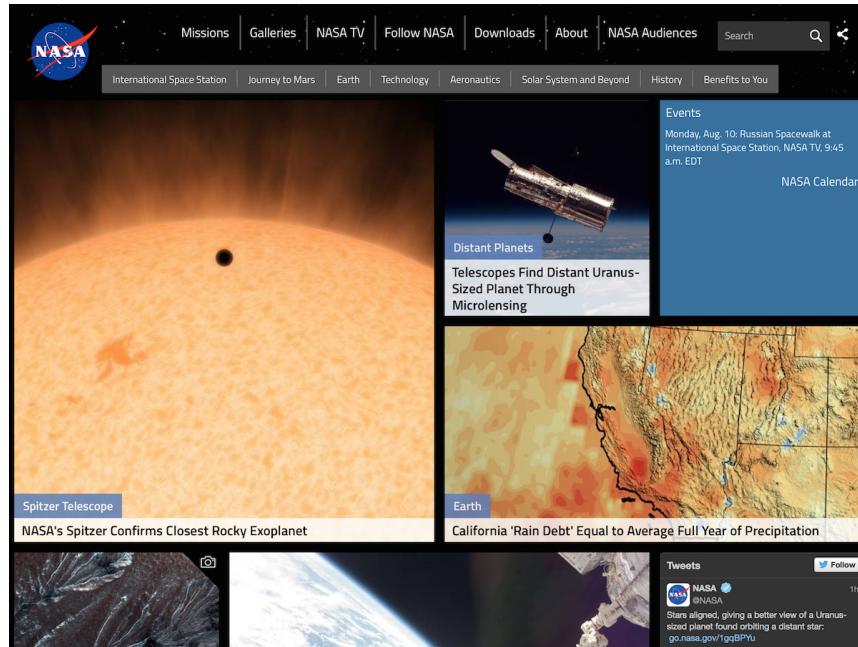


Figura 3.10: Web de la NASA.

En este proyecto se usa la versión 3 para la maquetación de la página web.

3.10. ICE for Javascript

ICE for Javascript o Ice-JS[22], es una nueva implementación de ICE adaptada para aplicaciones web que usan JavaScript. Como los navegadores web no pueden usar los protocolos comunes de ICE, sino que sólo pueden usar *websocket*[23], incluye un *plugin* para procesos en C++ que les permite usar este protocolo. Un esquema del funcionamiento de Ice-JS se encuentra en la figura 3.11.

En la versión 3.5 de ICE todavía era una beta y estaba aparte¹⁰, pero ya en la 3.6 está totalmente incluida en el paquete general de ICE. En este proyecto se usa la versión beta.

¹⁰descarga de Ice-JS <https://zeroc.com/labs/icejs/download.html>

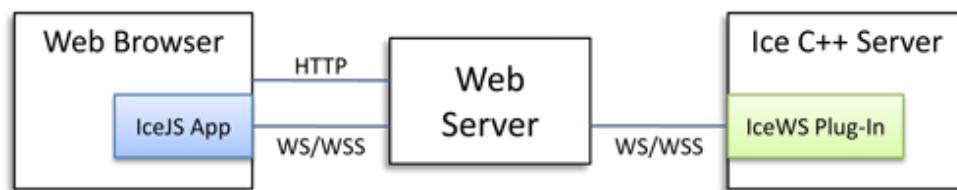


Figura 3.11: Esquema de funcionamiento de Ice-JS.

Capítulo 4

Diseño y programación

Una vez explicados en los capítulos anteriores los requisitos y las herramientas necesarias para la elaboración del proyecto, en este capítulo se va a explicar la solución desarrollada al problema planteado en el capítulo objetivos. Se va a comenzar dando una visión general del diseño y posteriormente se describirán sus detalles y la implementación de cada una de sus partes.

4.1. Diseño global

El diseño que se presenta se ha elegido porque cumple todos los objetivos y requisitos expuestos en el capítulo 2. La arquitectura del sistema tiene dos partes diferenciadas tal y como se observa en la figura 4.1: Servidores y clientes.

Servidores. Se han utilizado 4 servidores distintos, dos de sensores (el de cámara y el de Kinect) y dos de robots (Kobuki y Drone). Estos dos últimos pueden ser reales o simulados mediante Gazebo. Cada uno de los servidores se corresponde con componentes de JdeRobot ya existentes a los que se les ha añadido en la configuración la ubicación de un *plugin* para que puedan usar el protocolo *WebSockect*, necesario para que los navegadores web puedan comunicarse con estos servidores. Este *plugin* viene con la instalación de Ice-JS. Además, todos los componentes involucrados usan una interfaz ICE para comunicarse y transmitir esos datos al servidor Web o a otros componentes. Esto hace que cada componente pueda correr en una máquina distinta.

Los **clientes** están desarrollados mediante JavaScript, HTML y CSS. Para dotarlos de la mayor flexibilidad posible constan de 3 partes muy diferenciadas, como puede verse en la figura 4.2, conexión con servidor, núcleo e interfaz gráfico.

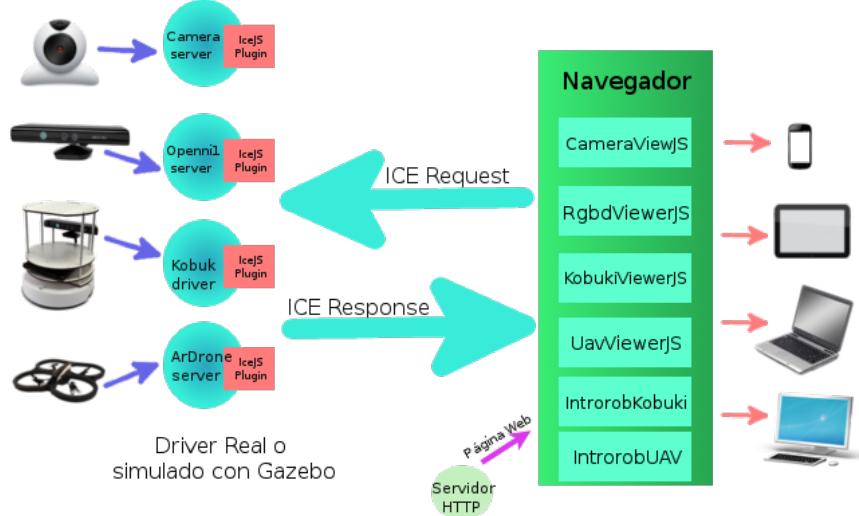


Figura 4.1: Arquitectura de JdeRobotWebClients

1. Conector. Está desarrollada íntegramente en JavaScript. Se encarga de la comunicación con el servidor de sensores o de robots. Permite crear la conexión y recibir los datos. Para ello primero crea un hilo separado que es el que se conecta con el servidor ICE. Una vez creada la conexión, el hilo principal mediante RPC's indica a este segundo hilo los métodos que debe ejecutar del servidor. Un cliente tiene tantos conectores como interfaces ICE tenga el servidor. Para comunicarse con dicho servidor utilizan ICE-JS que está escrito en JavaScript y permite utilizar interfaces ICE sobre el protocolo **WebSckect**.
2. Núcleo. Esta parte está desarrollada íntegramente en JavaScript. Se encarga del funcionamiento interno de cada uno de los *widgets* del cliente, ya sea mostrar los datos recibidos de los sensores o mandar los datos a los actuadores.
3. Interfaz gráfica. Engloba la parte HTML y CSS y algo de JavaScript de los clientes e indica cómo se organizan cada uno de los *widgets* del cliente dentro de la página web.

En total son 6 clientes: **CameraViewJS** (1), que recibe datos de *CameraServer*; **RgbdViewerJS** (2), que recibe datos de *Openni1Server*; **KobukiViewerJS** (3), que permite teleoperar robots Kobuki; **UavViewerJS** (4), que permite teleoperar drones y los dos que además de teleoperar, dan soporte para crear comportamiento autónomo en los robots o drones, **IntrorobKobukiJS** (5) e **IntrorobUavJS** (6).

Adicionalmente, hemos creado una página web, a modo de envoltorio único, que empaña en pestañas los seis clientes desarrollados.



Figura 4.2: Niveles de los clientes

4.1.1. Servidores

En esta sección se describe el funcionamiento de los componentes de JdeRobot ya existentes usados en este TFG. Su funcionamiento general se ha detallado en la sección 3.2. **Cameraserver** [3] es el componente encargado de acceder a la cámara web para entregar vía ICE las imágenes necesarias para *streaming* de vídeo. **Openni1Server** [4] es el componente necesario para acceder al sensor Kinect. De este componente se obtienen las imágenes de profundidad. **Ardrone_server** [7] es el componente necesario para acceder al drone. Hace de intermediario entre el drone y el cliente. **Kobuki_driver** [8] es el componente necesario para acceder al Kobuki. Hace de intermediario entre el Kobuki y el cliente.

Se ha tenido que hacer variaciones en sus ficheros de configuración para permitirles usar el protocolo *websocket* ya que es el único que usan los Navegadores web.

Para ello hay que indicarle en la configuración que use el *plugin* mediante la siguiente línea (No hace falta recompilar el código fuente del propio servidor):

```
# Ice-JS
Ice.Plugin.IceWS=IceWS:createIceWS
```

Una vez hecho esto ya se le puede indicar que escuche en un puerto mediante el protocolo *WebSocket* (ws) añadiendo al *endpoint*:

```
# Ice-JS
:ws -h IP -p PUERTO
```

Por ejemplo, el fichero de configuración de **cameraserver** quedaría así:

```
# client/server mode
```

```

# rpc=1 ; request=0
CameraSrv.DefaultMode=1
CameraSrv.TopicManager=IceStorm/TopicManager:default -t 5000 -p
10000

#General Config
CameraSrv.Endpoints=default -h 0.0.0.0 -p 9999:ws -h 0.0.0.0 -p
11000
CameraSrv.NCameras=1
CameraSrv.Camera.0.Name=cameraA
#0 corresponds to /dev/video0, 1 to /dev/video1, and so on...
CameraSrv.Camera.0.Uri=0
CameraSrv.Camera.0.FramerateN=25
CameraSrv.Camera.0.FramerateD=1
CameraSrv.Camera.0.Format=RGB8
CameraSrv.Camera.0.ImageWidth=640
CameraSrv.Camera.0.ImageHeight=480

# Ice-JS
Ice.Plugin.IceWS=IceWS:createIceWS

# If you want a mirror image, set to 1
CameraSrv.Camera.0.Mirror=1

```

4.1.2. Clientes

Son los encargados de manejar el contenido de cada *widget* de la página web.

A la hora de crearlos, reciben una variable `config` que debe incluir todos los datos necesarios para su creación (identificador de los *widgets*, servidor y *endpoint* de cada conector usado,...).

Tienen todos los mismos métodos: `start`, `stop`, `restart`, `setConfig` e `isRunning`. El funcionamiento de todos los clientes es el mismo: primero se crea el cliente, después se inicia con `start`, en este momento se comprueban las variables de configuración (en caso de ocurrir algún problema se da un aviso y no se inicia el cliente), se crea el contenido de los *widgets*, se crean los conectores y se inician. Para detener el cliente se usa `stop` y en

caso de querer cambiar algún parámetro de la configuración se usa `setConfig`. Si el cliente está funcionando en este momento hay que reiniciarlo con `restart`. `isRunning` devuelve un booleano indicando si el cliente está funcionando o no.

Una vez presentado el diseño global del sistema detallaremos en las siguientes secciones cada uno de los clientes desarrollados.

4.2. CameraViewJS

Este cliente se conecta con un servidor de WebCam, por ejemplo CameraServer, y muestra en un `Canvas` de HTML5 la imagen y los fotogramas por segundo (FPS) en el nodo HTML que se le indique (figura 4.3).

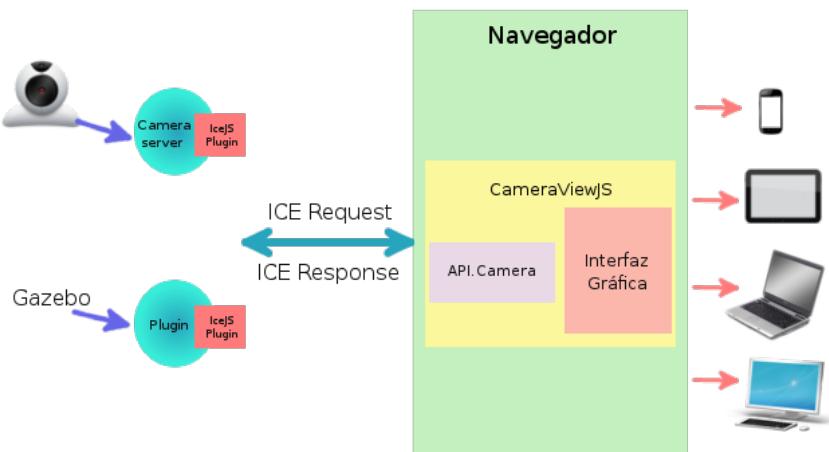


Figura 4.3: Arquitectura de CameraViewJS

La variable `config` para crearlo contiene:

- `serv`: dirección y puerto del servidor (`dir:address, port:port`).
- `camename`: *endpoint* del servidor, por defecto “cameraA”.
- `camid`: id del canvas que muestra la imagen.
- `fpsid`: id del elemento donde se pone el FPS.

4.2.1. Conektor

El único conector que utiliza CameraViewJS es **API.Camera**:

Al igual que todos los conectores para sensores, su funcionamiento es el siguiente (detallado en la figura 4.4): En el momento de crear el objeto se establecen las variables de configuración y se crea el *WebWorker*. Mediante la función `connect` se establece una promesa que se resuelve una vez esté establecida la conexión con el servidor. Para ello se manda un mensaje al *WebWorker* indicando que se conecte y éste a su vez comienza la conexión con el servidor. Una vez establecida se envía al hilo principal un mensaje indicando que está establecida y éste resuelve la promesa.

```

this.connect = function () {
    this.conPromise = new Promise(
        // The resolver function is called with the ability to
        resolve or
        // reject the promise
        function(resolve, reject) {
            self.w.onmessage = function(mes){
                if (mes.data){
                    self.isRunning = true;
                    resolve();
                }else{
                    self.isRunning = false;
                    reject();
                }
            };
            self.w.postMessage({func:"connect", serv:self.server,
epname: self.epname});
        });
}

```

Todas las funciones que se ejecuten, ya sea pidiendo o enviando datos se quedan paradas esperando a que se resuelva y una vez hecho se ejecutan con normalidad.

```

this.getImage = function(){
    this.conPromise.then(
        function() {

```

```

        self.w.onmessage = self.onmessage || self.
onmessageDefault;
        self.w.postMessage({func:"getImage", imgFormat: self.
imgFormat});
    });

}

```

En el caso de los sensores se puede iniciar un *streaming* que consiste en darle la orden al *WebWorker* y éste se dedica a hacer peticiones al servidor una tras otra y a devolver el resultado al hilo principal hasta que se le indique que pare.

```

this.startStreaming = function(){
    this.conPromise.then(
        function() {
            self.w.onmessage = self.onmessage || self.
onmessageDefault;
            self.w.postMessage({func:"startImageStream", imgFormat
: self.imgFormat});
            self.toError=setTimeout(self.conErr, self.timeoutE);
        });
};

this.stopStreaming = function(){
    this.conPromise.then(
        function() {
            self.w.postMessage({func:"stopImageStream"});
            if(self.toError){
                clearTimeout(self.toError);
                toError=undefined;
            }
        });
};

```

Hablando ya en concreto de `API.Camera`, permite conectar con la interfaz Camera de JdeRobot. El contenido de `config` es la siguiente:

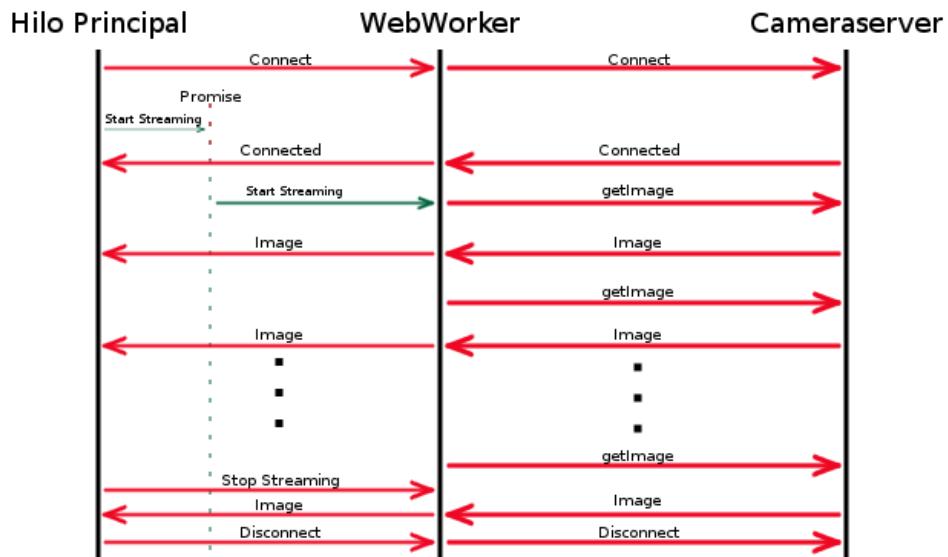


Figura 4.4: Mensajes API.Camera.

- *server*: dirección y puerto del servidor (dir:address, port:port).
- *epname*: endpoint del servidor, por defecto “cameraA”.
- *imgFormat*: formato de imagen que se va a pedir al servidor, por defecto “RGB8”.

El contenido de la variable **data** (datos recibidos de la cámara) es el siguiente:

- *width*: Ancho en píxeles de la imagen recibida.
- *height*: Alto en píxeles de la imagen recibida.
- *imgData*: Imagen preparada para ser mostrada en un **canvas** de HTML5.
- *pixelData*: Imagen recibida desde el servidor.
- *fps*: fotogramas por segundo recibidos.

y su lista de métodos es:

- *createWork*: Crea el *WebWorker* (se ejecuta cuando se crea el conector).
- *deleteWork*: Elimina el *WebWorker*.
- *connect*: Inicia la conexión con el servidor.

- *disconnect*: Desconecta del servidor.
- *getImage*: Pide una imagen al servidor.
- *startStreaming*: Activa el *streaming* de imágenes (es como ejecutar *getImage* constantemente).
- *stopStreaming*: Detiene el *streaming* de imágenes.
- *getDescription*: Pide la descripción de la cámara al servidor y la almacena en la variable **description**.

4.2.2. Núcleo

Cuando se inicia el cliente **CameraViewJS**, éste inicia un *streaming* en el API `.Camera`. Cada vez que se recibe la imagen, crea un **canvas** nuevo donde poner la imagen de la cámara recibida, después lo introduce como si fuera una foto en el **canvas** dado en la configuración. Esto se hace porque usando un sólo **canvas** si se cambia el tamaño de la página web, por ejemplo, girando la tablet se deforma la imagen recibida de la cámara. Mientras que de esta manera se adapta la imagen al tamaño del **canvas** externo como si fuera cualquier archivo de imagen.

```

camera.onmessage = function (event){
    camera.onmessageDefault(event);
    var respwork = camera.data;
    //camera
    var canvas2 = document.createElement('canvas');
    var ctx2=canvas2.getContext("2d");
    var imgData=ctx2.getImageData(0,0,respwork.width,respwork
.height);
    ctx2.canvas.width=respwork.width;
    ctx2.canvas.height=respwork.height;
    var ctx=canvas.getContext("2d");
    imgData.data.set(respwork.imgData);
    ctx2.putImageData(imgData,0,0);
    ctx.drawImage(canvas2, 0, 0,ctx.canvas.width,ctx.canvas.
height);

//FPS
if (respwork.fps) {

```

```

        fps.html(Math.floor(respwork.fps));
    }
};
```

4.2.3. Interfaz gráfico

Para la interfaz de los clientes se ha optado por un color oscuro de fondo para reducir el consumo de las pantallas de los dispositivos. Además se usa **Bootstrap** para todos los elementos, organización y para hacer la web responsive (se adapta a cualquier dispositivo), como se puede ver en la figura 4.5.

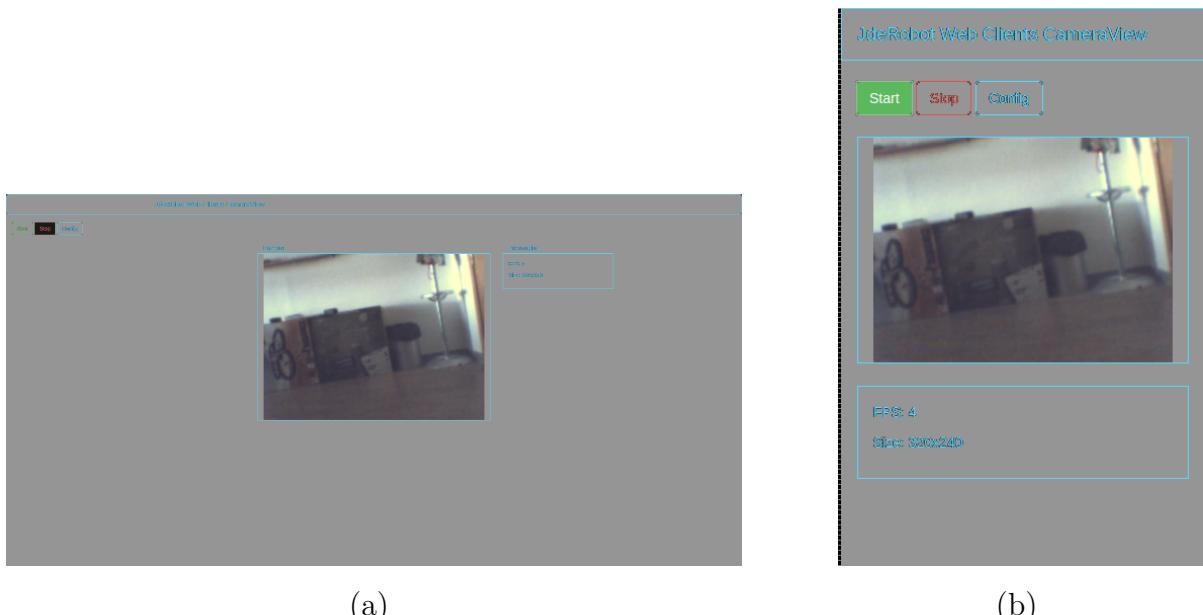


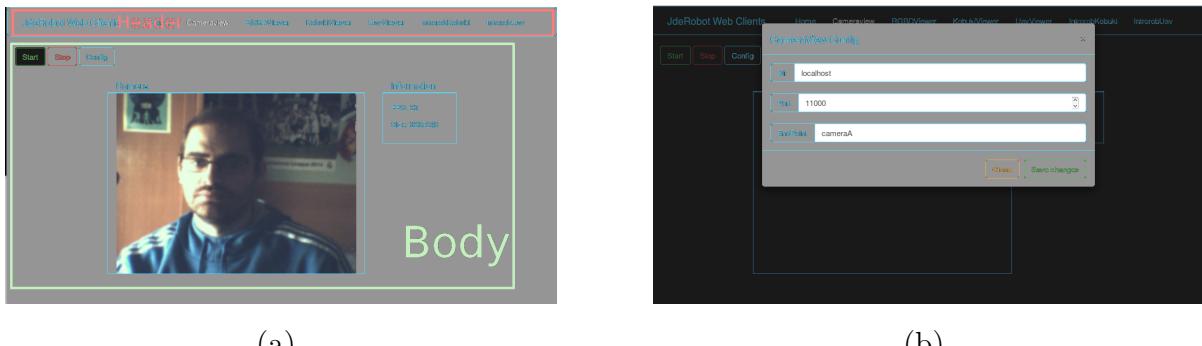
Figura 4.5: Escritorio (a) y Móvil (b)

La interfaz de cada cliente consta de un fichero HTML y varios CSS y JavaScript. El *body* de del HTML se divide es 3 partes (figura 4.6):

- **Header:** En todos es igual y contiene la cabecera de la página web.

```

<header id="header">
    <nav class="navbar navbar-inverse" role="banner">
        <div class="container">
            <div class="navbar-header">
```



(a)

(b)

Figura 4.6: Header y Body (a) y Modal (b)

```

    <a class="navbar-brand" href="#">> JdeRobot Web Clients</a> </div>
</div>
<!--/.container-->
</nav>
<!--/nav-->
</header>

```

- **Body:** es completamente diferente en cada cliente y contiene los *widgets* propios de cada uno. En el caso de CameraViewJS consta de 3 botones (`start`, `stop`, `config`), el `canvas` para mostrar la imagen de la cámara y un cuadro donde se muestran los FPS y el tamaño de la imagen recibida.

```

<div id="body" class="container">
  <div class="row">
    <div id="buttons" class="col-xs-12 col-sm-12 col-md-12
    col-lg-12">
      <p>
        <button id="start" type="button" class="btn btn-md
        btn-success">Start</button>
        <button id="stop" type="button" class="btn btn-md
        btn-danger">Stop</button>
        <button id="config" type="button" class="btn btn-
        info" data-toggle="modal" data-target="#configure">Config</
        button>
      </p>
    </div>
  </div>

```

```
</div>

<div class="row">
    <div class="col-xs-12 col-sm-8 col-md-6 col-lg-4 col-lg-
offset-4 col-md-offset-2">
        <div class="border-carbon panel panel-info">
            <div class="letrero panel-heading hidden-sm
hidden-xs">
                <span class="panel-title">Camera</span>
            </div>
            <div class="panel-body padding0 border-blue
container-fluid">
                <canvas id="camView" class="col-xs-12 col-sm-12
col-md-12 col-lg-12 cam">Your browser does not support the
HTML5 canvas tag.</canvas>
            </div>
        </div>
    </div>
    <div class="col-xs-12 col-sm-2 col-md-2 col-lg-2">
        <div class="border-carbon panel panel-info">
            <div class="letrero panel-heading hidden-sm
hidden-xs">
                <span class="panel-title">Information</span>
            </div>
            <div class="panel-body border-blue container-fluid
">
                <p><span class="bold">FPS: </span><span id="fps
"></span></p>
                <p><span class="bold">Size: </span><span id="
size"></span></p>
            </div>
        </div>
    </div>
</div>
```

- **Modal:** Contiene un formulario para guardar la configuración de cada cliente y la única diferencia entre los clientes es en el número de elementos del formulario.

```
<div class="modal fade" id="configure" tabindex="-1" role="dialog" aria-labelledby="configLabel">
<div class="modal-dialog" role="document">
<div class="modal-content bg-carbon">
<div class="modal-header bg-orange">
<button type="button" class="close" data-dismiss="modal" aria-label="Close"><span aria-hidden="true">&times;</span></button>
<h4 class="modal-title" id="configLabel">CameraView Config</h4>
</div>
<div class="modal-body">
<div class="input-group">
<span class="input-group-addon" id="basic-addon1">Dir</span>
<input id="dir" type="text" class="form-control" placeholder="Direction" value="localhost" aria-describedby="basic-addon1">
</div>
<br>
<div class="input-group">
<span class="input-group-addon" id="basic-addon1">Port</span>
<input id="port" type="number" class="form-control" value="11000" aria-describedby="basic-addon1">
</div>
<br>
<div class="input-group">
<span class="input-group-addon" id="basic-addon1">EndPoint</span>
<input id="ep" type="text" class="form-control" value="cameraA" aria-describedby="basic-addon1">
</div>
</div>
<div class="modal-footer">
<button type="button" class="btn btn-warning" data-dismiss="modal">Close</button>
```

```
<button id="save" type="button" class="btn btn-success"
" data-dismiss="modal">Save changes</button>
</div>
</div>
</div>
```

Debajo de estos tres grandes bloques ya se sitúan los ficheros JavaScript necesarios para cada cliente. Así se agiliza la carga de la web.

4.3. RgbdViewerJS

Este cliente se conecta con un servidor de `Openni1Server` y muestra en dos `Canvas` de `HTML5` la imagen `RGB` y la de `distancia`. Los fotogramas por segundo (FPS) de cada imagen se muestran en su respectivo nodo `HTML`, además en un tercer `canvas` muestra en `3D` una escena de la combinación de estas dos imágenes (figura 4.7). Este último elemento es la gran diferencia respecto al cliente anterior.

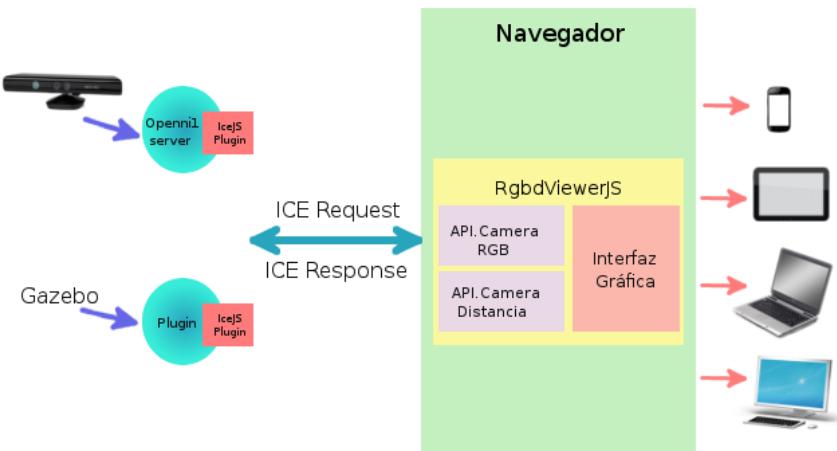


Figura 4.7: Arquitectura de RgbdViewerJS

La variable `config` para crearlo contiene:

- `serv`: dirección y puerto del servidor (`dir:address, port:port`).
- `camename`: *endpoint* del servidor de `RGB`, por defecto “`cameraA`”.
- `camid`: id del `canvas` que muestra la imagen `RGB`.
- `fpscamid`: id del elemento donde se pone el FPS `RGB`.
- `depepname`: *endpoint* del servidor de `distancia`, por defecto “`cameraB`”.
- `depthid`: id del `canvas` que muestra la imagen de `distancia`.
- `fpsdepid`: id del elemento donde se pone el FPS de la cámara de `distancia`.
- `modelid`: id del `canvas` que muestra la reconstrucción `3D`.

Este cliente usa dos *conectores* `API.Camera`, ya explicados en `CameraViewJS`. El intercambio de mensajes con el servidor es el mismo que en el cliente anterior.

4.3.1. Núcleo

Cuando se inicia el cliente, éste inicia un *streaming* en los dos `API.Camera` y se crea el entorno ThreeJS para mostrar la escena. El tratamiento de las imágenes recibidas es el mismo que en `CameraViewJS` pero añadiendo el procesado necesario para crear la escena 3D (figura 4.8).

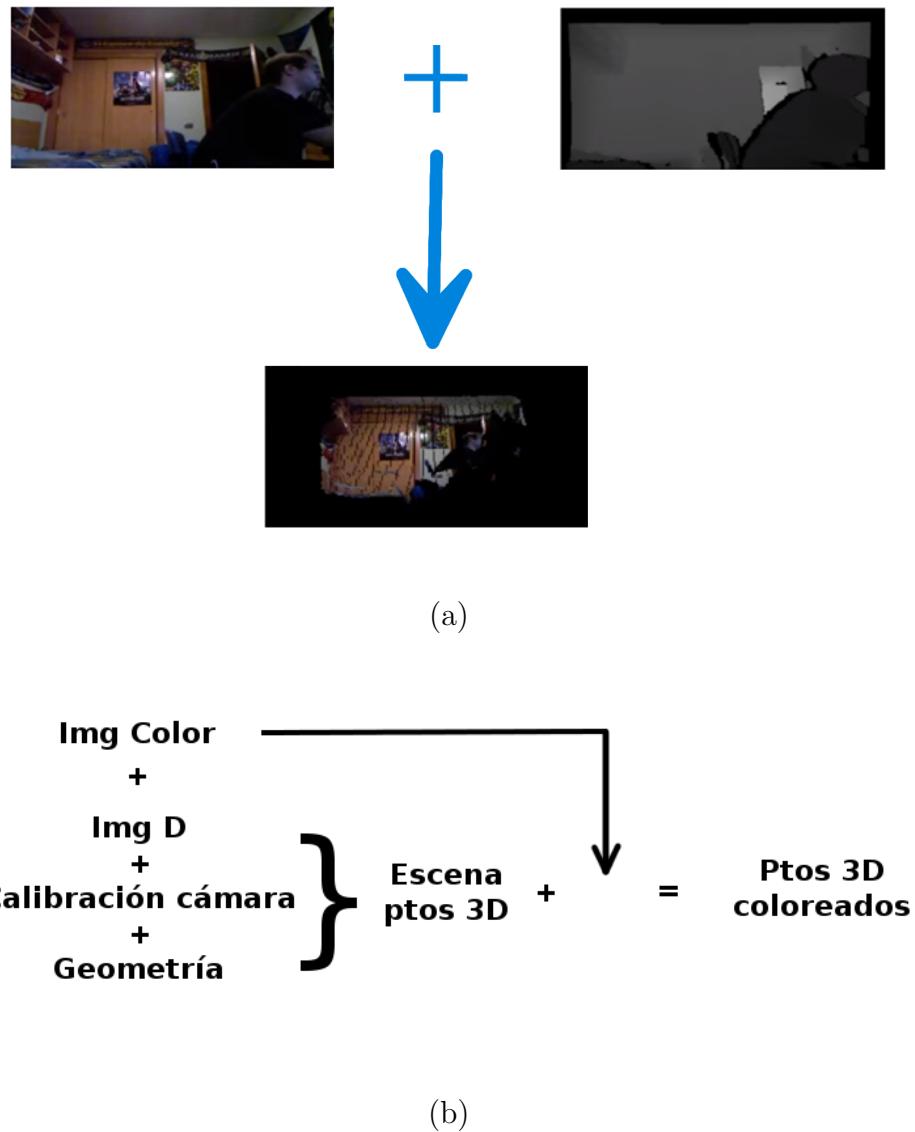


Figura 4.8: Creación de la escena 3D

Para crearla primero hay que crear un *buffer* de puntos con sus respectivas coordenadas 3D, además de otro *buffer* con el color, en formato RGB, que se va a aplicar en cada punto. Para crear el primer *buffer* partimos de la imagen de distancia que sólo nos da valores del eje Z. Aplicando el modelo *Pin Hole*, mostrado en la figura 4.9 se ha creado una adaptación

de la librería Progeo ya existente en JdeRobot para JavaScript y que permite mediante la aplicación de una matriz de translación y rotación pasar de los píxeles y el valor Z de cada uno a una coordenada (X,Y,Z) real. Dicha matriz se calcula mediante las coordenadas del Kinect y sus ángulos sobre los ejes con las siguientes fórmulas:

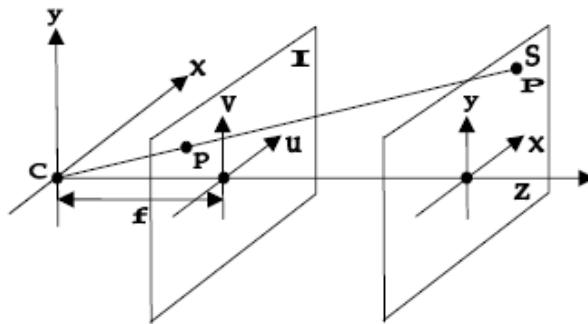


Figura 4.9: Modelo de cámara Pin Hole

```

rt[0][0] = Math.cos(this.roll) * Math.cos(this.pitch);
rt[0][1] = (-Math.sin(this.roll) * Math.cos(this.yaw) + Math.
cos(this.roll) * Math.sin(this.pitch) * Math.sin(this.yaw));
rt[0][2] = (Math.sin(this.roll) * Math.sin(this.yaw) + Math.
cos(this.roll) * Math.sin(this.pitch) * Math.cos(this.yaw));
rt[0][3] = this.position.x;
rt[1][0] = Math.sin(this.roll) * Math.cos(this.pitch);
rt[1][1] = (Math.cos(this.roll) * Math.cos(this.yaw) + Math.
sin(this.roll) * Math.sin(this.pitch) * Math.sin(this.yaw));
rt[1][2] = (-Math.cos(this.roll) * Math.sin(this.yaw) + Math
.sin(this.roll) * Math.sin(this.pitch) * Math.cos(this.yaw));
rt[1][3] = this.position.y;
rt[2][0] = Math.sin(this.pitch);
rt[2][1] = Math.cos(this.pitch) * Math.sin(this.yaw);
rt[2][2] = Math.cos(this.yaw) * Math.cos(this.pitch);
rt[2][3] = this.position.z;
rt[3][0] = 0;
rt[3][1] = 0;
rt[3][2] = 0;
rt[3][3] = 1;

```

Una vez hecho esto sólo hay que aplicar la Matriz a cada punto y ya se tendría el *buffer* de puntos en 3D, el *buffer* RGB es el valor RGB del píxel en dicha imagen.

```

p.x = point.x * rt[0][0] +
      point.y * rt[0][1] +
      point.z * rt[0][2] +
      point.h * rt[0][3];

p.y = point.x * rt[1][0] +
      point.y * rt[1][1] +
      point.z * rt[1][2] +
      point.h * rt[1][3];

p.z = - point.x * rt[2][0] +
      point.y * rt[2][1] +
      point.z * rt[2][2] +
      point.h * rt[2][3];

p.h = point.x * rt[3][0] +
      point.y * rt[3][1] +
      point.z * rt[3][2] +
      point.h * rt[3][3];

```

Cuando ya se tienen los dos *buffers* se crea la nube de puntos, se borra la anterior nube y se agrega al modelo 3D la nueva.

```

//prepare the points and each point color
for (var i=0; i<depth.data.length;i+=depth.width*3*SAMPLE){
    x=-depth.width/2;
    for (var j=0; j<depth.width*3;j+=3*SAMPLE){
        z=(depth.data[i+j+1]<<8) + depth.data[i+j+2]);
        points[a]=x;
        points[a+1]=y;
        points[a+2]=z;

        color.setRGB(rgb.data[i+j]/255,rgb.data[i+j+1]/255,rgb
        .data[i+j+2]/255);

        colors[ a ]      =color.r;
        colors[ a+ 1 ] =color.g;
    }
}

```

```
    colors[ a+ 2 ] = color.b;

    a+=3;

    x+=(w/depth.width)*SAMPLE;
}
y+=(h/depth.height)*SAMPLE;
}

var hpoints = conicProjectionCloudHPoint3D(
cloudPoint2CloudHPoint3D(points),tphCamera);
var cloudpoint = cloudHPoint3D2CloudPoint(hpoints);

//I do the object using the points, its color and a material
that shows the color of each point
var geometry = new THREE.BufferGeometry();
geometry.addAttribute( 'position', new THREE.BufferAttribute(
new Float32Array(cloudpoint), 3 ) );
geometry.addAttribute( 'color', new THREE.BufferAttribute(
new Float32Array(colors), 3 ) );

var material = new THREE.PointsMaterial( { vertexColors:
THREE.VertexColors} );
image= new THREE.Points( geometry, material);

if (lastCP){
scenegl.remove(lastCP);
}
scenegl.add(image);
lastCP=image;

depth.update=false;
rgb.update=false;
}
```

4.3.2. Interfaz gráfico

Cómo ya se comentó en CameraViewJS, el elemento `header` y el elemento `modal` vienen a ser iguales con alguna pequeña modificación, por lo que nos centramos ahora en el elemento `body`.

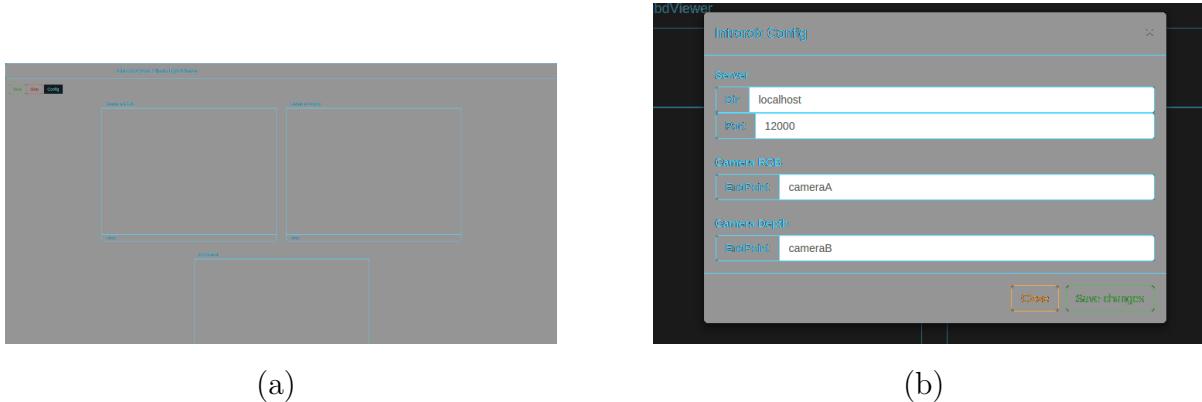


Figura 4.10: Interfaz (a) y Modal (b)

El `body` consta de 3 `canvas` ordenados mediante el sistema *Grid* de Bootstrap, los dos de arriba para las imágenes RGB y de distancia, con un hueco debajo para poner los fotogramas por segundo de cada imagen y el tercero para la escena 3D (figura 4.11).

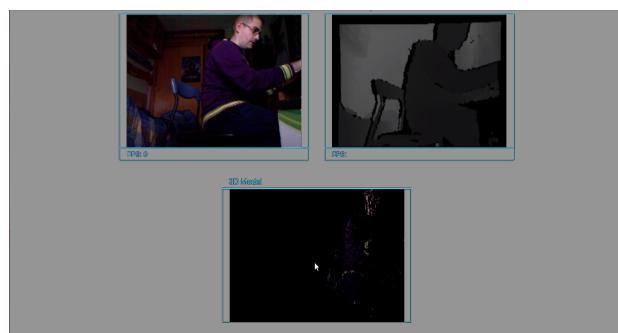


Figura 4.11: Body de RgbdViewerJS

```
<div id="body" class="container">
  <div class="row">
    <div id="buttons" class="col-xs-12 col-sm-12 col-md-12 col-lg-12"><p>
```

```
        <button id="start" type="button" class="btn btn-md btn-success">Start</button>
        <button id="stop" type="button" class="btn btn-md btn-danger">Stop</button>
        <button id="config" type="button" class="btn btn-info" data-toggle="modal" data-target="#configure">Config</button>
    </p>
    </div>
</div>
<div class="row">
    <div class="col-xs-12 col-sm-8 col-md-6 col-lg-4 col-lg-offset-2 col-md-offset-1">
        <div class="border-carbon panel panel-info">
            <div class="letrero panel-heading hidden-sm hidden-xs">
                <span class="panel-title">Camera RGB</span>
            </div>
            <div class="panel-body padding0 border-blue container-fluid">
                <canvas id="camView" class="col-xs-12 col-sm-12 col-md-12 col-lg-12 cam">Your browser does not support the HTML5 canvas tag.</canvas>
            </div>
            <div class="panel-footer border-blue letrero"><span class="bold">FPS: </span><span id="fps"></span></div>
            </div>
        </div>
        <div class="col-xs-12 col-sm-8 col-md-6 col-lg-4">
            <div class="border-carbon panel panel-info">
                <div class="letrero panel-heading hidden-sm hidden-xs">
                    <span class="panel-title">Camera Depth</span>
                </div>
                <div class="panel-body padding0 border-blue container-fluid">
                    <canvas id="camView2" class="col-xs-12 col-sm-12 col-md-12 col-lg-12 cam">Your browser does not support the HTML5 canvas tag.</canvas>
```

```

        </div>
        <div class="panel-footer border-blue letrero"><span
class="bold">FPS: </span><span id="fps2"></span></div>
    </div>
</div>

<div class="row">
    <div class="col-xs-12 col-sm-8 col-md-6 col-lg-4 col-lg-
offset-4 col-md-offset-2">
        <div class="border-carbon panel panel-info">
            <div class="letrero panel-heading hidden-sm hidden-xs
">
                <span class="panel-title">3D Model</span>
            </div>
            <div class="panel-body padding0 border-blue container-
fluid">
                <canvas id="model" class="col-xs-12 col-sm-12 col-
md-12 col-lg-12 cam">Your browser does not support the HTML5
canvas tag.</canvas>
            </div>
        </div>
    </div>
</div>
</div>

```

Además en este modelo se puede manejar la cámara de observación de la escena para poder ver la reconstrucción desde distintos puntos de vista (figura 4.12)

4.4. KobukiViewerJS

Este cliente se conecta con un robot con ruedas, modelo Kobuki, para poderlo teleoperar desde la web. Para ello usa dos API.Camera, un API.Laser, un API.Pose3D y un API.Motors (figura 4.13).

La variable `config` para crearlo contiene:

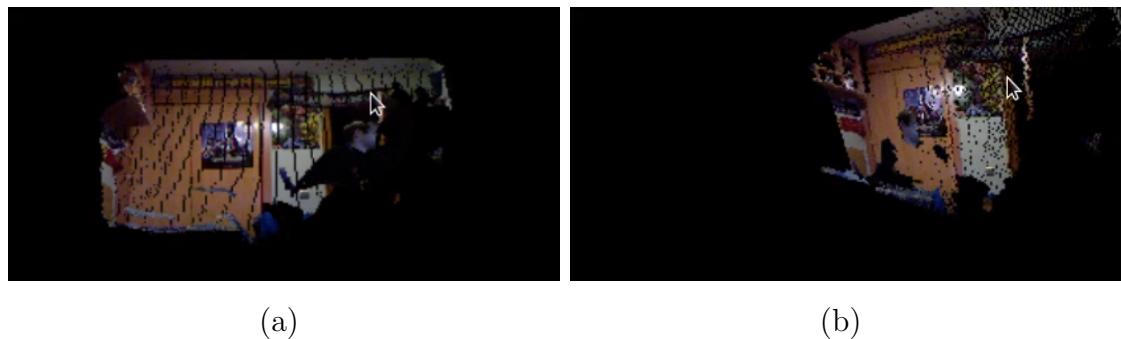


Figura 4.12: Visualización con cámaras de observación de frente (a) y a la derecha (b)

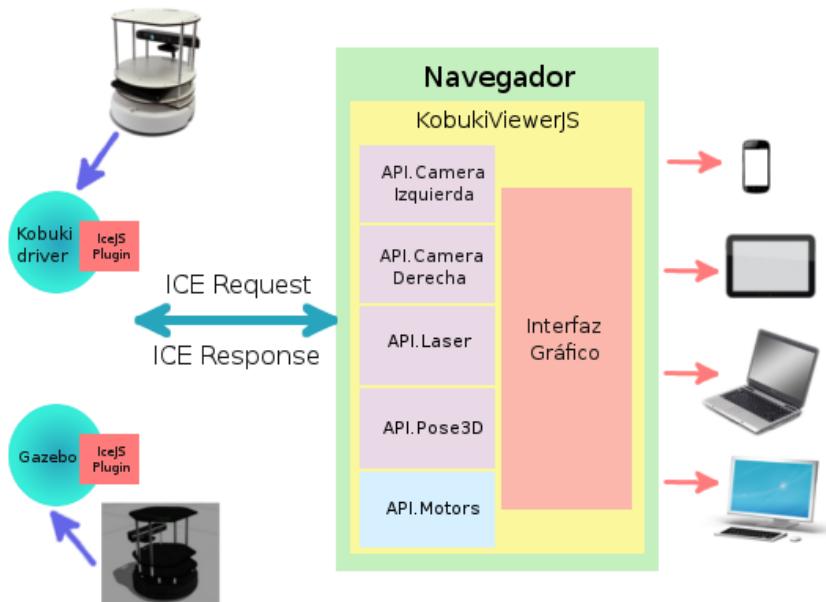


Figura 4.13: Arquitectura de KobukiViewerJS

- *camleftserv*: dirección y puerto del servidor de la cámara izquierda (dir:address, port:port).
- *camleftepname*: endpoint del servidor de la cámara izquierda.
- *camleftid*: id del canvas que muestra la imagen de la cámara izquierda.
- *camrightserv*: dirección y puerto del servidor de la cámara derecha (dir:address, port:port).
- *camrightepname*: endpoint del servidor de la cámara derecha.
- *camrightid*: id del canvas que muestra la imagen de la cámara derecha.
- *motorserv*: dirección y puerto del servidor de los motores (dir:address, port:port).

- *motorsepname*: *endpoint* del servidor de los motores.
- *controlid*: id del canvas que muestra el control para teleoperar el robot.
- *modelid*: id del canvas que muestra una representación en 3D del robot.
- *stopbtnid*: id del botón que detiene el robot.
- *pose3dserv*: dirección y puerto del servidor de Pose3D (dir:address, port:port).
- *pose3depname*: *endpoint* del servidor de Pose3D.
- *laserserv*: dirección y puerto del servidor del laser(dir:address, port:port).
- *laserepname*: *endpoint* del servidor del laser.
- *laserid*: id del canvas que muestra una representación en 2D del láser.

4.4.1. Conectores

KobukiViewerJS utiliza 4 conectores: API.Camera, API.Laser, API.Pose3D y API.Motors. API.Camera ya está explicado anteriormente, así que vamos a seguir por los demás.

API.Laser, como todo conector con sensores, básicamente funciona igual que API.Camera, pero con las siguientes diferencias:

Permite conectar con la interfaz Laser de JdeRobot. El contenido de `config` es la siguiente:

- *server*: dirección y puerto del servidor (dir:address, port:port).
- *epname*: *endpoint* del servidor, por defecto “Laser”.

El contenido de la variable `data` es el siguiente:

- *distanceData*: Array de distancias recibidas del servidor.
- *numLaser*: Número de láseres recibidos del servidor.
- *canv2dData*: Representación de `distanceData` para ser mostrada en un canvas de HTML5.
- *array3dData*: Representación de `distanceData` para ser mostrada en el modelo 3D.

y su lista de métodos es:

- *createWork*: Crea el *WebWorker* (se ejecuta cuando se crea el conector).
- *deleteWork*: Elimina el *WebWorker*.
- *connect*: Inicia la conexión con el servidor.
- *disconnect*: Desconecta del servidor.
- *getLaser*: Pide una medida de lásers al servidor.
- *startStreaming*: Activa el **Streaming** de **Laser** (es como ejecutar *getLaser* constantemente).
- *stopStreaming*: Detiene el **Streaming** de **Laser**.

A **API.Pose3D** le pasa lo mismo que a **API.Laser**:

Permite conectar con la interfaz Pose3D de JdeRobot. El contenido de **config** es la siguiente:

- *server*: dirección y puerto del servidor (dir:address, port:port).
- *epname*: *endpoint* del servidor, por defecto “Pose3D”.

El contenido de la variable **data** es el siguiente:

- *x,y,z*: Coordenadas X,Y,Z que representa las posición del robot en el espacio.
- *q0,q1,q2,q3*: cuaternión que representa la orientación del robot.
- *yaw,pitch,roll*: También representan la orientación pero de manera más cómoda de usar, son en radianes.

y su lista de métodos es:

- *createWork*: Crea el *WebWorker* (se ejecuta cuando se crea el conector).
- *deleteWork*: Elimina el *WebWorker*.
- *connect*: Inicia la conexión con el servidor.
- *disconnect*: Desconecta del servidor.

- *getPose3D*: Pide un Pose3D al servidor.
- *startStreaming*: Activa el Streaming de Pose3D (es como ejecutar *getPose3D* constantemente).
- *stopStreaming*: Detiene el Streaming de Pose3D.

El caso de **API.Motors** ya es algo diferente porque ya no conecta con un sensor, sino con un actuador. La parte de la conexión con el servidor es igual, pero ya el intercambio de mensajes es diferente, porque en este caso se envían datos en vez de recibirlos (figura 4.14)

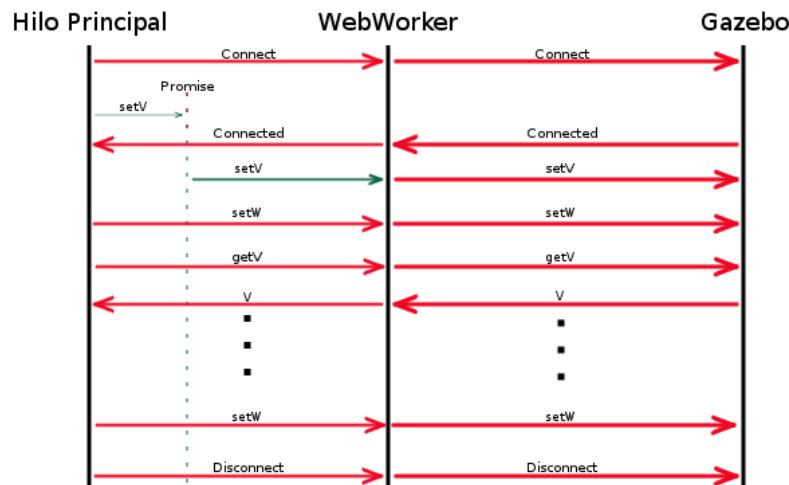


Figura 4.14: Mensajes API.Motors

API.Motors permite conectar con la interfaz Motors de JdeRobot. El contenido de **config** es la siguiente:

- *server*: dirección y puerto del servidor (dir:address, port:port).
- *epname*: endpoint del servidor, por defecto “Motors”.

El contenido de la variable **data** es el siguiente:

- *v,w,l*: velocidades recibidas del servidor.

y su lista de métodos es:

- *createWork*: Crea el *WebWorker* (se ejecuta cuando se crea el conector).

- *deleteWork*: Elimina el *WebWorker*.
- *connect*: Inicia la conexión con el servidor.
- *disconnect*: Desconecta del servidor.
- *getV*: Pide la velocidad V al servidor.
- *getW*: Pide la velocidad W al servidor.
- *getL*: Pide la velocidad L al servidor.
- *setV*: Envía la velocidad V al servidor.
- *setW*: Envía la velocidad W al servidor.
- *setL*: Envía la velocidad L al servidor.
- *setAll*: Envía todas las velocidades al servidor.

4.4.2. Núcleo

Cuando se inicia el cliente *KobukiViewerJS*, éste inicia un *streaming* de los sensores y se crean el control y la representación 3D. El tratamiento de las imágenes recibidas es el mismo que en *CameraViewJS*. Para el control se ha creado un pequeño componente que permite dibujar el Joystick en un *canvas* (figura 4.15(a)) y que permite dar comportamiento a las interacciones del ratón sobre éste, permitiendo enviar velocidades los motores.

```
control = new GUI.Control ({id:self.controlid});

control.lastW=0;
control.lastV=0;

//comportamiento cuando se mueve el Joystick
control.onPointerM = function (event){
    control.onPointerMDefault(event);
    var distSend = 2;
    var pos = control.position;

    if (calcDist(pos.x,control.lastW)>=distSend){
        motors.setW(pos.x/3);
        control.lastW = pos.x;
    }
}
```

```

    }

    if (calcDist(pos.z, control.lastV)>=distSend){
        motors.setV(pos.z);
        control.lastV = pos.z;
    }
};

control.initControl();

```

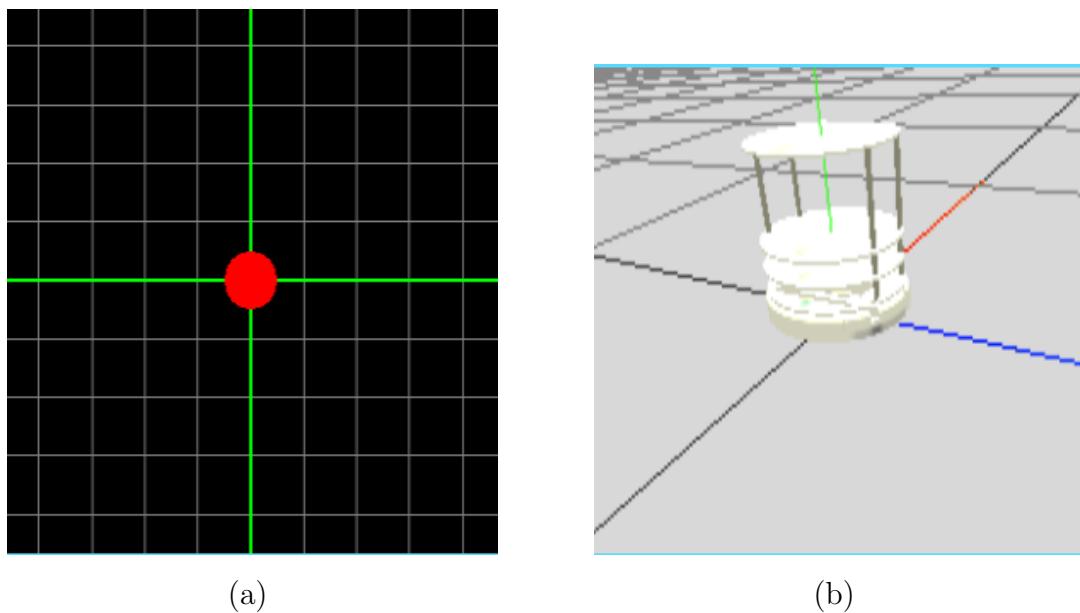


Figura 4.15: Control (a) y Modelo 3D (b)

El botón `stopBot`, coloca el Joystick en el centro y envía velocidades 0 a los motores. Para el modelo 3D (figura 4.15(b)) se ha creado otro pequeño componente que permite cargar desde modelos Collada tanto los robots Pioneer y Kobuki como un Drone. Se ha hecho porque los modelos de los robots venían por partes y así se facilita su uso, además de porque la carga de cada parte es asíncrona y se necesitaba sincronizarlo todo para poder ejecutar una función al final de la carga del modelo. En esta escena de pinta el robot en la posición que indican los sensores.

```

this.loadPioneer = function (scale, onLoad){
    this.minYPos = 0.11;
    this.robot = new THREE.Group();
    var loaded = 2;

```

```
this.manager.onLoad = onLoad;
var chassisLoader = new THREE.ColladaLoader(self.manager);
chassisLoader.options.convertUpAxis = true;
chassisLoader.load(
    'js/libs/robotloaders/pioneer/chassis.dae',
    function ( collada ) {

        var obj = collada.scene;

        obj.scale.x =obj.scale.y = obj.scale.z = scale;
        obj.position.y=(self.minYPos+0.05)*scale;
        obj.updateMatrix();

        self.robot.add(obj);
        loaded--;
        if (loaded==0){
            onLoad();
        }
    });

var wheelLoader = new THREE.ColladaLoader(self.manager);
wheelLoader.options.convertUpAxis = true;
wheelLoader.load(
    'js/libs/robotloaders/pioneer/wheel.dae',
    function ( collada ) {

        var obj = collada.scene;
        var obj2 = obj.clone();

        obj.scale.x =obj.scale.y = obj.scale.z = scale;
        obj.position.z=-0.17*scale;
        obj.position.x=0.1*scale;
        obj.position.y=self.minYPos*scale;
        obj.updateMatrix();

        obj2.scale.x =obj2.scale.y = obj2.scale.z = scale;
        obj2.position.z=0.17*scale;
        obj2.position.x=0.1*scale;
```

```

    obj2.position.y=self.minYPos*scale;
    obj2.rotation.y=Math.PI;
    obj2.updateMatrix();

    self.robot.add(obj);
    self.robot.add(obj2);
    loaded--;
    if (loaded==0){
        onLoad();
    }
});

```

Una vez creado todo el modelo se inician los dos sensores que van a interactuar con éste, como son el Pose3D, que cada vez que se reciben datos cambia la posición y la orientación del robot en el modelo, y el láser que además de representar en un `canvas` sus datos en 2D, se añaden al modelo en 3D (figura 4.16).

```

loader.loadKobuki(1, function () {
    model.robot=loader.robot;
    model.scene.add( model.robot );

    pose3d = new API.Pose3D({server:self.pose3dserv,
epname:self.pose3depname});
    pose3d.onmessage= function (event){
        pose3d.onmessageDefault(event);
        model.robot.position.set(pose3d.data.x/1000,
pose3d.data.z/1000,-pose3d.data.y/1000);
        model.robot.rotation.y=(pose3d.data.yaw);
        model.robot.updateMatrix();
        model.renderer.render(model.scene,model.camera);
    };

    pose3d.timeoutE=timeout;

    pose3d.connect();
    pose3d.startStreaming();
}

```

```
        laser= new API.Laser({server:self.laserserv,epname:
self.laserepname,canv2dWidth:lasercanv.width,scale3d:0.001,
convertUpAxis:true});

        laser.onmessage= function (event){
            laser.onmessageDefault(event);
            //2D
            var dist = laser.data.canv2dData;
            var ctx = lasercanv.getContext("2d");
            ctx.beginPath();
            ctx.clearRect(0,0,lasercanv.width,lasercanv.
height);
            ctx.fillRect(0,0,lasercanv.width,lasercanv.
height);
            ctx.strokeStyle="white";
            ctx.moveTo(dist[0], dist[1]);
            for (var i = 2;i<dist.length; i = i+2 ){
                ctx.lineTo(dist[i], dist[i+1]);
            }
            ctx.moveTo(lasercanv.width/2, lasercanv.height);
            ctx.lineTo(lasercanv.width/2, lasercanv.height
-10);
            ctx.stroke();

            //3D
            var geometry = new THREE.BufferGeometry();
            geometry.addAttribute( 'position' , new THREE.
BufferAttribute( new Float32Array(laser.data.array3dData), 3 )
);
            var material = new THREE.MeshBasicMaterial( {
color: 0x00ff00 } );
            material.transparent = true;
            material.opacity=0.5;
            material.side = THREE.DoubleSide;
            var las = new THREE.Mesh( geometry, material );
            if (model.laser){
                model.robot.remove(model.laser);
            };
            model.robot.add(las);
```

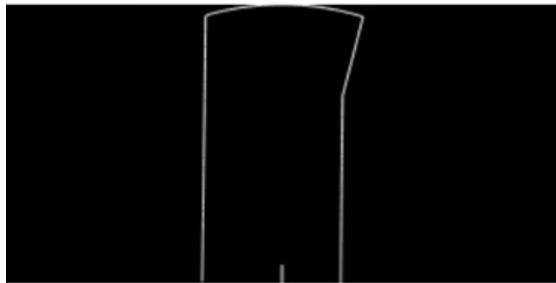
```

        model.laser = las;

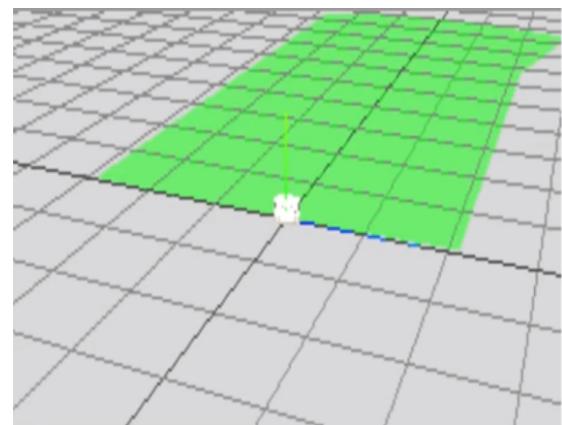
    };
    laser.connect();
    laser.startStreaming();

    modelAnimation();
}) ;

```



(a)



(b)

Figura 4.16: Láser en 2D (a) y 3D (b)

4.4.3. Interfaz gráfico

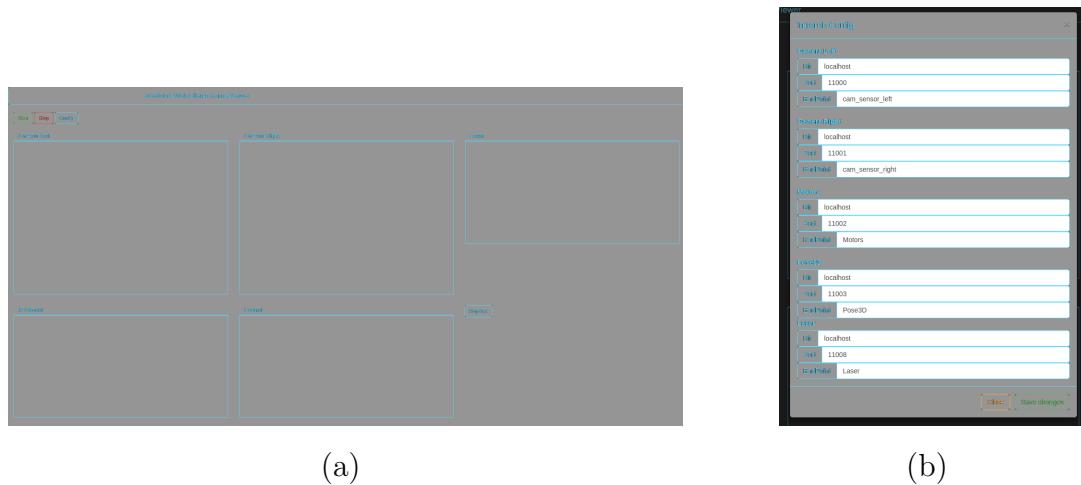
los elementos `header` y `modal` son iguales a los de `CameraViewJS` con alguna pequeña modificación, por lo que nos vamos a centrar en el `body`.

El `body` consta de 5 `canvas` y un botón ordenados mediante el sistema *Grid* de Bootstrap. Los dos primeros de arriba para las imágenes de las dos cámaras, el tercero para el láser en 2D, el cuarto para el modelo 3D y el quinto para el control (figura 4.18).

```

<div id="body" class="container">
  <div class="row">
    <div id="buttons" class="col-xs-12 col-sm-12 col-md-12 col-
      lg-12">
      <p>

```



(a)

(b)

Figura 4.17: Interfaz (a) y Modal (b)

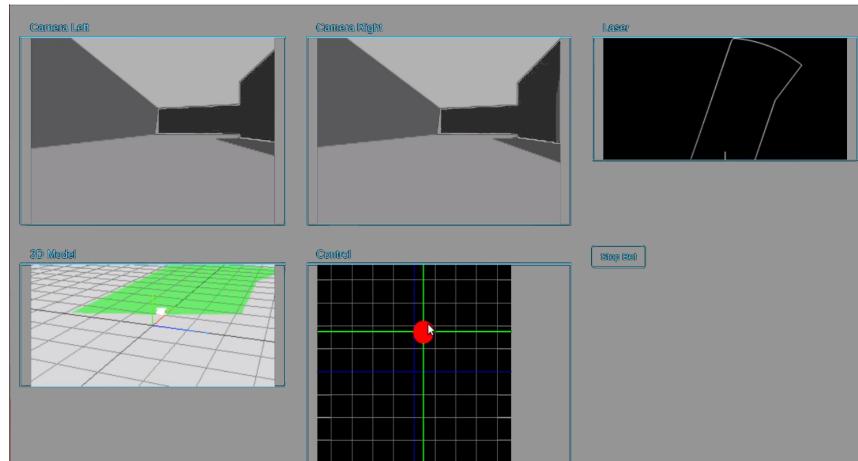


Figura 4.18: Body de KobukiViewerJS

```

        <button id="start" type="button" class="btn btn-md btn-success">Start</button>
        <button id="stop" type="button" class="btn btn-md btn-danger">Stop</button>
        <button id="config" type="button" class="btn btn-info" data-toggle="modal" data-target="#configure">Config</button>
    </p>
</div>
<div class="row">
    <div class="col-xs-12 col-sm-8 col-md-6 col-lg-4">
        <div class="border-carbon panel panel-info">
            <div class="letrero panel-heading hidden-sm hidden-xs">

```

```
">

    <span class="panel-title">Camera Left</span>
  </div>
  <div class="panel-body padding0 border-blue container-fluid">
    <canvas id="camView" class="col-xs-12 col-sm-12 col-md-12 col-lg-12 cam">Your browser does not support the HTML5 canvas tag.</canvas>
  </div>
</div>
<div class="col-xs-12 col-sm-8 col-md-6 col-lg-4">
  <div class="border-carbon panel panel-info">
    <div class="letrero panel-heading hidden-sm hidden-xs">
      <span class="panel-title">Camera Right</span>
    </div>
    <div class="panel-body padding0 border-blue container-fluid">
      <canvas id="camView2" class="col-xs-12 col-sm-12 col-md-12 col-lg-12 cam">Your browser does not support the HTML5 canvas tag.</canvas>
    </div>
  </div>
</div>
<div class="col-xs-12 col-sm-8 col-md-6 col-lg-4">
  <div class="border-carbon panel panel-info">
    <div class="letrero panel-heading hidden-sm hidden-xs">
      <span class="panel-title">Laser</span>
    </div>
    <div class="panel-body padding0 border-blue container-fluid">
      <canvas id="laser" class="col-xs-12 col-sm-12 col-md-12 col-lg-12">Your browser does not support the HTML5 canvas tag.</canvas>
    </div>
  </div>
</div>
```

```
</div>
</div>
<div class="row">
    <div class="col-xs-12 col-sm-8 col-md-6 col-lg-4">
        <div class="border-carbon panel panel-info">
            <div class="letrero panel-heading hidden-sm hidden-xs">
                <span class="panel-title">3D Model</span>
            </div>
            <div class="panel-body padding0 border-blue container-fluid">
                <canvas id="model" class="col-xs-12 col-sm-12 col-md-12 col-lg-12">Your browser does not support the HTML5 canvas tag.</canvas>
            </div>
        </div>
    </div>
    <div class="col-xs-12 col-sm-8 col-md-6 col-lg-4">
        <div class="border-carbon panel panel-info">
            <div class="letrero panel-heading hidden-sm hidden-xs">
                <span class="panel-title">Control</span>
            </div>
            <div class="panel-body padding0 border-blue container-fluid">
                <canvas id="control" class="col-xs-12 col-sm-12 col-md-12 col-lg-12">Your browser does not support the HTML5 canvas tag.</canvas>
            </div>
        </div>
    </div>
    <div class="col-xs-6 col-sm-4 col-md-2 col-lg-2">
        <button id="stopR" type="button" class="btn btn-info" >
            Stop Bot</button>
    </div>
</div>
</div>
```

Por último, cabe destacar que la escena 3D del robot usa exactamente el mismo control de cámara que el usado en `RgbdViewerJS` (figura 4.19), de modo que la cámara de observación de la escena 3D se puede mover y girar a voluntad.

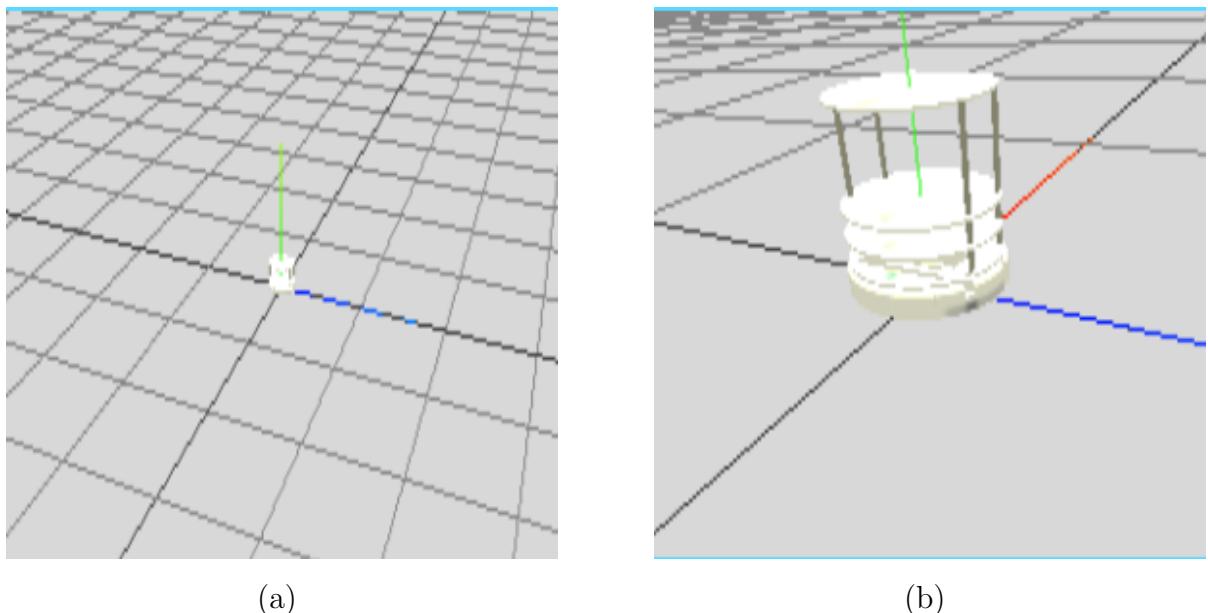


Figura 4.19: sin (a) y con (b) Zoom

4.5. UavViewerJS

Este cliente se conecta con un drone para poderlo teleoperar desde la web. Para ello usa un `API.Camera`, un `API.Pose3D`, un `API.CmdVel` y un `API.Extra` (figura 4.20).

La variable `config` para crearlo contiene:

- `cam1serv`: dirección y puerto del servidor de la cámara(`dir:address, port:port`).
- `cam1epname`: *endpoint* del servidor de la cámara.
- `cam1id`: id del canvas que muestra la imagen de la cámara.
- `cmdvelserv`: dirección y puerto del servidor de la velocidad (`dir:address, port:port`).
- `cmdvelepname`: *endpoint* del servidor de la velocidad.

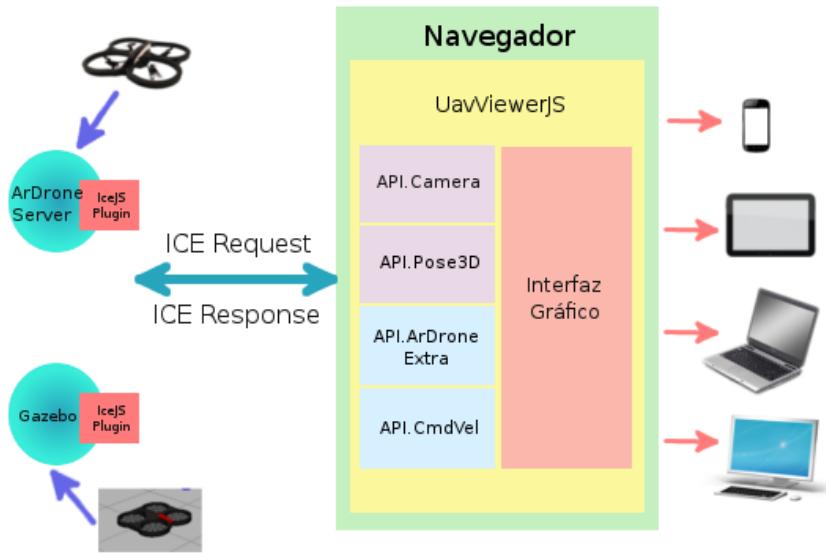


Figura 4.20: Arquitectura de UavViewerJS

- *control1id, control2id*: id de los canvas de los controles.
- *modelid*: id del canvas que muestra una representación en 3D del drone.
- *pose3dserv*: dirección y puerto del servidor de Pose3D (dir:address, port:port).
- *pose3dename*: endpoint del servidor de Pose3D.
- *takeoffbtnid*: id del botón que despega el drone.
- *stopbtnid*: id del botón que detiene el drone.
- *landbtnid*: id del botón que aterriza el drone.
- *resetbtnid*: id del botón que resetea el drone.
- *extraserv*: endpoint del servidor de ArDroneExtra, funciones propias del drone.
- *extraepname*: dirección y puerto del servidor del interfaz Extra(dir:address, port:port).
- *attitudeid, headingid, altimeterid, turn-coordinatorid*: ids de los indicadores de vuelo.

4.5.1. Conectores

Tanto `API.Camera` como `API.Pose3D` ya están descritos en los clientes anteriores, por lo que vamos a describir los nuevos conectores.

API.ArDroneExtra permite conectar con la interfaz ArDroneExtra de JdeRobot. El contenido de `config` es la siguiente:

- *server*: dirección y puerto del servidor (dir:address, port:port).
- *epname*: endpoint del servidor, por defecto “Extra”.

y su lista de métodos es:

- *createWork*: Crea el *WebWorker* (se ejecuta cuando se crea el conector).
- *deleteWork*: Elimina el *WebWorker*.
- *connect*: Inicia la conexión con el servidor.
- *disconnect*: Desconecta del servidor.
- *toggleCam*: Cambia de cámara del drone.
- *land*: Aterriza el drone.
- *takeoff*: Despegue el drone.

`API.CmdVel`, como todo conector con actuadores, básicamente funciona igual que `API.Motors`, pero con las siguientes diferencias: permite conectar con la interfaz CmdVel de JdeRobot. El contenido de `config` es la siguiente:

- *server*: dirección y puerto del servidor (dir:address, port:port).
- *epname*: endpoint del servidor, por defecto “CmdVel”.

y su lista de métodos es:

- *createWork*: Crea el *WebWorker* (se ejecuta cuando se crea el conector).
- *deleteWork*: Elimina el *WebWorker*.
- *connect*: Inicia la conexión con el servidor.

- *disconnect*: Desconecta del servidor.
- *setCmdVel*: Envía las velocidades al servidor (*linearX*, *linearY*, *linearZ*, *angularX*, *angularY*, *angularZ*).

4.5.2. Núcleo

Cuando se inicia el cliente, éste inicia un *streaming* de los sensores y se crean dos controles y la representación 3D. El tratamiento de las imágenes recibidas es el mismo que en CameraViewJS. Los controles son los mismos que los creados para KobukiViewerJS, pero cada uno controla dos velocidades.

```

control1 = new GUI.Control ({id:self.control1id});
control2 = new GUI.Control ({id:self.control2id});

control1.onPointerM = function (event){
    control1.onPointerMDefault(event);
    var distSend = 1;
    var pos = control1.position;
    var send = false;
    if (calcDist(pos.x,cmdSend.linearY)>=distSend){
        cmdSend.linearY = pos.x/2;
        send = true;
    }
    if (calcDist(pos.z,cmdSend.linearX)>=distSend){
        cmdSend.linearX = pos.z/2;
        send = true;
    }
    if (send){
        cmdvel.setCmdVel(cmdSend);
    }
};

control2.onPointerM = function (event){
    control2.onPointerMDefault(event);
    var distSend = 2;
    var pos = control2.position;
    var send = false;
    if (calcDist(pos.x/10,cmdSend.angularZ)>=distSend){

```

```

        cmdSend.angularZ = pos.x/10;
        send = true;
    }

    if (calcDist(pos.z/2,cmdSend.linearZ)>=distSend){
        cmdSend.linearZ = pos.z/2;
        send = true;
    }

    if (send){
        cmdvel.setCmdVel(cmdSend);
    }
}

control1.initControl();
control2.initControl();

```

El botón `stop drone`, coloca los Joystick en el centro y envía velocidades 0 a los motores, el resto de botones ejecutan las funciones homónimas de `API.ArDroneExtra` (figura 4.21).

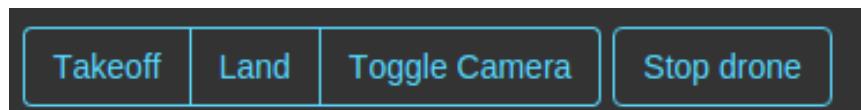


Figura 4.21: Botones de funciones del drone

Para el modelo 3D se ha usado el mismo componente que en `KobukiViewerJS`, una vez creado el modelo se inicia el `Pose3D` y cada vez que se recibe información de éste varía la posición del modelo (figura 4.22(a)) y de los indicadores de vuelo¹ (figura 4.22(b)).



Figura 4.22: Modelo 3D (a) e indicadores de vuelo (b)

¹<https://github.com/sebmatton/jQuery-Flight-Indicators>

```
loader.loadQuadrotor(0.05, function () {
    model.robot=loader.robot;
    model.scene.add( model.robot );

    pose3d = new API.Pose3D({server:self.pose3dserv,
epname:self.pose3dename});
    pose3d.onmessage= function (event){
        pose3d.onmessageDefault(event);
        model.robot.position.set(pose3d.data.x,pose3d.
data.z,-pose3d.data.y);
        model.robot.rotation.set(pose3d.data.pitch,
pose3d.data.yaw,pose3d.data.roll);
        model.robot.updateMatrix();
        model.renderer.render(model.scene,model.camera);
        // Attitude update
        attitude.setRoll(-pose3d.data.roll * toDegrees);
        attitude.setPitch(-pose3d.data.pitch * toDegrees
);

        // Altimeter update
        altimeter.setAltitude(pose3d.data.z*100);

        // TC update
        turn_coordinator.setTurn(-pose3d.data.roll *
toDegrees);

        // Heading update
        heading.setHeading(pose3d.data.yaw * toDegrees)
;
    };

    pose3d.timeoutE=timeout;

    pose3d.connect();
    pose3d.startStreaming();

    modelAnimation();
```

```
}
```

4.5.3. Interfaz gráfico

Cómo ya se comentó en los anteriores clientes, los elementos `header` y `modal` son casi iguales con alguna pequeña modificación, por lo que nos vamos a centrar ahora en el `body`.

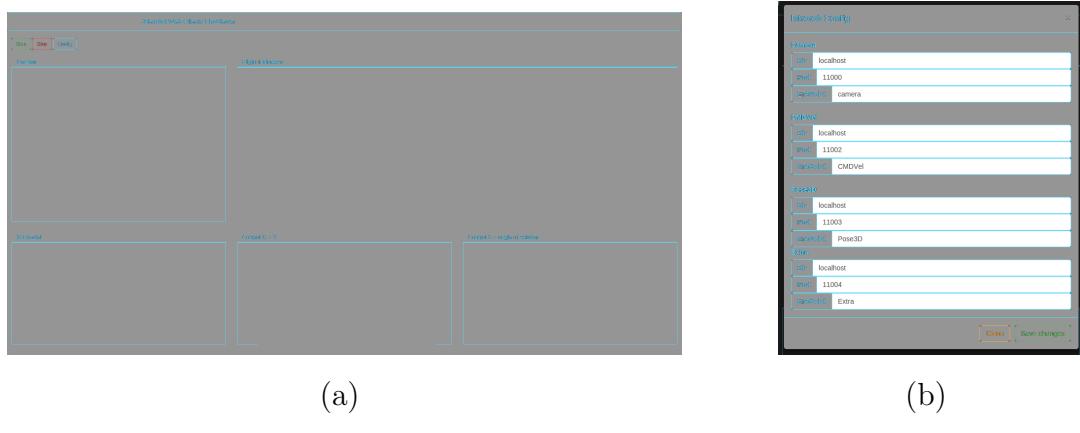


Figura 4.23: Interfaz (a) y Modal (b)

El `body` consta de 3 filas, la primera contiene el `canvas` de la cámara y los indicadores de vuelo, la segunda los `canvas` del modelo 3D y los controles y la tercera contiene los botones (figura 4.24).

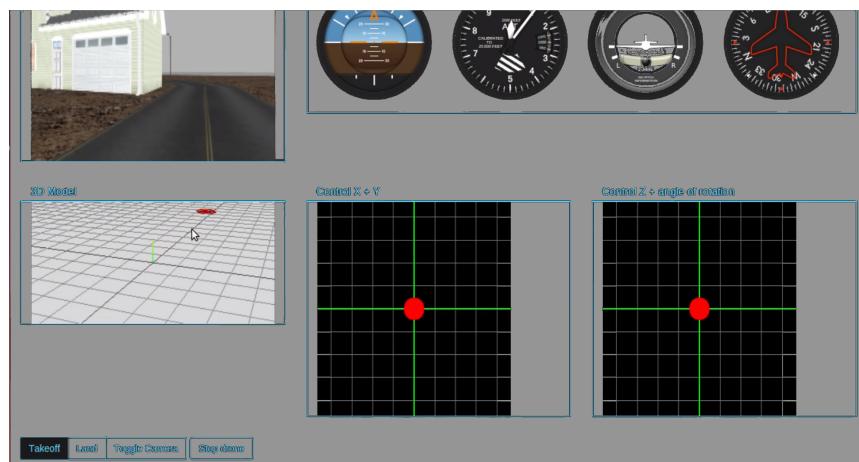


Figura 4.24: Body de UavViewerJS

```
<div id="body" class="container">
```

```
<div class="row">
    <div id="buttons" class="col-xs-12 col-sm-12 col-md-12 col-lg-12"><p>
        <button id="start" type="button" class="btn btn-md btn-success">Start</button>
        <button id="stop" type="button" class="btn btn-md btn-danger">Stop</button>
        <button id="config" type="button" class="btn btn-info" data-toggle="modal" data-target="#configure">Config</button>
    </p>
    </div>
</div>
<div class="row">
    <div class="col-xs-12 col-sm-8 col-md-6 col-lg-4">
        <div class="border-carbon panel panel-info">
            <div class="letrero panel-heading hidden-sm hidden-xs">
                <span class="panel-title">Camera</span>
            </div>
            <div class="panel-body padding0 border-blue container-fluid">
                <canvas id="camView" class="col-xs-12 col-sm-12 col-md-12 col-lg-12 cam">Your browser does not support the HTML5 canvas tag.</canvas>
            </div>
        </div>
    </div>
    <div class="col-xs-12 col-sm-12 col-md-6 col-lg-8">
        <div class="border-carbon panel panel-info">
            <div class="letrero panel-heading hidden-sm hidden-xs">
                <span class="panel-title">Flight Indicators</span>
            </div>
            <div class="panel-body padding0 border-blue container-fluid">
                <span id="attitude"></span>
                <span id="altimeter"></span>
                <span id="turn_coordinator"></span>
            </div>
        </div>
    </div>
</div>
```

```
        <span id="heading"></span>
    </div>
</div>
</div>

<div class="row">
    <div class="col-xs-12 col-sm-8 col-md-6 col-lg-4">
        <div class="border-carbon panel panel-info">
            <div class="letrero panel-heading hidden-sm hidden-xs">
                <span class="panel-title">3D Model</span>
            </div>
            <div class="panel-body padding0 border-blue container-fluid">
                <canvas id="model" class="col-xs-12 col-sm-12 col-md-12 col-lg-12">Your browser does not support the HTML5 canvas tag.</canvas>
            </div>
        </div>
    </div>
    <div class="col-xs-12 col-sm-8 col-md-6 col-lg-4">
        <div class="border-carbon panel panel-info">
            <div class="letrero panel-heading hidden-sm hidden-xs">
                <span class="panel-title">Control X + Y</span>
            </div>
            <div class="panel-body padding0 border-blue container-fluid">
                <canvas id="control1" class="col-xs-12 col-sm-12 col-md-12 col-lg-12">Your browser does not support the HTML5 canvas tag.</canvas>
            </div>
        </div>
    </div>
    <div class="col-xs-12 col-sm-8 col-md-6 col-lg-4">
        <div class="border-carbon panel panel-info">
            <div class="letrero panel-heading hidden-sm hidden-xs">
```

```
">

    <span class="panel-title">Control Z + angle of
rotation</span>
    </div>
    <div class="panel-body padding0 border-blue container-
fluid">
        <canvas id="control2" class="col-xs-12 col-sm-12
col-md-12 col-lg-12">Your browser does not support the HTML5
canvas tag.</canvas>
    </div>
    </div>
</div>
<div class="row">
    <div id="buttons2" class="btn-toolbar col-xs-12 col-sm-12
col-md-12 col-lg-12" role="toolbar" aria-label="buttons2">
        <div class="btn-group" role="group" aria-label="bextra">
            <button id="takeoff" type="button" class="btn btn-info">
Takeoff</button>
            <button id="land" type="button" class="btn btn-info">
Land</button>
            <button id="toggle" type="button" class="btn btn-info">
Toggle Camera</button>
        </div>
        <div class="btn-group" role="group" aria-label="stp">
            <button id="stopb" type="button" class="btn btn-info">
Stop drone</button>
        </div>
    </div>
</div>
</div>
```

4.6. IntrorobKobukiJS

Este cliente es exactamente el mismo que **KobukiViewerJS** pero añadiendo la infraestructura para el comportamiento autónomo (figura 4.25), por lo que la variable **config** que recibe el constructor es igual. Se le han añadido dos métodos nuevos aparte de los comunes a todos los clientes:

- *startMyAlgorithm*: inicia el comportamiento autónomo.
- *stopMyAlgorithm*: detiene el comportamiento autónomo.

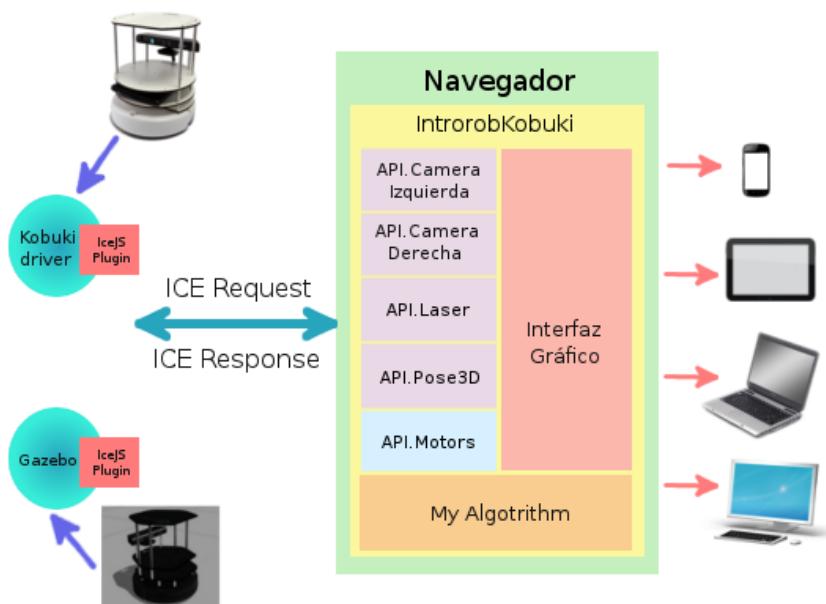


Figura 4.25: Arquitectura de IntrorobKobukiJS

Utiliza los mismos *conectores* que **KobukiViewerJS**.

4.6.1. Núcleo

La única diferencia con **KobukiViewerJS** es la opción de agregar comportamiento autónomo. Cuando se inicia el comportamiento autónomo se crea un *WebWorker* donde se ha programado dicho comportamiento. El hilo principal cada 100ms le envía los datos recibidos por los sensores al *WebWorker*, y éste a su vez, después de ejecutar el comportamiento autónomo, devuelve una orden de velocidad para los motores (en caso de ser necesario incluso puede detener e iniciar el envío de datos del hilo principal):

```
this.startMyAlgorithm = function (){  
    //document.getElementById(self.controlid).SetActive(false);  
    control.removeListeners();  
    document.getElementById(self.stopbtnid).disabled = true;  
    var f = function(){  
        if (laser.data && cameraright.data && pose3d.data &&  
            cameraleft.data ){  
            var msg ={pose3d:pose3d.data,  
                      laser:{distanceData:laser.data.distanceData,  
                             numLasers:laser.data.numLaser},  
                      camr:{pixelData:cameraright.data.pixelData,  
                             height:cameraright.data.height, width:cameraright.data.width},  
                      caml:{pixelData:cameraleft.data.pixelData,height  
                             :cameraleft.data.height, width:cameraleft.data.width}};  
            worker.postMessage(msg);  
        }  
    };  
    worker = new Worker(workerFile);  
    worker.onmessage = function (m){  
        if (lastV!=m.data.v || lastW!=m.data.w){  
            lastV = m.data.v;  
            lastW = m.data.w;  
            motors.setV(lastV);  
            motors.setW(lastW);  
        }  
        var d = m.data.interval;  
        switch (d){  
            case 1:  
                interval = setInterval(f,100);  
                break;  
            case 2:  
                clearInterval(interval);  
                break;  
            default:  
  
        }  
    };
```

```
interval = setInterval(f,100);
```

Para detener el comportamiento, se deja de enviar datos al *WebWorker* (se detiene el *interval*) y se elimina dicho *worker*. El contenido de fichero **myalgorithm_worker.js**, que contiene el *script* que va a interpretar el *WebWorker* es el siguiente:

```
/* Data Received
 * - laser:
 *     + distanceData : Array of distances
 *     + numLaser: numer of lasers
 * - pose3d:
 *     + x,y,z: coords
 *     + q1,q2,q3,q4 : quaternion
 *     + yaw, pitch, roll: orientation
 * - caml, camr:
 *     + data: Array of Image data
 *     + width, height: width y height of the image, in pixels
 *
 ****
 * Data Response
 * - v,w: velocities
 * - interval: if the action takes more than time reception sensor
   (100ms). Possible values:
 * + 0 : do nothing
 * + 1 : start interval in main thread
 * + 2 : stop interval in main thread
 */
onmessage = function(e) {

  var laser = e.data.laser;
  var pose3d = e.data.pose3d;
  var caml = e.data.caml;
  var camr = e.data.camr;

  var v = 0;
  var w = 0;
```

```

    postMessage({v:v,w:w,interval:0});
}

```

Como se puede ver, tiene un primer comentario explicando los datos recibidos y los que se pueden enviar, además de la función donde se tiene que programar en JavaScript el comportamiento autónomo, con un ejemplo de respuesta.

4.6.2. Interfaz gráfico

La única variación a la interfaz de KobukiViewerJS es que se le ha agregado un botón más a la botonera para iniciar y detener el comportamiento autónomo (figura 4.26), que se traduce en eliminar o crear el *WebWorker* subyacente que incluye la lógica del comportamiento autónomo y el *interval* de envío de datos. .

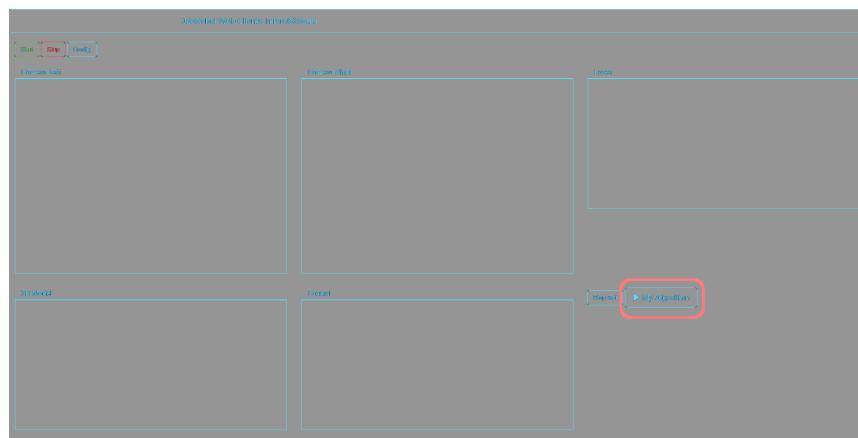


Figura 4.26: Interfaz de IntrorobKobukiJS

4.7. IntrorobUavJS

Es el equivalente a IntrorobKobukiJS pero para drones, por lo que la base es UavViewerJS (figura 4.27).

Utiliza los mismos *conectores* que UavViewerJS.

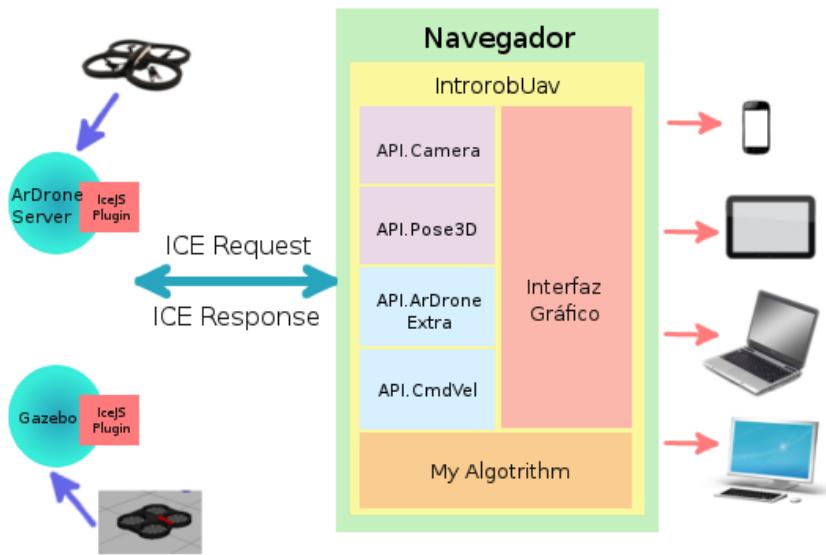


Figura 4.27: Arquitectura de IntrorobUavJS

4.7.1. Núcleo

Es este caso, el mensaje de respuesta del *WebWorker* es más complejo porque se pueden enviar más órdenes que al robot kobuki, por lo tanto el fichero **myalgorithm_worker.js** quedaría así:

```
/* Data Received
 * - pose3d:
 *   + x,y,z: coords
 *   + q1,q2,q3,q4 : quaternion
 *   + yaw, pitch, roll: orientation
 * - cam1:
 *   + data: Array of Image data
 *   + width, height: width y height of the image, in pixels
 *
 ****
 * Data Response
 * - com: command to send to drone. Possible values:
 *   + sendVel : send Velocities
 *   + takeoff : takeoff drone
 *   + land : land drone
 *   + toggleCam : change camera
 * - linearX,linearY,linearZ,angularZ: velocities
```

```

* - interval: if the action takes more than time reception sensor
  (100ms). Possible values:
*   + 0 : do nothing
*   + 1 : start interval in main thread
*   + 2 : stop interval in main thread
*/

```

```

onmessage = function(e) {

  var pose3d = e.data.pose3d;
  var cam1 = e.data.cam1;

  var linearX = 0;
  var linearY = 0;
  var linearZ = 0;
  var angularZ = 0;

  postMessage({com:"sendVel",linearX:linearX,linearY:linearY,
  linearZ:linearZ,angularZ:angularZ,interval:0});
}

```

4.7.2. Interfaz gráfico

La única variación a la interfaz de UavViewerJS es que se le ha agregado un botón mas a la botonera para iniciar y detener el comportamiento autónomo (figura 4.28) .

4.8. JdeRobotWebClients

Además de los seis clientes web descritos se ha creado un `index.html` con una pequeña explicación de la página web, como se puede ver en la figura 4.29. Que integra en sendas pestañas cada uno de ellos.

Cada cliente sigue en un fichero HTML distinto, para permitir separarlos de la web más fácilmente. La única modificación que se les ha hecho es añadir un menú de navegación en el `header` para facilitar el cambio entre clientes.

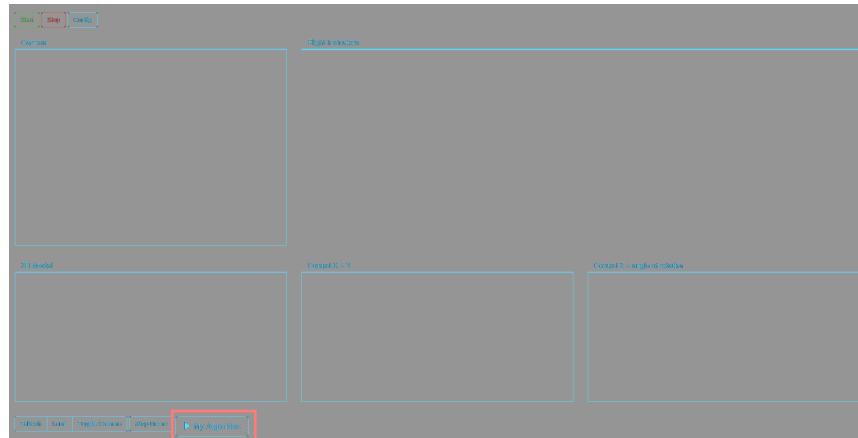


Figura 4.28: Interfaz de IntrorobUavJS



(a)

(b)

Figura 4.29: Escritorio (a) y Móvil (b)

```

<header id="header">
  <nav class="navbar navbar-inverse" role="banner">
    <div class="container">
      <div class="navbar-header">
        <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse"> <span class="sr-only">Toggle navigation</span> <span class="icon-bar"></span> <span class="icon-bar"></span> <span class="icon-bar"></span> </button>
        <a class="navbar-brand" href="index.html"><i class="fa fa-bolt"></i> JdeRobot Web Clients</a></a>
      </div>
    </div>
  </nav>

```

```
<div class="collapse navbar-collapse navbar-right">
  <ul class="nav navbar-nav">
    <li ><a href="index.html">Home</a></li>
    <li class="active"><a href="cameraview.html">Cameraview</a></li>
    <li><a href="rgbdviewer.html">RGBDViewer</a></li>
    <li><a href="kobukiviewer.html">KobukiViewer</a></li>
    <li><a href="uavviewer.html">UavViewer</a></li>
    <li><a href="introrobkobuki.html">IntrorobKobuki</a></li>
    <li><a href="introrobuav.html">IntrorobUav</a></li>
  </ul>
</div>
</div>
<!--/.container-->
</nav>
<!--/nav-->
</header>
```

Capítulo 5

Experimentos

En este capítulo se presentan las pruebas realizadas al sistema y el **hardware** necesario para la reproducción de las mismas. Estos experimentos se dividen en siete pruebas, una por cada cliente con el objetivo de comprobar su correcto funcionamiento y una séptima de rendimiento temporal de recepción de vídeo de las cámaras.

5.1. Experimentos con CameraViewJS

Este experimento consiste en una ejecución típica del cliente CameraViewJS desarrollado. Para ello se ejecuta **cameraserver** en un PC y el cliente tanto en otro PC como en un móvil estando todos en la misma red local.

En la figura 5.1 se puede observar la función que cumplen cada uno de los componentes de las pruebas. Como servidor de imágenes ICE se utilizó un ordenador de la universidad con un procesador Intel(R) Core(TM)2 Quad 2.66GHz con 4GbiB de memoria RAM. En este PC se instaló la distribución Ubuntu de JdeRobot 5.3.1 y se le conectó la webcam logitech. Esta máquina se usa tanto como servidor **cameraserver** como servidor HTTP donde albergar los clientes.

Como cliente 1 se utilizó un portátil Acer Ferrari One con un procesador AMD Athlon(tm) Dual Core L310 y 4 GbiB de memoria RAM. En él se instalaron los navegadores Firefox, Chrome y Opera para realizar la pruebas.

Como cliente 2 se utilizó un *SmartPhone* Elephone P6000 con un procesador MTK6732 de 64Bits a 1.5GHz y 2GbiB de memoria RAM. En él se instalaron los navegadores Firefox, Chrome y Opera para realizar la pruebas.



Figura 5.1: Equipos usados en este experimento y su función.

Como se puede ver en las figuras 5.2, funciona igual tanto en móvil como en PC. La tasa de refresco de las imágenes varía si cliente y servidor se ejecutan en el mismo PC (8 FPS) o en diferente (4 FPS). El tamaño de imagen idóneo es 320x240 píxeles porque más pequeñas se perdería mucha calidad y mayores producen un desplome de los fotogramas por segundo hasta quedarse en 2-3 FPS en la misma máquina. Además hay que comentar que en algunas de las pruebas se han llegado a conseguir 25 FPS en el mismo PC sin ninguna razón aparente porque no se realizó ningún cambio, lo único que se hizo fue parar tanto cliente y servidor y volverlos a arrancar.



Figura 5.2: Cliente 1 (a) y Cliente 2 (b)

5.2. Experimentos con RgbdViewerJS

Este experimento consiste en una ejecución típica del cliente `RgbdViewerJS` desarrollado. Para ello se ejecuta `Openni1Server` en un PC teniendo conectado un Kinect y el cliente en otro PC estando todos en la misma red local.

En la figura 5.3 se puede observar la función que cumplen cada uno de los componentes de las pruebas. Como servidor de imágenes RGBD ICE se utilizó un ordenador de la universidad con un procesador Intel(R) Core(TM)2 Quad 2.66GHz con 4GbiB de memoria RAM. En este PC se instaló la distribución Ubuntu de JdeRobot 5.3.1 y se le conectó el Kinect. Esta máquina se usa tanto como servidor `Openni1Server` como servidor HTTP donde albergar los clientes.

Como cliente se utilizó un portátil Acer Ferrari One con un procesador AMD Athlon(tm) Dual Core L310 y 4 GbiB de memoria RAM. En él se instalaron los navegadores Firefox, Chrome y Opera para realizar el experimento.



Figura 5.3: Equipos usados en estas pruebas y su función.

como se puede ver en la figura 5.4, funciona correctamente. En este caso los experimentos han estado algo limitados debido a que para trabajar con el Kinect se necesita mucha capacidad de procesamiento, por lo que rara vez se pasaba de 1 FPS en la recepción de las imágenes. Hay que destacar que se comprobaron los valores mediante el `RgbdViewer` escrito en C++ y este no pasaba de 2 FPS así que asumimos que el Hardware limitaba dicho experimento.

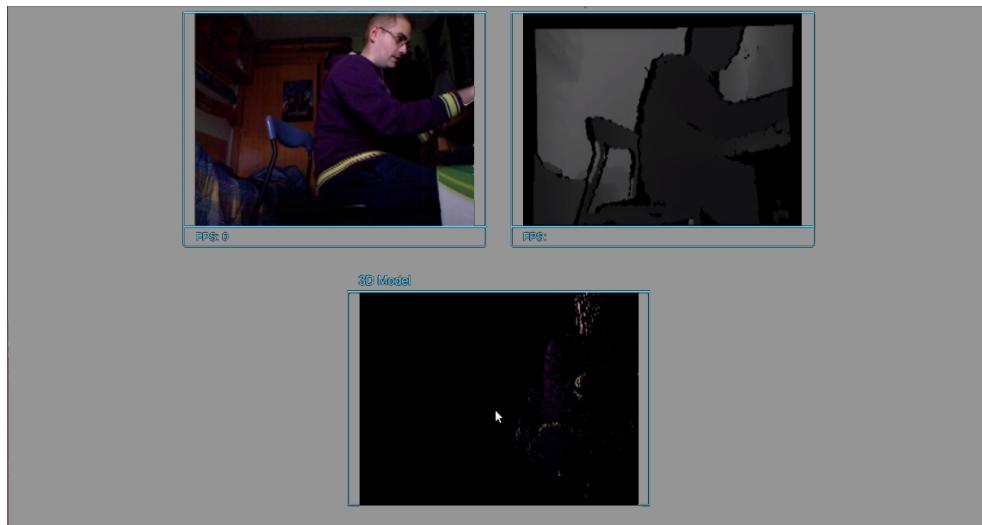


Figura 5.4: Ejecución de cliente

5.3. Experimentos con KobukiViewerJS

Este experimento consiste en una ejecución típica del cliente KobukiViewerJS desarrollado tanto en real como simulado. las pruebas consistieron en teleoperar un robot Kobuki tanto real como simulado estando cliente y servidor en la misma red local.

En la figura 5.5 se puede observar la función que cumplen cada uno de los componentes de las pruebas. Como servidor de robot simulado con gazebo se utilizó un ordenador de la universidad con un procesador Intel(R) Core(TM)2 Quad 2.66GHz con 4GbiB de memoria RAM. En este PC se instaló la distribución Ubuntu de Jderobot 5.3.1. Esta máquina se usa tanto como servidor de kobuki simulado como servidor HTTP donde albergar los clientes.

Como cliente de simulador y servidor Kobuki_driver se utilizó un portátil Acer Ferrari One con un procesador AMD Athlon(tm) Dual Core L310 y 4 GbiB de memoria RAM. En él se instalaron los navegadores Firefox, Chrome y Opera para realizar la pruebas.

Como cliente para Kobuki real se utilizó un *SmartPhone* Galaxy Note II con un procesador Quad-Core a 1.6GHz y 2GbiB de memoria RAM. En él se instalaron los navegadores Firefox, Chrome y Opera para realizar la pruebas.

La prueba con el robot simulado consistió en ejecutar en el PC de la universidad el simulador Gazebo y teleoperarlo desde el cliente KobukiViewerJS ejecutado en el portátil. La prueba con el robot real se realizó en el Laboratorio de robótica de la Universidad, en este caso el servidor era el Portátil que estaba ejecutando Kobuki_driver conectado al robot Kobuki mediante USB y el cliente era el *SmartPhone*. Ambos experimentos tuvieron un resultado satisfactorio como se muestra en las imágenes 5.6. Un pequeño problema que



Figura 5.5: Equipos usados en este experimento y su función.

si se ha detectado en algunos casos es un pequeño retardo a la hora de recibir las imágenes que en algunos casos llegaba incluso a perder la conexión con las cámaras. En estos casos el resto de conexiones han seguido funcionando perfectamente en todo momento.

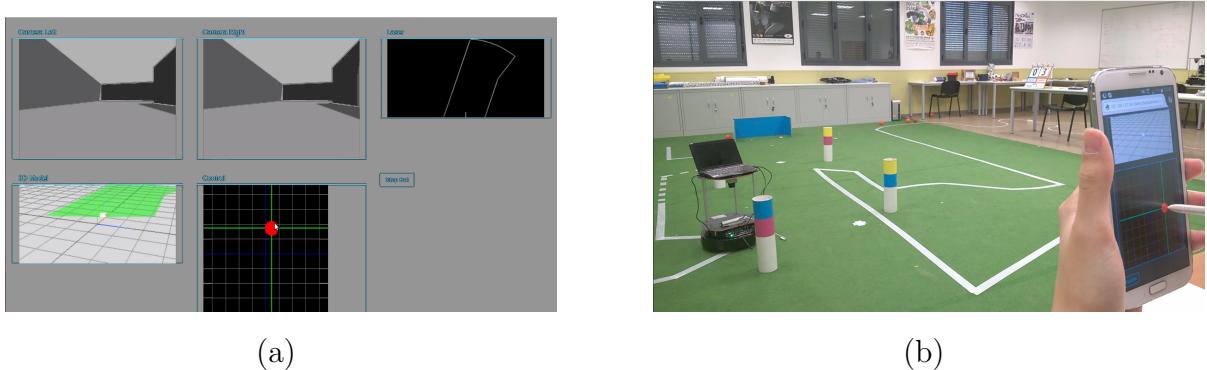


Figura 5.6: Simulado (a) y Real (b)

5.4. Experimentos con UavViewerJS

Este experimento consiste en una ejecución típica del cliente **UavViewerJS** desarrollado tanto en real como simulado. las pruebas consistieron en teleoperar un robot Kobuki tanto real como simulado estando cliente y servidor en la misma red local.

En la figura 5.7 se puede observar la función que cumplen cada uno de los componentes de las pruebas. Como servidor de drone simulado con gazebo se utilizó un ordenador de la universidad con un procesador Intel(R) Core(TM)2 Quad 2.66GHz con 4GbiB de memoria RAM. En este PC se instaló la distribución Ubuntu de Jderobot 5.3.1. Esta máquina se usa tanto como servidor de drone simulado como servidor HTTP donde albergar los clientes.

Como cliente de simulador y servidor `Ardrone_server` se utilizó un portátil Acer Ferrari One con un procesador AMD Athlon(tm) Dual Core L310 y 4 GbiB de memoria RAM. En él se instalaron los navegadores Firefox, Chrome y Opera para realizar la pruebas.

Como cliente para el drone real se utilizó un *SmartPhone* Elephone P6000 con un procesador MTK6732 de 64Bits a 1.5GHz y 2GbiB de memoria RAM. En él se instalaron los navegadores Firefox, Chrome y Opera para realizar la pruebas.



Figura 5.7: Equipos usados en este experimento y su función.

La prueba con el drone simulado consistió en ejecutar en el PC de la universidad el simulador gazebo y teleoperarlo desde el cliente `UavViewerJS` ejecutado en el portátil. La prueba con el drone real (nombrado vídeo del mes de enero en JdeRobot¹) se realizó en el Laboratorio de robótica de la Universidad, en este caso el servidor era el Portátil que estaba ejecutando `Ardrone_server` conectado al drone mediante la red WiFi que genera el propio drone y el cliente era el *SmartPhone*. Ambos experimentos tuvieron un resultado satisfactorio como se muestra en las imágenes 5.8. En el caso del real hay un pequeño retardo en el envío de órdenes al drone

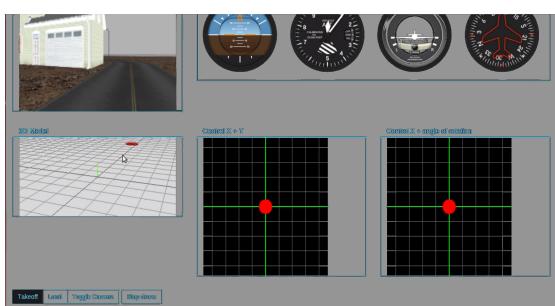
Además en el caso de este cliente se hicieron unas exhibiciones en la Global Robot Expo, feria de robótica de Madrid² (figura 5.9)

5.5. Experimento con IntronrobKobukiJS

Este experimento consiste en la programación de una aplicación concreta en JavaScript para el robot kobuki, hecha dentro de la herramienta `IntronrobKobukiJS` desarrollada. La

¹<https://www.youtube.com/watch?v=rRUxgtEY0ys>

²<http://blog.jderobot.org/estuvimos-en-global-robot-expo/>



(a)



(b)

Figura 5.8: Simulado (a) y Real (b)



(a)



(b)

Figura 5.9: Presentación en Global Robot Expo

prueba consistió en programar un pequeño comportamiento en JavaScript probarlo en el kobuki simulado. Se utilizó el mismo Hardware que KobukiViewerJS

La prueba con el robot simulado consistió en ejecutar en el PC de la universidad el simulador gazebo y ejecutar el comportamiento programado desde el cliente IntronrobKobukiJS. El cliente estaba ejecutándose desde el Portátil. El comportamiento desarrollado era un choca-gira, el robot avanza hasta que percibe mediante el laser que tiene un obstáculo muy cerca, entonces retrocede, gira y vuelve a avanzar.

```
function reanudar (){
    postMessage({v:30,w:0,interval:1});
}

function giro(){
    var t = Math.floor((Math.random() * 10) + 1);
    postMessage({v:0,w:-10,interval:0});
    setTimeout(reanudar,t*100);
}

function atras(){
    postMessage({v:-30,w:0,interval:1});
}
```

```

postMessage({v:-20,w:0,interval:2});
setTimeout(giro,500);
}

onmessage = function(e) {

    var laser = e.data.laser;
    var pose3d = e.data.pose3d;
    var caml = e.data.caml;
    var camr = e.data.camr;

    var v = 0;
    var w = 0;

    if (laser.distanceData[90]<1000){
        atras();
    }else{
        postMessage({v:30,w:0,interval:0});
    };
}

```

El experimento tuvo un resultado satisfactorio como se muestra en las imágenes 5.10. Cada vez que se acercaba a una pared, retrocedía, giraba un tiempo aleatorio y después seguía con su avance.

5.6. Experimentos con IntorobUavJS

Este experimento consiste en la programación de una aplicación concreta en JavaScript para el robot kobuki, hecha dentro de la herramienta **IntorobUavJS** desarrollada. La prueba consistió en programar un pequeño comportamiento en JavaScript probarlo en el drone simulado. Se utilizó el mismo Hardware que en el experimento con **UavViewerJS**

La prueba con el drone simulado consistió en ejecutar en el PC de la universidad el simulador Gazebo y ejecutar el comportamiento programado desde el cliente **IntorobUavJS**. El cliente estaba ejecutándose desde el Portátil. El comportamiento

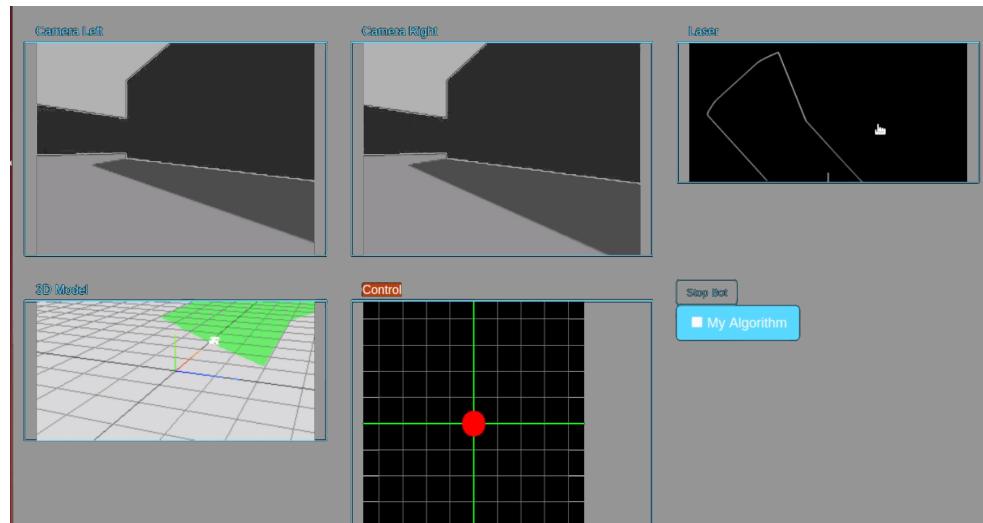


Figura 5.10: Imagen del experimento

consistía en despegar, realizar un cuadrado volando y aterrizar en el mismo sitio aproximadamente.

```

function calcDist (a,b){
    return Math.sqrt(Math.pow(b.x-a.x,2)+Math.pow(b.y-a.y,2));
}

var step =0;
var origen;
var giro = 1;

onmessage = function(e) {

    var pose3d = e.data.pose3d;
    var cam1 = e.data.cam1;

    var linearX = 0;
    var linearY = 0;
    var linearZ = 0;
    var angularZ = 0;
    switch (step){

        case 0:
            postMessage({com:"takeoff",interval:0});
            step++;
            origen = pose3d;
            break;
    }
}

```

```
case 1:  
case 3:  
case 5:  
case 7:  
    console.log(calcDist(origen,pose3d));  
    if (calcDist(origen,pose3d)<2){  
        linearX = 2;  
        postMessage({com:"sendVel",linearX:linearX,linearY:0,  
linearZ:0,angularZ:0,interval:0});  
        //console.log("send: linearX");  
    }else{  
        linearX = 0;  
        postMessage({com:"sendVel",linearX:linearX,linearY:0,  
linearZ:0,angularZ:0,interval:0});  
        console.log("send: linearX = 0");  
        step++;  
    }  
    break;  
case 2:  
    console.log(Math.PI/2);  
    console.log(pose3d.yaw);  
    if (Math.PI/2>pose3d.yaw){  
        angularZ = 0.1;  
        postMessage({com:"sendVel",linearX:0,linearY:0,linearZ  
:0,angularZ:angularZ,interval:0});  
        console.log("send: angularZ = 1");  
    }else{  
        postMessage({com:"sendVel",linearX:0,linearY:0,linearZ  
:0,angularZ:0,interval:0});  
        console.log("send: angularZ = 0");  
        origen = pose3d;  
        giro++;  
        step++;  
    }  
    break;  
case 4:  
    console.log(Math.PI*0.95);  
    console.log(pose3d.yaw);
```

```
    if (0<pose3d.yaw){
        angularZ = 0.1;
        postMessage({com: "sendVel", linearX:0, linearY:0, linearZ
:0, angularZ:angularZ ,interval:0});
        console.log("send: angularZ = 1");
    }else{
        postMessage({com: "sendVel", linearX:0, linearY:0, linearZ
:0, angularZ:0 ,interval:0});
        console.log("send: angularZ = 0");
        origen = pose3d;
        giro++;
        step++;
    }
    break;
case 6:
    console.log(-Math.PI/2);
    console.log(pose3d.yaw);
    if (-Math.PI/2>pose3d.yaw){
        angularZ = 0.1;
        postMessage({com: "sendVel", linearX:0, linearY:0, linearZ
:0, angularZ:angularZ ,interval:0});
        console.log("send: angularZ = 1");
    }else{
        postMessage({com: "sendVel", linearX:0, linearY:0, linearZ
:0, angularZ:0 ,interval:0});
        console.log("send: angularZ = 0");
        origen = pose3d;
        giro++;
        step++;
    }
    break;
case 8:
    postMessage({com: "land", interval:2});
    break;
default:
}
//postMessage({linearX:linearX, linearY:linearY, linearZ:linearZ,
```

```

    angularZ:angularZ,interval:0});
}

```

El experimento tuvo un resultado satisfactorio como se muestra en las imágenes 5.11.

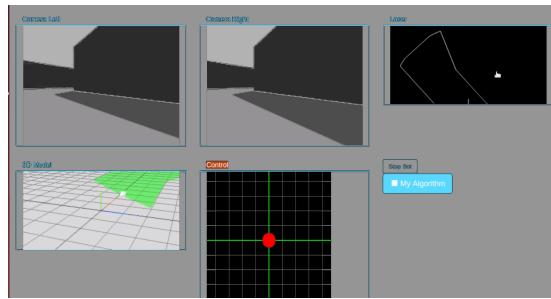


Figura 5.11: Imagen del experimento

5.7. Rendimiento Temporal

Además de verificar el correcto funcionamiento de todas las herramientas desarrolladas, este experimento consiste en una comparación del cliente **CameraViewJS** desarrollado frente al **CameraView** ya existente. Sólo se ha considerado este cliente porque el vídeo es el que tiene una mayor tasa de refresco, y por lo tanto, es el que será más sensible a restricciones de ancho de banda.

Se han definido dos localizaciones con distintas conexiones a Internet y distintos anchos de banda. Las características de dichas conexiones han sido medidas con la herramienta web *Speed Test*³.

- **Universidad (URJC)**: La primera localización es el campus de Fuenlabrada de la Universidad Rey Juan Carlos. La universidad dispone de una conexión con un ancho de banda que supera los 100 Mbps característicos de la tecnología Fast Ethernet, por lo que tomaremos ese valor de 100 Mbps como el ancho de banda disponible para las pruebas.
- **Domicilio (Casa)**: La segunda localización es un domicilio particular con una conexión de fibra óptica. Es una conexión asimétrica con un ancho de banda de 5 Mbps de subida y 50 Mbps de bajada.

³<http://www.speedtest.net/es/>

Cameraserver		Clientes			FPS cameraView	FPS CameraViewJS
Ubicación	Mbps bajada/subida	Tipo	Ubicación	Mbps bajada/subida		
Mismo PC	-	PC	Mismo PC	-	9	8
Red Local	1000/1000	PC	Red Local	65/65	6	4
Red Local	1000/1000	Móvil	Red Local	65/65	-	4
Casa	50/5	PC	URJC	15/15	1	1
Casa	50/5	Móvil	3G	5/1	-	1
Casa	50/5	Móvil	4G	13/4	-	1
URJC	100/100	PC	Casa	50/5	6	4
URJC	100/100	Móvil	3G	5/1	-	2
URJC	100/100	Móvil	4G	13/4	-	4

Cuadro 5.1: Resultados de las pruebas

Los equipos usados para las pruebas y su función dentro del sistema son los mismo usado en los experimentos con `CameraViewJS`. La única excepción es que el portátil ejecuta tanto `CameraViewJS` como `cameraview` de C++. En las pruebas se va a medir los fotogramas por segundo del *streaming* de vídeo.

Se puede comprobar que los clientes funcionan bien en situaciones reales. La principal restricción que tiene es el ancho de banda de subida de `cameraserver` y en general de todos los servidores ya que las imágenes se envían sin comprimir y necesitan mucho ancho. La tabla 5.7 muestra un resumen de los resultados de las pruebas

5.7.1. Todos los elementos dentro de una red local

La primera prueba se realiza en una subred local. Todas las máquinas se conectan entre sí usando un *router Wifi*. En esta subred el servidor se conecta al *router* mediante red cableada Gigabit Ethernet y los clientes mediante *wifi N* con un ancho de banda de 65Mbps. Con esta primera prueba se pretende medir las prestaciones del sistema en un entorno ideal con conexiones de alto ancho de banda y sin latencia apreciable. La figura 5.12 representa un esquema del montaje realizado.

El resultado obtenido en esta prueba es un *streaming* con una tasa de refresco de imagen de alrededor de 6 fotogramas por segundo para `cameraview` y 4 fotogramas por segundo para `CameraViewJS` tanto en el portátil como en el móvil. Esto demuestra que el cliente funciona igual tanto en un móvil como en un PC. Además si los comparamos con los resultados en el mismo PC se nota un descenso lógico de FPS ya que ahora hay más distancia entre cliente y servidor.

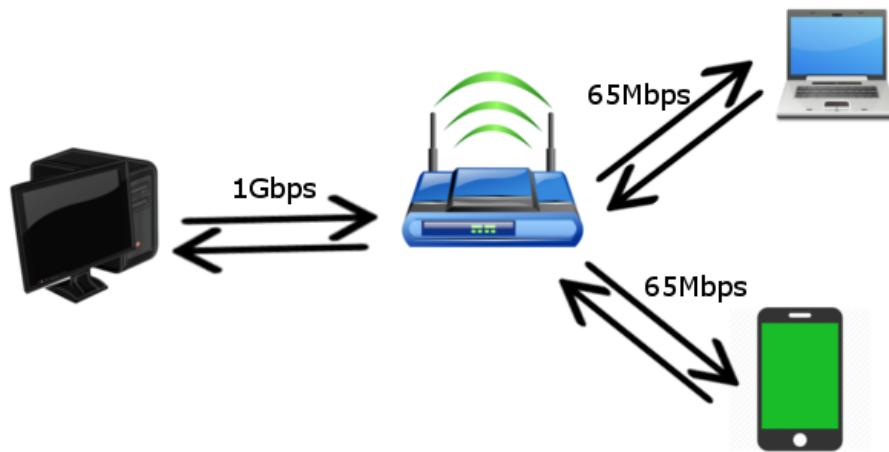


Figura 5.12: Montaje de todos los equipos en la misma red local

5.7.2. Servidor en el domicilio y los clientes en otras redes

La segunda prueba se realiza estando cada máquina en una red diferente. El servidor se encuentra en el domicilio con una conexión a Internet de fibra óptica asimétrica de 50Mbps de bajada y 5Mbps de subida, el cliente 1 se encuentra en la universidad con velocidades de acceso de 15Mbps de subida y 15Mbps de bajada y por último el cliente 2 se conecta mediante 3G (5Mbps/1Mbps) y 4G(13Mbps/4Mbps). La figura 5.13 representa un esquema del montaje realizado.

El resultado obtenido en esta prueba es un *streaming* con una tasa de refresco de imagen de alrededor de 1 fotograma por segundo para `cameraview` y 1 fotograma por segundo para `CameraViewJS` en el portátil y de 1 fps en el móvil tanto en 3G como en 4G. En este caso que todos los clientes funcionen a 1 FPS demuestra que la velocidad de subida de la casa limita la conexión del servidor.

5.7.3. Servidor en la universidad y los clientes en otras redes

La tercera prueba se realiza estando cada máquina en una red diferente. El servidor se encuentra en la universidad con velocidades de acceso de 100Mbps de subida y 100Mbps de bajada, el cliente 1 se encuentra en el domicilio con una conexión a Internet de fibra óptica asimétrica de 50Mbps de bajada y 5Mbps de subida y por último el cliente 2 se conecta mediante 3G (5Mbps/1Mbps) y 4G(13Mbps/4Mbps). La figura 5.14 representa un

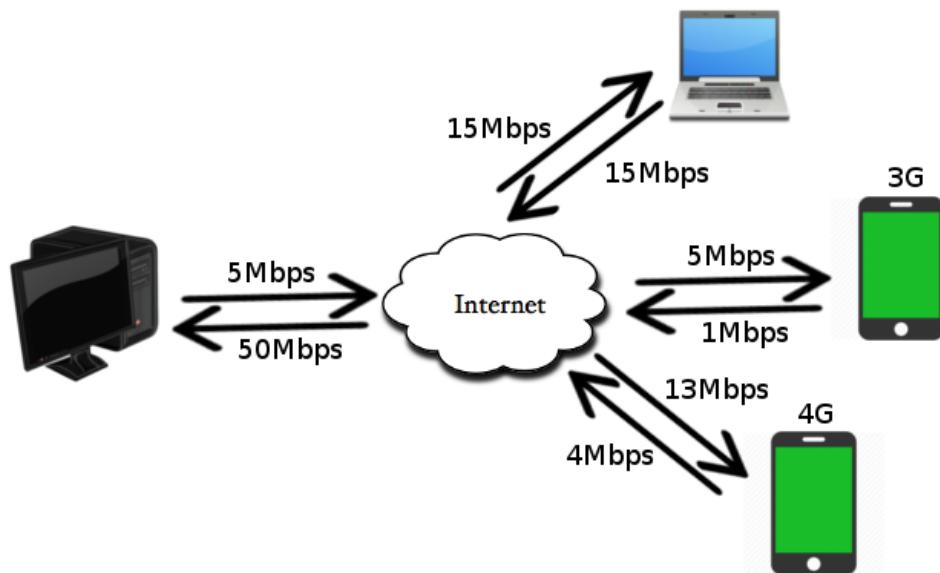


Figura 5.13: Montaje del servidor en casa y los clientes en otras redes

esquema del montaje realizado.

El resultado obtenido en esta prueba es un *streaming* con una tasa de refresco de imagen de alrededor de 6 fotogramas por segundo para `cameraview` y 4 fotogramas por segundo para `CameraViewJS` en el portátil y de 4 fps en el móvil con 4G y 2 con 3G. de esta última prueba se pueden sacar dos conclusiones, la primera es que la conexión entre cliente y servidor por lo menos necesita 10Mbps para conseguir un funcionamiento razonable. La segunda es que el *hardware* usado ha limitado las pruebas porque no se han notado mejoría de FPS desde los 10Mbps hasta los 100Mbps de la red local, por lo que se les puede achacar tanto a los clientes como al servidor.

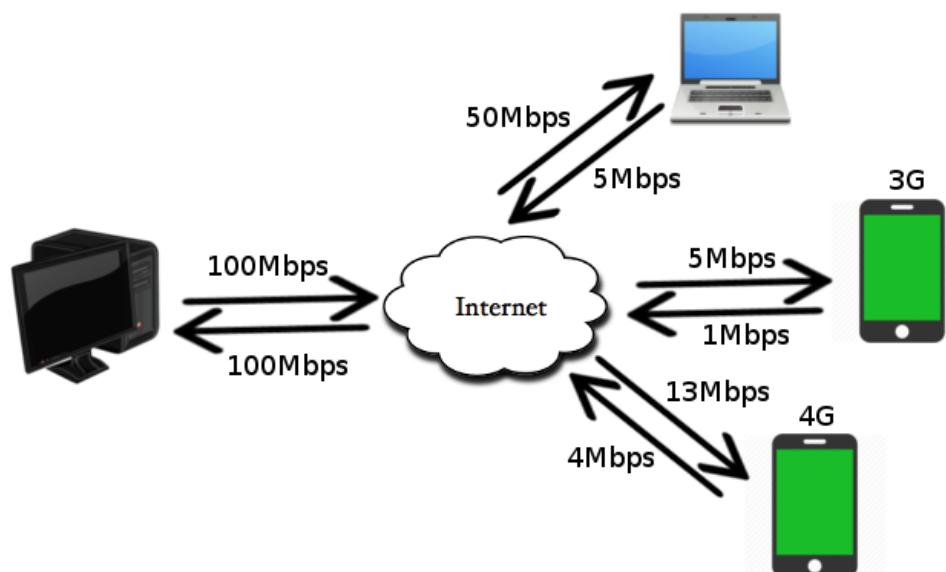


Figura 5.14: Montaje del servidor en la universidad y los clientes en otras redes

Capítulo 6

Conclusiones

En los capítulos anteriores se ha hecho una descripción del problema y se ha presentado la solución desarrollada. Además, se ha argumentado la elección final de diseño y se han presentado las pruebas realizadas. En este capítulo se analizarán las conclusiones a la luz del trabajo realizado y se proponen posibles líneas futuras de desarrollo.

6.1. Conclusiones

Tras analizar el trabajo realizado se puede apreciar que se ha conseguido el objetivo general. Se han creado 6 clientes web multiplataforma y multidispositivo para JdeRobot que permiten acceder sensores y actuadores de robots mediante el navegador web sin servidores intermedios, para ello se han programado 7245 líneas de código. Además se han hecho lo suficientemente modulares para que sea fácil extender su funcionalidad. Los 6 clientes están validados experimentalmente, valen tanto para sensores, robots y drones reales como para simulados, incluso se han exhibido en la Global Robot Expo, feria de robótica de Madrid¹

Se han conseguido todos los subobjetivos: se han creado `CameraViewJS` que recibe imágenes de `cameraserver`, `RgbdViewerJS` que visualiza datos de profundidad y color. También se han creado `KobukiVieweJS` y `UavViewerJS` que permiten teleoperar los robots y drones del laboratorio de robótica de la URJC. El último subobjetivo también se ha cumplido, se han desarrollado `IntrorobKobukiJS` e `IntrorobUavJS` que permiten insertar código que gobierna el comportamiento autónomo de los robots y drones.

También se han satisfecho todos los requisitos del capítulo 2:

¹<http://blog.jderobot.org/estuvimos-en-global-robot-expo/>

- Los componentes desarrollados usan la última versión de JdeRobot: 5.3.1.
- Los seis clientes se conectan a los servidores ICE sin usar servidor web intermedio gracias a ICE JS. Cada conexión a dichos servidores se hace desde un hilo diferente para no sobrecargar la interfaz.
- Las comunicaciones con los servidores son en tiempo real.
- Son lo suficientemente maduros para incluirlos en el repositorio oficial² de JdeRobot.
- Son multiplataforma y multidispositivo, al estar desarrollados en tecnologías web, con un sólo desarrollo, se pueden ejecutar en cualquier sistema con navegador web, ya sean Windows, Linux, MacOS, IOS,...

Los resultados de este proyecto, documentación, código, vídeos y demás material están disponible en la página web del TFG ³.

6.2. Trabajos futuros

Este TFG sienta las bases para posteriores trabajos en el uso de las tecnologías web de última generación en JdeRobot. Hemos identificado cuatro líneas por las que se puede extender este TFG, primero se podrían crear extensiones para los navegadores con cada cliente, las extensiones permiten agregar funcionalidades que no vienen de serie a los navegadores, así por ejemplo se pueden bloquear anuncios, tener conexión directa con nuestros correos desde cualquier web,.... En nuestro caso ya no sería necesario un servidor HTTP que nos proporcione la página web del cliente sino que la extensión sería el propio cliente.

También se podrían adaptar **IntrorobUavJS** e **IntrorobKobukiJS** a **Node JS**. Así se podrían programar comportamientos autónomos de los robots en JavaScript usando ICE-JS para comunicar con los drivers y el núcleo de dichos clientes pero sin necesidad de GUI.

Otro futuro trabajo pueden ser modificar **API.Camera** para que permita la compresión de imágenes e incluso negociar la resolución de las mismas para aumentar el flujo. El servidor ya está preparado para ello pero los clientes no. Y en el caso de **RgbdViewerJS**

²<https://github.com/RoboticsURJC/JdeRobot/pull/322>

³<http://jderobot.org/Aitormf-tfg>

poderse conectar con la interfaz de la nube de puntos de `openni1Server`, que actualmente no es compatible con ICE JS por parte del servidor.

Por último, se puede mejorar la interfaz de los teleoperadores, en especial `UavViewerJS`. Actualmente en los móviles no se pueden tener los dos controles en la pantalla ya que son pequeñas, esto dificulta su uso. El objetivo sería poder manejar el drone con el móvil/tablet en horizontal usando las dos manos. Además de añadir el uso de *gamepads* para complementar los controles y botones.

Bibliografía

- [1] Proyecto JdeRobot <http://jderobot.org/>
- [2] Repositorio de JdeRobot <https://github.com/RoboticsURJC/JdeRobot>
- [3] Componente CameraServer JdeRobot <http://jderobot.org/index.php/Drivers#cameraserver>
- [4] Componente Openni1Server JdeRobot <http://jderobot.org/index.php/Drivers#OpenniServer>
- [5] Página oficial GStreamer <http://gstreamer.freedesktop.org/>
- [6] Componente GazeboServer JdeRobot <http://jderobot.org/index.php/Drivers#gazeboserver>
- [7] Componente Ardrone_server JdeRobot http://jderobot.org/index.php/Drivers#ardrone_server
- [8] Componente Kobuki_driver JdeRobot http://jderobot.org/index.php/Drivers#kobuki_driver
- [9] Página oficial Gazebo <http://gazebosim.org/>
- [10] Curso de HTML5 de W3C http://www.w3schools.com/html/html5_intro.asp
- [11] Curso de Javascript de W3C <http://www.w3schools.com/js/default.asp>
- [12] Guía de Javascript de Mozilla <https://developer.mozilla.org/es/docs/Web/JavaScript/Guide>
- [13] Curso de CSS3 de W3C http://www.w3schools.com/css/css3_intro.asp
- [14] Página oficial WebGL <http://get.webgl.org/>

- [15] Mediawiki Daniel Castellano (Surveillance 4.0)
<http://jderobot.org/D.castellanob-pfc>
- [16] Mediawiki Edgar Barrero (Surveillance 5.1) <http://jderobot.org/Aerobeat-colab>
- [17] Mediawiki Aitor Martínez (JdeRobot Web Clients) <http://jderobot.org/Aitormf-tfg>
- [18] Repositorio de Aitor Martínez (JdeRobot Web Clients) <https://svn.jderobot.org/users/aitormf/tfg/>
- [19] Página oficial ICE <http://www.zeroc.com/>
- [20] Ice 3.5.1 Documentation <https://doc.zeroc.com/display/Ice35/Home>
- [21] Uso de los compiladores de Slice <https://doc.zeroc.com/display/Ice36/Using+the+Slice+Compilers>
- [22] Página de Ice for Javascript <https://zeroc.com/labs/icejs/index.html>
- [23] Websockets en ICE <https://zeroc.com/labs/icejs/websocket.html>
- [24] Página de Three.js <http://threejs.org/>
- [25] Documentación de Three.js <http://mrdoob.github.io/three.js/docs/>
- [26] Curso para aprender Three.js <http://stemkoski.github.io/Three.js/>
- [27] Página de JQuery <https://jquery.com/>
- [28] Página de Bootstrap <http://getbootstrap.com/>