



Grado en Ingeniería en Sistemas Audiovisuales y Multimedia

Escuela Técnica Superior de Ingeniería de Telecomunicación

Curso académico 2016-2017

Trabajo Fin de Grado

Interacción entre YouTube y el entorno
JdeRobot usando streaming

Autor: Alberto Pavo Blas
Tutor: José María Cañas Plaza

Curso académico 2016/2017

Índice general

1. Introducción	7
1.1. Tecnologías Web	8
1.1.1. HTTP	8
1.1.2. Navegadores	9
1.1.3. HTML	9
1.1.4. Multimedia en la web	10
1.2. Streaming	10
1.2.1. Youtube	12
1.3. Antecedentes	13
1.3.1. Aplicación web para Videovigilancia	13
1.3.2. Tecnologías web en JdeRobot	14
1.3.3. Drone WebRTC	15
2. Planificación	17
2.1. Objetivos	17
2.2. Metodología	18
2.3. Plan de trabajo	19
3. Tecnologías usadas	20
3.1. Streaming	20
3.1.1. Protocolos Streaming	21
3.2. YouTube	24
3.2.1. Procesado del contenido subido	24
3.2.2. Reproducción	25
3.2.3. YouTube Live Streaming	25
3.3. FFmpeg	30
3.3.1. FFmpeg streaming	31
3.4. Open Broadcaster Software	31
3.5. JdeRobot	32
3.5.1. ArDroneServer	33
3.5.2. CameraServer	34
3.5.3. Plugins de Gazebo que incluyen Camera	35
3.6. Gazebo	35

4. Aplicación web de difusión por YouTube	37
4.1. Diseño	37
4.2. Ejecución de Python en NodeJS	38
4.3. Servidor NodeJS: Diálogo con YouTube	40
4.3.1. Seguridad	40
4.3.2. Creación de eventos	41
4.3.3. Recuperación de información de evento	42
4.3.4. Terminar la retransmisión del evento	43
4.4. Servidor NodeJS: Comunicación con Ffmpeg	44
4.5. Dialogo HTTP e interfaz de cliente	45
4.5.1. Seguridad	45
4.5.2. Diálogo HTTP	46
4.5.3. Interfaz	48
4.6. Experimentos	50
6. Streaming web con YouTube desde un Drone	61
6.1. Diseño	61
6.2. Adaptador ffmpeg a JdeRobot	62
6.3. Comunicacion NodeJS	65
6.4. Experimentos	66
6. Streaming web con YouTube desde un Drone	61
6.1. Diseño	61
6.2. Adaptador ffmpeg a JdeRobot	62
6.3. Comunicacion NodeJS	65
6.4. Experimentos	66
7. Servidor Imágenes de vídeos en la red	70
7.1. Diseño	70
7.2. Comunicación con YouTube	71
7.3. Extracción de fotogramas y comunicación ICE	72
7.4. Experimentos	74
8. Conclusiones	76
8.1. Lineas Futuras	77

Índice de figuras

1.1.	Diagrama comunicación HTTP	9
1.2.	Multimedia	10
1.3.	Streaming en un quirófano	11
1.4.	Distribución tráfico de Internet	12
1.5.	UAViewerJS	14
1.6.	Interfaz de manejo drone Remoto	15
1.7.	Arquitectura de la aplicación	16
2.1.	Desarrollo en Espiral	18
3.1.	Arquitectura DASH	23
3.2.	Arquitectura HLS	24
3.3.	Protocolo Oauth YouTube	26
3.4.	Interfaz OBS	32
3.5.	Estructura ICE	33
3.6.	Arquitectura ArDroneServer	34
3.7.	Modelo CameraServer	35
3.8.	Escenario Quadrotor2 Gazebo	35
3.9.	Escenario creado con gazebo	36
4.1.	Arquitectura SurveillanceApp	38
4.2.	Diagrama de la Aplicación	48
4.3.	Interfaz Creación de Eventos	50
4.4.	Experimento	51
6.1.	Arquitectura Aplicación	62
6.2.	Esquema adaptador JdeRobot a ffmpeg	63
6.3.	Técnica de doble buffer	65
6.4.	Arquitectura Experimento	67
6.5.	arDrone de Parrot	67
6.6.	Arquitectura Experimento	69
6.7.	Experimento	69
6.1.	Arquitectura Aplicación	62
6.2.	Esquema adaptador JdeRobot a ffmpeg	63
6.3.	Técnica de doble buffer	65
6.4.	Arquitectura Experimento	67

ÍNDICE DE FIGURAS

6

6.5. arDrone de Parrot	67
6.6. Arquitectura Experimento	69
6.7. Experimento	69
7.1. Arquitectura YouTubeServer	71
7.2. Vídeo de YouTube mostrado en un cliente web normal y en la aplicación UAVviewer de JdeRobot usando el driver YouTubeServer	75

Capítulo 1

Introducción

El término multimedia se utiliza para referirse a cualquier objeto o sistema que utiliza múltiples medios de expresión físicos o digitales para presentar o comunicar información. Es una tecnología que permite integrar texto, números, gráficos, imágenes fijas o en movimiento, sonidos, alto nivel de interactividad y además, las posibilidades de navegación a lo largo de diferentes documentos. Las presentaciones multimedia pueden verse en un escenario, proyectarse, transmitirse, o reproducirse localmente en un dispositivo por medio de un reproductor multimedia.

El *streaming* (retransmisión), es una forma de distribuir contenido multimedia, de forma que el usuario consume dicho contenido a la vez que se descarga. Dentro de este campo encontramos el llamado *live streaming* cuya característica radica en que el contenido multimedia es retransmitido en directo, sin necesidad de ser grabado anteriormente por lo que el usuario lo consume en tiempo real. En los últimos años estas tecnologías han experimentado un gran crecimiento y han surgido múltiples plataformas que lo soportan (Youtube , Twitch , Periscope) proporcionando a los desarrolladores distintas opciones para crear aplicaciones web que manejen estas tecnologías.

Por otro lado, otro mundo que ha crecido rápidamente en este tiempo es el mundo de los drones. Estos vehículos aéreos no tripulados, se pueden usar para la grabación o retransmisión de eventos incluyendo una cámara en su diseño. Finalmente todos estos avances van de la mano del desarrollo de tecnologías web que permiten crear aplicaciones cada vez más sofisticadas y potentes aportando nuevas funcionalidades sin la necesidad de instalar nada en tu ordenador.

La temática principal de este proyecto gira entorno a todos estos campos, la recogida del contenido multimedia, procesado y su posterior retransmisión a través de una plataforma web que será YouTube.

A continuación se incluye una introducción a las tecnologías web , al *streaming* y las tecnologías y protocolos que lo respaldan, así como de distintas plataformas que permiten retransmisión en directo de eventos, principalmente YouTube que es la elegida para este proyecto.

1.1. Tecnologías Web

Las tecnologías web¹ son aquellas que se encargan de resolver el acceso y manejo de recursos alojados en Internet o en las *intranets*. Como es un campo muy extenso se hará una pequeña introducción de las más importantes y relevantes para el proyecto presentado.

1.1.1. HTTP

Para empezar a hablar de la web primero hay que resaltar el protocolo usado para la transferencia de información, HTTP(Hypertext Transfer Protocol). HTTP define la sintaxis y la semántica que utilizan los elementos de software de la arquitectura web (clientes, servidores, proxies) para comunicarse.

Los mensajes en HTTP se encuentran en texto plano haciendo que estos sean más legibles y fáciles de procesar. También cuenta con una serie de métodos o peticiones que hacen referencia a acciones, como puede ser el método **GET** cuyo objetivo es la obtención de recursos, otro método importante es **POST**, este método es el encargado de enviar datos alojados en el cuerpo del mensaje para que posteriormente sean procesados, **DELETE** y **PUT** son otros métodos destacados del protocolo. HTTP también añade códigos de respuesta en función del resultado del procesamiento de nuestra petición, lo que indica si ha tenido éxito o por el contrario ha fracasado. Por último el mensaje HTTP incluye una serie de cabeceras formadas por metadatos que se envían en las peticiones o respuesta HTTP para proporcionar información esencial sobre la transacción en curso.

HTTP es un protocolo sin estado, es decir, no guarda ninguna información sobre conexiones anteriores. El desarrollo de aplicaciones web necesita frecuentemente mantener estado. Para esto se usan las *cookies*, información que un servidor puede almacenar en el sistema cliente. Esto le permite a las aplicaciones web instituir la noción de sesión, y también permite rastrear usuarios ya que las *cookies* pueden guardarse en el cliente por tiempo indeterminado

¹<http://www.w3c.es/Consorcio/about-w3c.html>

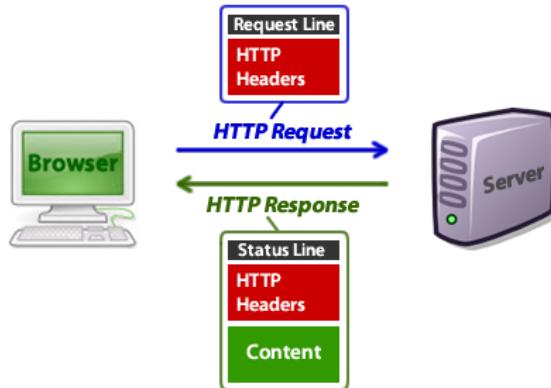


Figura 1.1: Diagrama comunicación HTTP

1.1.2. Navegadores

Otra parte fundamental de la web son los navegadores: un software, aplicación o programa que permite el acceso a la Web, mostrando al usuario la información contenida en la misma, fotos, documentos, vídeos, audio... Además permite la interacción del usuario con la misma. Los navegadores poseen la tecnología necesaria para interpretar el código que posee la información y mostrarla al usuario. Los navegadores son los encargados de comunicarse con los servidores web, que alojan la información, a través del protocolo HTTP, o derivados como puede ser HTTPS, protocolo más seguro que el citado anteriormente.

En la actualidad existen distintos navegadores pero los mas usados son Google Chrome desarrollado por Google, Firefox navegador de código abierto coordinado por la Corporación Mozilla y la Fundación Mozilla e Internet Explorer desarrollado por Microsoft.

1.1.3. HTML

HTML, *HyperText Markup Language*, es un lenguaje de marcado usado en la elaboración de páginas web. La última versión del estándar de HTML publicada por *World Wide Web Consortium* es HTML5.

HTML es el encargado de estructurar el contenido de la página web que combinado con JavaScript u otros lenguajes es capaz de dotar a la página de funcionalidades. Dicho lenguaje se escribe a través de etiquetas encasilladas entre corchete angulares, cada una de estas etiquetas son elementos de la página web que pueden contener atributos y valores, podemos insertar imágenes, vídeos, sonido, texto....

Como comentamos anteriormente la última versión es HTML5 donde se han añadido nuevos elementos como el audio o vídeo, o nuevas etiquetas para la estructuración

de contenido entre otros.

1.1.4. Multimedia en la web

La parte multimedia ha tomado gran importancia en las aplicaciones web, tanto es así que la mayor parte de flujo de datos en internet está destinada a este tipo de contenido. Al mencionar la palabra multimedia lo primero que se nos viene a la mente es audio y vídeo, que por otro lado son las partes mas importantes, pero este concepto también engloba otros campos como pueden ser animaciones o juegos en red.



Figura 1.2: Multimedia

Dado a este gran crecimiento en la última revisión de HTML, lenguaje de programación en el que se encuentran escritas las webs, HTML5 incluyó dos importantes elementos para el ámbito multimedia, vídeo y audio. En esta versión también aparecieron elementos gráficos como **Canvas** o **SVG** con los que podemos realizar animaciones o incluso juegos.

Con la inclusión de estos nuevos elementos se puede incrustar vídeo o audio con gran facilidad en un documento HTML. El elemento vídeo, nos permite un control total, pudiéndole añadir distintas calidades, subtítulos o por ejemplo controles de vídeo.

Aprovechando todo esto en los últimos tiempo se han desarrollado tecnologías web orientadas a la comunicación en tiempo real entre dos navegadores, WebRTC². La idea de WebRTC consiste en comunicar dos navegadores en tiempo real sin necesidad de ningún servidor intermedio, de forma que estos navegadores puedan intercambiar datos de vídeo, audio, u archivos. Esta tecnología ha dado lugar principalmente a aplicaciones de videoconferencia como Hangouts, desarrollada por Google.

1.2. Streaming

Como se ha mencionado anteriormente el *streaming* consiste en poder consumir contenido multimedia sin que este haya sido previamente descargado. Antes de la aparición de dicha tecnología en 1995 de la mano de la aplicación *RealAudio*, basada en la retransmisión de audio a través de internet, era necesario descargar y almacenar en el disco duro dicho contenido al completo antes de poder ser consumido. Tras la

²<https://webrtc.org/>

aparición de esta primera aplicación de *streaming* de audio le siguieron otras muchas incluyendo además el vídeo.

Este progreso siempre ha ido ligado a un factor limitante, el ancho de banda que se encuentra estrechamente relacionado con la difusión de vídeo, ya que tanto como para retransmitir un evento como para consumirlo con cierta calidad y sin esperas necesitamos un ancho de banda aceptable, que en España con la llegada de la fibra óptica se ha conseguido en los últimos años. Otro factor que ha ralentizado el desarrollo, es el estado de los equipos ya que estos no poseían la suficiente potencia para visualizar correctamente estas transmisiones.

Otro de los campos en los que el *streaming* se está asentando es en el televisivo, plataformas como Netflix, HBO , Hulu o muchas otras están sustituyendo a la televisión tradicional. Estas plataformas permiten visualizar cierto contenido en cualquier momento sin estar sujeto a un horario, lo cuál es la principal ventaja de estas plataformas.

Con la expansión del *streaming* se abrieron nuevos campos de desarrollo e investigación así nació el *live streaming*, que consiste en retransmitir eventos en directo a través de internet. Con este fin se desarrollaron multitud de aplicaciones como YouTube live events, periscope, yomvi ... que ofrecen la posibilidad de consumir o retransmitir eventos en directo.

A parte del entretenimiento, el *streaming* es usado con otros fines como puede ser la enseñanza donde se obtiene una gran libertad ya que tanto alumno como profesor pueden estar en distintas partes del mundo. Uno de los ejemplos mas claros de este uso podemos observarlo en la medicina, donde se han hecho retransmisiones en directo de operaciones quirúrgicas de forma que tanto alumnos como otros profesionales de la medicina puedan aprender nuevas técnicas. Otra de las áreas donde se ha implantado esta tecnología es en la vídeo vigilancia, donde a través de la red IP se monitoriza la actividad del lugar deseado.



Figura 1.3: Streaming en un quirófano

A continuación se presenta un gráfico donde se puede verla evolución del consumo

multimedia.

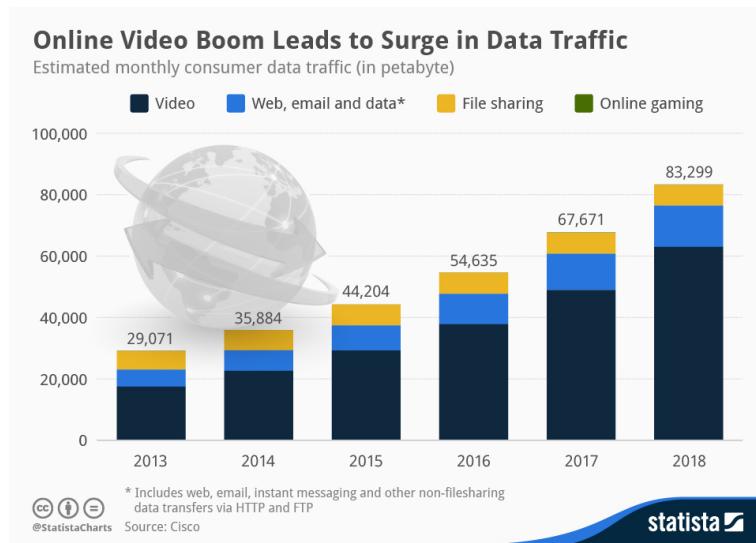


Figura 1.4: Distribución tráfico de Internet

1.2.1. Youtube

Desde su creación en 2005 YouTube ha experimentado un crecimiento brutal, de forma que hoy en día cuenta con más de mil millones de usuarios a nivel mundial subiéndose alrededor de 300 horas de video por minuto. Tras su lanzamiento solo un año después, en 2006 , Google compró la compañía por 1.650 millones de dólares, tras esto se aumentó el número de videos que la plataforma contenía a lo que le siguieron multitud de avances, se incluyeron listas de reproducción y videos relacionados, traducción a distintos tipos de idiomas en forma de subtítulos, canales en los cuales cada usuario puede publicar su contenido para que sea visto, se dio la posibilidad de añadir anuncios a los videos, videos de mayor duración, desarrollo de una aplicación móvil, contenido de alta definición y eventos en directo.

Pero no todo es positivo, uno de los problemas más importantes que YouTube arrastra es la piratería y la publicación de contenido inapropiado. Debido a la gran cantidad de volumen de videos que maneja YouTube se hace muy complicado controlarlo todo de forma que la piratería esta proliferando en esta plataforma, se puede encontrar música o películas publicadas sin posesión de los derechos de autor, para intentar paliarlo YouTube creo el *Content-ID* de forma que los propietarios de los derechos de autor puedan identificar y gestionar fácilmente su contenido en YouTube. Aún así se acusa a Google de hacer la vista gorda ante este tipo de contenido.

YouTube Live

A principios del año 2011 YouTube puso a disposición de los usuarios la posibilidad de emitir eventos en directo de forma gratuita. Para la emisión de estos eventos es necesario únicamente un codificador que recoja el flujo de datos de tu ordenador y lo transfiera a YouTube. En este momento existen bastantes alternativas tanto gratuitas

como de pago. Estos eventos pueden ser programados para una fecha concreta, poseen opciones de privacidad, puede emitir más de un evento a la vez y pueden ser estos grabados.

Actualmente YouTube esta perfeccionando otra modalidad de emisión en la cual la retransmisión se realiza inmediatamente, a diferencia de los eventos el proceso de transcodificación es llevado a cabo por YouTube quien automáticamente detecta la resolución y frecuencia de tu transmisión.

Otra de las funcionalidades que YouTube está incorporando actualmente es la retransmisión en directo con calidad 4K y de grabaciones de 360°.

Todos estos avances han supuesto un gran salto ya que cualquier usuario sin una gran infraestructura puede realizar una emisión en directo. A continuación se adjunta una tabla con los requisitos mínimos de las emisiones extraído de YouTube³.

CALIDAD	RESOLUCIÓN	TASA DE BITS
240P	426X240	300-700 Kbps
480P	854X480	500-2000 Kbps
720P	1280X720P	1500-4000 Kbps
1080P	1920X1080	3000-6000 Kbps
4k/2160P a 30 FPS	3480X2160	13.000-34.000 Kbps

Cuadro 1.1: Ancho de banda recomendado en función de la calidad

1.3. Antecedentes

Como contexto y punto de partida para este proyecto, sobre todo en el ámbito del manejo de UAV, se tomaron tres trabajos anteriores realizados por alumnos de la URJC, todos ellos apoyados en el software JdeRobot que nos proporciona herramientas para el manejo y control de estos dispositivos.

1.3.1. Aplicación web para Videovigilancia

Este es un trabajo fin de grado realizado por Edgar Barreiro, donde desarrolla una aplicación llamada *Surveillance 5.1*⁴.

Surveillance 5.1 esta desarrollado usando Ruby on Rails, un entorno de código abierto para el desarrollo de aplicaciones web. El servidor web se conecta a componentes JdeRobot que ofrecen interfaces ICE de objetos distribuidos. De esta forma obtiene datos de los distintos sensores y actuadores de la aplicación. En el lado cliente, el navegador refresca estos datos realizando peticiones AJAX.

El sistema *Surveillance 5.1* desarrollado en este Trabajo de Fin de Grado une estas dos tecnologías ofreciendo una aplicación web atractiva e intuitiva orientada a controlar remotamente sensores y actuadores domóticos de un hogar. La aplicación web que se describe en esta memoria ofrece un flujo de vídeo desde una cámara web, un

³<https://support.google.com/youtube/answer/2853702?hl=es>

⁴<http://jderobot.org/Aerobeat-colab>

flujo de imagen de profundidad procedente de un sensor Kinect y su representación en 3D. También ofrece el acceso a un sensor de humedad y un actuador como ejemplos de dispositivos domóticos de bajo coste.

Cabe destacar que esta aplicación es accesible remotamente desde cualquier navegador.

1.3.2. Tecnologías web en JdeRobot

Este trabajo fin de grado realizado por Aitor Martínez⁵ consiste en cuatro aplicaciones web *Camera ViewJS*, *RGBD ViewerJS*, *KobukiViewerJS* y *UavViewerJS*. Para familiarizarnos un poco más con el proyecto se explicaran brevemente las cuatro aplicaciones webs que forman este trabajo.

- Camera ViewJS, este cliente nos permite visualizar las imágenes tomadas por una cámara conectada al drone.
- RGBD ViewerJS, proporciona los datos de color y profundidad
- Kobuki ViewerJS, se trata de un teleoperador capaz de manejar y monitorizar datos de los robots Kobuki y Pioneer del laboratorio de la URJC.
- UAVViewerJS, a través de esta herramienta es posible teleoperar drones a la vez que se pueden visualizar los sensores del drone.

Estas aplicaciones se encuentran desarrolladas en JavaScript obtienen información de actuadores y sensores del drone a través de los servidores de JdeRobot con los cuales se comunica mediante *websockets*. También es capaz de recuperar imágenes de un drone fotograma a fotograma desde el servidor de imágenes de JdeRobot y mostrarlas en tiempo real en la aplicación, funcionalidad que mas tarde se verá como se ha implementado en el trabajo expuesto.

Estos clientes o aplicaciones son compatibles también con el navegador web del teléfono móvil.

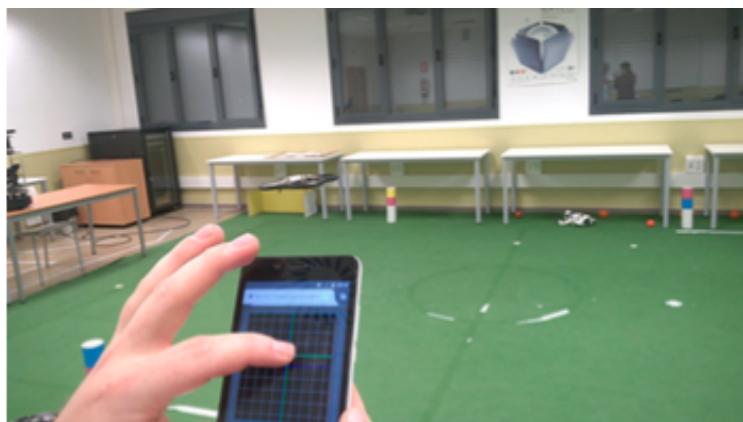


Figura 1.5: UAVViewerJS

⁵<http://jderobot.org/Aitormf-tfg>

1.3.3. Drone WebRTC

Este proyecto fue realizado por Iván Rodríguez⁶ y consiste en combinar la tecnología WebRTC, con las herramientas proporcionadas por JdeRobot de forma que el resultado final es una aplicación web en la que podemos teleoperar un cuadricóptero con la ayuda de controles y de un flujo de vídeo capturado por una cámara incorporada en el drone.



Figura 1.6: Interfaz de manejo drone Remoto

WebRTC es una tecnología que se encarga de la comunicación entre navegadores, en este proyecto la comunicación se realiza entre un navegador a borde del drone y otro navegador situado en un ordenador remoto. A través de las herramientas de JdeRobot y junto con IceJS, las instrucciones y datos de los sensores y actuadores del drone son transferidas del navegador del drone al ordenador remoto y viceversa de esta forma se puede realizar la teleoperación del aparato.

Por otro lado de la toma de imágenes se encarga el navegador , el cual recupera las imágenes captadas por una cámara web incorporada al drone y se encarga de transmitirlas vía webRTC al ordenador local.

⁶<http://jderobot.org/Irodmar-tfg>

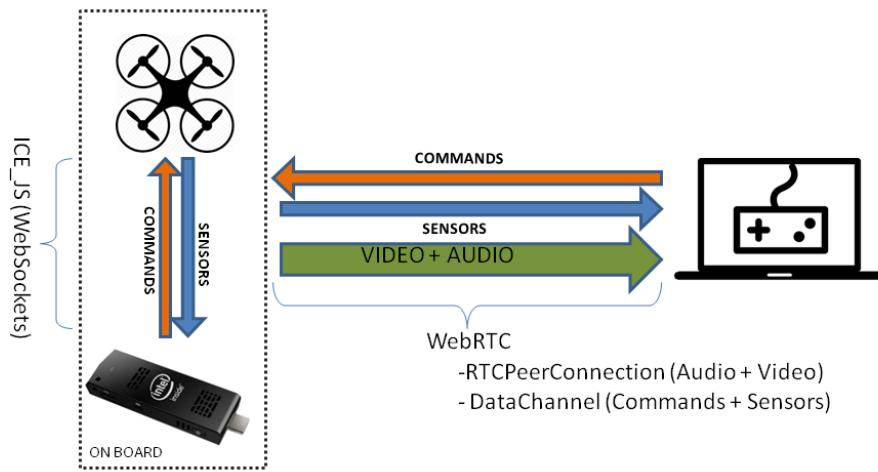


Figura 1.7: Arquitectura de la aplicación

En el proyecto que se presenta a continuación se hace uso de las herramientas proporcionadas por JdeRobot y YouTube, con el objetivo de crear aplicaciones que recojan un flujo audiovisual y sea publicado en YouTube a tiempo real.

Una vez puesto en contexto el proyecto se presentarán los objetivos del mismo así como las tecnologías usadas y la explicación del software desarrollado para dar paso finalmente a las conclusiones.

Capítulo 2

Planificación

A continuación se presentan los objetivos del trabajo así como las distintas etapas planificadas para conseguirlos.

2.1. Objetivos

El objetivo general planteado para este trabajo consiste en construir aplicaciones que automaticen el proceso de retransmitir contenido audiovisual a través de YouTube en tiempo real, así como automatizar la visualización de contenido YouTube dentro de aplicaciones de la plataforma de software robótico JdeRobot.

Este trabajo aborda la interacción con YouTube desde aplicaciones JdeRobot tanto de subida como bajada, es decir tanto para difundir vía YouTube las imágenes captadas por un drone como para conectar aplicaciones de procesamiento de imágenes en JdeRobot con flujos de vídeo emitidos por YouTube

Para la consecución de este objetivo, el trabajo se divide en tres subobjetivos, el desarrollo de tres aplicaciones de procesado de vídeo en tiempo real a través de YouTube.

1. Diseñar y desarrollar una aplicación web que debe ser capaz de retransmitir vía YouTube el contenido captado por una cámara web conectada a un ordenador.
2. Diseñar y desarrollar una aplicación web que, con ayuda de las herramientas de JdeRobot, retransmite a través de YouTube las imágenes captadas por un drone en tiempo real.
3. Diseñar y desarrollar un driver que ofrezca imágenes en el interfaz de JdeRobot para flujos de vídeo provenientes de YouTube.

A parte de los objetivos se han marcado una serie de requisitos para el desarrollo del proyecto.

- En el desarrollo de estas aplicaciones buscamos crear prototipos que integren las tecnologías JdeRobot y YouTube. No es necesario que estas aplicaciones funcionen con una tasa de imágenes por segundo de tiempo real.

- El driver planteado en el tercer subobjetivo, será integrado en el repositorio oficial de JdeRobot y por lo tanto debe ser compatible con versión 5.5, su última versión.
- Otro requisito acordado con el tutor del proyecto, es que este debe ser de código abierto.

2.2. Metodología

La metodología elegida al principio del proyecto ha sido el modelo de desarrollo en espiral. Dicho modelo es ampliamente usado en el desarrollo de software. Consiste en definir un conjunto de actividades que se repiten hasta el final del proyecto, a su vez estas actividades se dividen en subtareas, de forma que al final de cada iteración se establecen puntos de control en los que se evalúa el trabajo realizado hasta el momento y se proponen futuros objetivos.

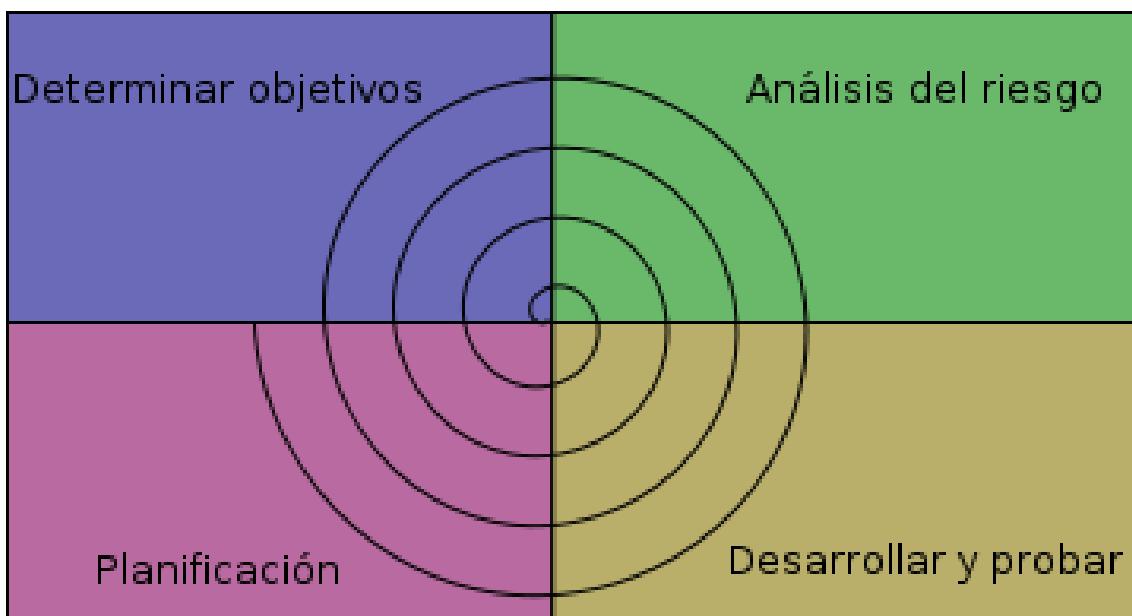


Figura 2.1: Desarrollo en Espiral

Cada ciclo o iteración está formado por cuatro tareas.

1. **Determinar los objetivos.**
2. **Análisis de riesgo**, se evalúan las posibles consecuencias negativas de las decisiones tomadas.
3. **Desarrollar**, las actividades propuestas.
4. **Planificar**, las siguientes etapas del proyecto.

Durante todo el proyecto se han realizado reuniones semanales con el tutor con el objetivo de evaluar los avances, y a partir de ellos fijar los siguientes pasos en el proyecto. Todo este trabajo se encuentra reflejado en un mediawiki¹, bitácora de los progresos y escaparate con los vídeos de los desarrollos conseguidos.

Por último el desarrollo para el desarrollo del código fuente se ha usado la plataforma de control de versiones GitHub².

2.3. Plan de trabajo

La primera parte es comprender y desarrollar el contexto relacionado con YouTube, las tecnologías *streaming*, así como las herramientas de retransmisión de contenido compatibles con YouTube y las librerías de Google para el desarrollo de aplicaciones relacionadas con YouTube todo esto nos permitirá desarrollar la primera aplicación.

A continuación se comenzará con el segundo subobjetivo, que implica el estudio de las tecnologías JdeRobot y la familiarización con su entorno.

Con los conocimientos de los puntos anteriores se pasará al desarrollo del driver para JdeRobot.

¹<http://jderobot.org/Apavo-tfg>

²<https://github.com/RoboticsURJC-students/2016-tfg-alberto-pavo>

Capítulo 3

Tecnologías usadas

Una vez introducidos en el proyecto y con los objetivos marcados, en este capítulo se hablará de las tecnologías usadas para el desarrollo de la aplicación más en profundidad. Las dos principales tecnologías sobre las que gira el proyecto son *Youtube live streaming API* y JdeRobot.

3.1. Streaming

Desde el punto de vista del usuario el uso del *streaming* es el siguiente. El usuario se conecta al servicio que le proporciona el contenido multimedia, dicho contenido se encuentra almacenado en servidores que contienen los archivos codificados en formatos como MP3, VP8, AVI... Una vez establecida la conexión comienza la transmisión que está basada en protocolos “ágiles” de transporte como RTSP, RTCP, UDP etc... La transmisión se hace en pequeñas partes de forma que se consigue una transmisión mas rápida al ser menos pesada. Por otro lado, para garantizar una reproducción continua se incluye un buffer en el cual van siendo almacenadas partes del contenido, de forma que el usuario no necesita descargar el contenido completo del archivo para empezar a reproducir, sino que solo necesita pequeñas partes de él.

En el proceso de *streaming* se deben tener en cuenta varios factores que pueden limitar la calidad del mismo. Desde el lado del consumidor el factor mas limitante es el ancho de banda, aunque con las nuevas velocidades de conexión casi cualquier compañía telefónica proporciona un ancho de banda suficiente. En el cuadro 1.1 se adjunta las recomendaciones de ancho de banda de Netflix en función de la calidad de vídeo.

CALIDAD	Mbps	GB POR HORA
STANDARD DEFINITION(SD)	3 MB	0,7 MB
HIGH DEFINITION(HD)	5 MB	3 MB
ULTRA HIGH DEFINITION(4K)	25 MB	7 GB

Cuadro 3.1: Ancho de banda recomendado en función de la calidad

Si nos situamos en el lado emisor, se deben tener en cuenta otros factores:

- **Codec**, es un programa o dispositivo hardware capaz de codificar o decodificar una señal o flujo de datos digitales. El códec elegido afecta tanto a la calidad como a la velocidad de transmisión del archivo, ya que a mayor calidad de codificación mayor tasa de bits necesitamos. Uno de los mas usado es H264 para vídeo que está dejando paso a su sucesor H265, ambos permiten codificar vídeo de alta calidad.
- **BitRate**, el *bitrate* o tasa de bits representa la cantidad de bits que se envían por unidad de tiempo y es uno de los factores más importantes a la hora de producir una retransmisión de calidad aunque este factor está limitado por el ancho de banda de subida disponible. En este punto cabe destacar el uso del *multi-bitrate* que consiste en enviar distintas señales cada una con un *bitrate* diferente de forma que nos aseguramos que este contenido pueda ser consumido por todo tipo de conexiones.
- **Key Frame**, también conocido como *i-frame*, este fotograma representa una imagen completa y sirve de referencia a las demás imágenes en la que el codificador solo almacena las diferencias entre una y otra. Cuanto mayor sea el periodo de tiempo entre *KeyFrames*, menos datos se transmitirán pero peor será la calidad del vídeo.
- **FrameRate**, son las imágenes por segundo con las que se reproduce el vídeo. A mayor *framerate* mayor calidad y más pesado será el archivo.

3.1.1. Protocolos Streaming

Antes de hablar sobre los protocolo de *streaming*, se debe hacer una pequeña introducción sobre UDP y TCP, protocolos del nivel de transporte. UDP (User Datagram Protocol) se encarga de enviar datagramas a través de la red sin necesidad de establecer una conexión previa, tampoco posee información de flujo ni confirmación por lo que los paquetes pueden llegar desordenados o no llegar.

TCP (Transmission Control Protocol), su función al igual que UDP es la del transporte de datos. La diferencia entre ambos protocolos es que TCP además de requerir un establecimiento de conexión previo a al envío de datos, asegura que los paquetes llegarán ordenados y sin perdidas gracias a su mecanismo de asentimientos(ACK) y reenvío de paquetes. En su contra, el uso de este protocolo significa un mayor retardo en la comunicación frente a UDP.

Existen diversos protocolos para sostener la tecnología *streaming*, en esta sección vamos a centrarnos en dos tipos de protocolos. El primer grupo no se apoya en el protocolo HTTP para realizar el *streaming*, como son el protocolo RTP y RTMP. Por otro lado protocolos como DASH o HLS usan HTTP para *streaming*.

- RTP

RTP son las siglas de Real-time Transport Protocol es un protocolo de nivel de sesión utilizado para la transmisión de información en tiempo real, como por ejemplo audio vídeo o datos. Junto a este protocolo se suele usar RTCP

(RTP Control Protocol) es un protocolo de comunicación que proporciona información de control que está asociado con un flujo de datos para una aplicación multimedia. Este protocolo no transporta ningún dato por si mismo, se encarga de transmitir paquetes de control, datos de la conexión, bytes enviados, control de calidad ... Estos protocolos se encapsulan sobre UDP

RTSP es un protocolo de señalización (Real Time Streaming Protocol) establece y controla uno o muchos flujos sincronizados de datos, ya sean de audio o de vídeo. Es un protocolo no orientado a conexión, en lugar de esto el servidor mantiene una sesión asociada a un identificador, en la mayoría de los casos RTSP usa TCP para datos de control del reproductor y UDP para los datos de audio y vídeo. RTSP es similar a HTTP a excepción de que introduce nuevos métodos y necesita mantener el estado de la conexión. Los métodos más importantes del protocolo son

- **Describe**, se solicita una descripción de un objeto multimedia contenido en el servidor. Con esta petición se comienza el diálogo.
- **Setup**, especifica como serán transportados los datos que suele incluir el puerto para recibir los datos RTP (audio y vídeo) y los datos de control RTCP.
- **Play**, esta petición provoca que el servidor comience a enviar el flujo de datos.
- **Pause**, detiene temporalmente el flujo de datos.
- **Teardown**, finaliza el envío de datos y libera los recursos usados.

■ RTMP

Protocolo de mensajería en tiempo real también conocido como *flash* y desarrollado por Adobe. Se trata de un protocolo basado en TCP que mantiene conexiones persistentes y comunicación en baja latencia. El flujo de información, en el cual se encuentran multiplexados datos multimedia y de conexión, es dividido en distintos fragmentos de forma que se entreguen flujos de información con la mayor cantidad de datos posible y sin problemas, este tamaño es negociado entre el cliente y el servidor. Por otro lado RTMP define varios canales para el intercambio de datos, estos canales pueden estar activo simultáneamente. Este encapsula por encima suya en MP3 o AAC el audio y en FLV el vídeo. Este protocolo presenta distintas variaciones puede usarse con conexiones TLS/SSL (RTMPS) junto con encriptación (RTMPE) , encapsulado dentro de HTTP (RTMPT) o sobre UDP (RTMFP).

■ DASH

Dynamic Adaptative Streamming Over HTTP también conocido como MPEG-DASH es un protocolo de *streaming* adaptativo cuyo objetivo es modular la tasa de bits en función del estado de la red. Para ello su idea principal es disponer del contenido en diferentes calidades y fragmentado de forma que cada segmento temporal puede ser enviado en distintas calidades. DASH usa HTTP como su

protocolo de transporte lo que simplifica las conexiones a la hora de atravesar nats o cortafuegos. Cabe destacar que este protocolo es un estándar abierto de W3C¹

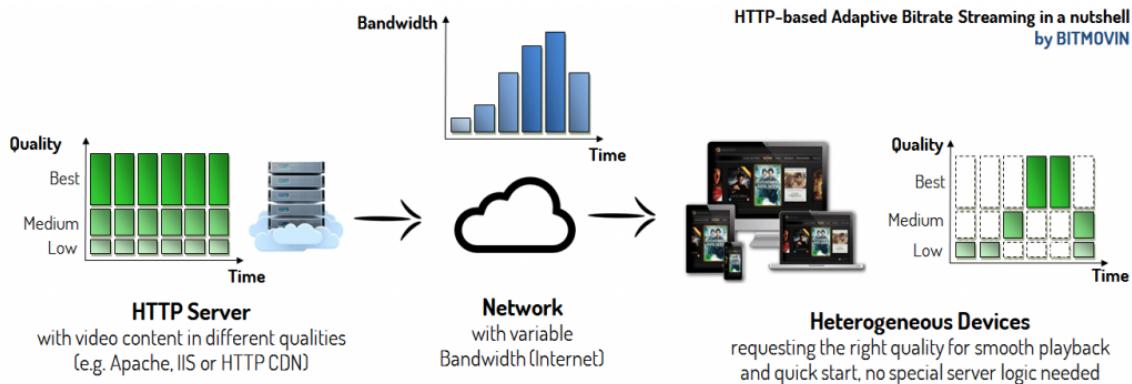


Figura 3.1: Arquitectura DASH

■ HLS

HTTP Live Streaming es un protocolo basado en HTTP implementado por Apple. Divide el flujo en pequeñas partes que son transmitidas, permitiendo al cliente a elegir el tipo de transmisión que más se adapte a su conexión. También implementa un mecanismo de codificación basado en AES.

La arquitectura del protocolo se divide en un servidor que se encarga de codificar y encapsular la entrada de vídeo para ello utiliza H.264 como códec de vídeo y MP3, HE-AAC o AC-3 como códec de audio. Un distribuidor, que es un servidor web convencional, procesa las peticiones y devuelve los recursos pedidos y por último un cliente que recibe el flujo de vídeo.

¹<https://www.w3.org/2011/09/webtv/slides/W3C-Workshop.pdf>

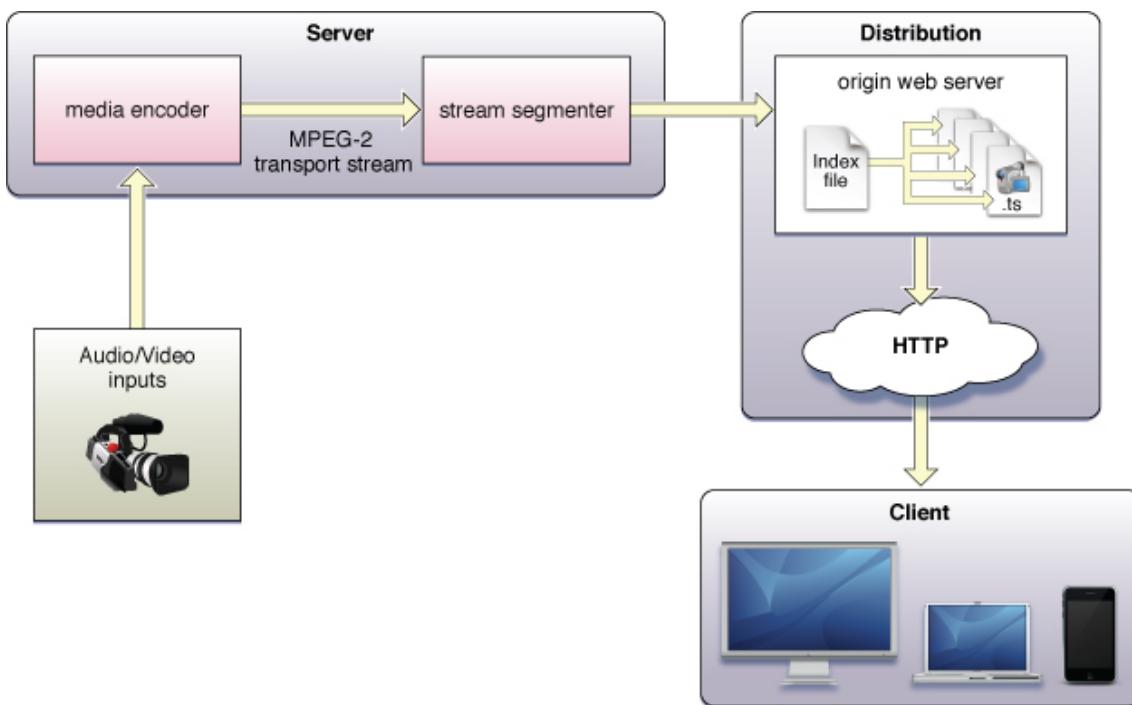


Figura 3.2: Arquitectura HLS

3.2. YouTube

Tras una breve introducción a YouTube en el primer capítulo, en este apartado revisaremos mas a fondo cómo funciona YouTube realmente y por otro lado el API de YouTube usada en el desarrollo de este proyecto.

Para comprender como funciona YouTube este apartado se dividirá en dos partes, en la primera se tratará el proceso de subida de vídeos a YouTube y la segunda parte se centrará en el proceso de reproducción.

3.2.1. Procesado del contenido subido

Los servidores de YouTube reciben vídeo en distintos formatos y calidades, no todos ellos son compatibles con los reproductores. Por lo que YouTube convierte estos vídeos a un formato genérico capaz de ser reproducido. De forma que nunca vemos el vídeo originalmente subido ya que aunque este tenga una calidad excelente sería demasiado pesado lo cual ralentizaría su reproducción y por tanto el usuario no disfrutaría de una buena experiencia de visualización.

La primera parte de la transformación del vídeo es el procesado, en el que averigua la calidad y *framerate* del mismo. Con estos datos se genera un primer archivo comprimido mínimamente llamado *mezzanine* y que posteriormente será comprimido de nuevo.

El siguiente paso consiste en dividir este vídeo en fragmentos, suelen ser de unos cinco segundos, estos fragmentos son procesados y comprimidos en paralelo por dis-

tintas máquinas dando lugar a distintas versiones de cada uno en distintos formatos de compresión.

Por último, estos fragmentos se unen formando vídeos con distintas resoluciones, que son procesados por *codecs* con el fin de reducir aún más su tamaño.

A todo este proceso se le debe añadir un software de análisis del contenido cuya función es buscar contenido protegido por derechos de autor o copyright, en caso de encontrarlo y el autor del vídeo no estar en posesión de dichos derechos ese contenido será eliminado por YouTube.

3.2.2. Reproducción

La segunda parte es la reproducción del contenido multimedia. En sus inicios YouTube tenía un único archivo de vídeo que era descargado y reproducido en la aplicación, este mecanismo generaba muchos retrasos y paros en la reproducción por lo que YouTube optó por dividir el vídeo en fragmentos. El tamaño de estos fragmentos se decide en función del estado de la red en lo que se llama *bitrate* adaptativo de forma que se minimicen las interrupciones.

Por otro lado la distribución de vídeo se hace a través de la llamada red de distribución de YouTube que se encuentra asociado con los distintos proveedores de internet repartidos por distintas localizaciones. Estos proveedores almacenan cierto contenido multimedia, de forma que cuando hacemos la petición de un vídeo no nos comunicamos con los servidores de YouTube directamente sino con el proveedor más cercano. Si nuestro proveedor no posee el contenido requerido este hace una petición a servidores superiores de tal forma que se establece una red, donde el servidor superior es el centro de datos de Google.

3.2.3. YouTube Live Streaming

Esta API creada por Google, forma parte de *YouTube data API* y permite crear y manejar los eventos en vivo de YouTube. Desde ella puedes programar eventos y asociarlos a un flujo. Antes de continuar explicando el API vamos a introducir unos términos que facilitaran su compresión.

- *Broadcast*, representa una emisión de un flujo multimedia, en este caso dicho flujo será vídeo y audio. YouTube le da el nombre de eventos en vivo de forma que permite programarlos a una hora determinada. Dentro del API se encuentra asociado a un recurso llamado `liveBroadcast`.
- *Streams*, se trata del flujo multimedia, audio y vídeo. Cada *stream* se encuentra asociado a una emisión, dentro del API se puede acceder a él a través de `liveStream`.

YouTube proporciona varias librerías que se encuentran disponibles en diversos lenguajes, algunas de ellas en fase de pruebas aún como las de JavaScript. Las librerías estables se encuentran desarrolladas en Java, Python y PHP. Para este proyecto por razones de compatibilidad se ha decidido usar las librerías escritas en Python.

A continuación se explicaran unos puntos importantes para comprender mejor el uso del API, como son los mecanismos de seguridad, los eventos en vivo y el manejo de estos a través de las librerías.

Seguridad

Para usar las funcionalidades que nos proporciona el API en nuestra aplicación debemos registrar dicha aplicación a través de *Google Developers Console*. Allí se conseguirán unas credenciales que darán acceso a su uso. Esto es necesario ya que acceder a YouTube, en este caso, significa acceder a un sitio privado con datos de usuario sensibles por lo que se necesitan mecanismos que los protejan.

Para este fin Google ha elegido el protocolo de seguridad Oauth 2.0 (Open Authorization). Dicho protocolo permite flujos simples de autorización para sitios web o aplicaciones informáticas. Además permite a terceros (clientes) acceder a contenidos propiedad de un usuario (alojados en aplicaciones de confianza, servidor de recursos) sin que éstos tengan que manejar ni conocer las credenciales del usuario. Es decir, aplicaciones de terceros, en este caso nuestro proyecto, pueden acceder a contenidos propiedad del usuario, nuestra cuenta de YouTube, pero estas aplicaciones no conocen las credenciales de autenticación.

La primera vez que accedes a tu aplicación a través del API serás redirigido al servidor de autenticación de Google donde debes dar tu autorización para que la aplicación pueda acceder a tus recursos de usuario, una vez aceptado se genera un *token* usado posteriormente por Oauth 2.0.

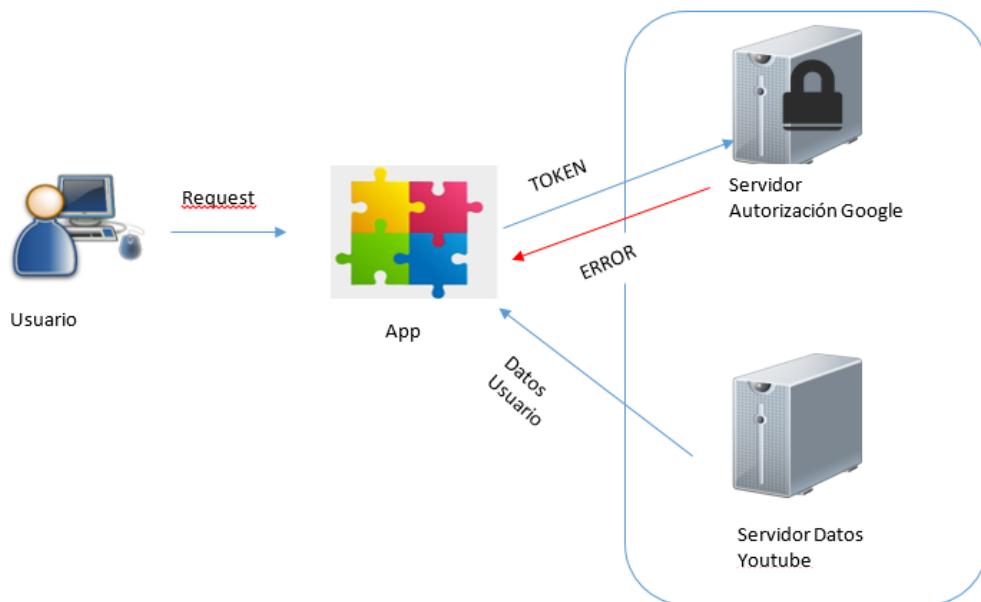


Figura 3.3: Protocolo Oauth YouTube

Recursos YouTube API: liveBroadcast y liveStream

La API presenta distintos recursos. A continuación presentaremos los recursos usado para este proyecto y cómo a través de su manipulación podemos manejar eventos en vivo desde una aplicación.

El primero, `liveBroadcast`, como su propio nombre indica, se encarga de manejar la información relacionada con la emisión del evento. Este recurso tiene asociadas ciertas propiedades como la hora del evento o la privacidad. Estas propiedades pueden ser definidas mediante un archivo en formato JSON, así a través de este archivo se puede configurar la emisión.

```
{  
  "kind": "youtube#liveBroadcast",  
  "etag": etag,  
  "id": string,  
  "snippet": {  
    "publishedAt": datetime,  
    "channelId": string,  
    "title": string,  
    "description": string,  
    "thumbnails": {  
      (key): {  
        "url": string,  
        "width": unsigned integer,  
        "height": unsigned integer  
      }  
    },  
    "scheduledStartTime": datetime,  
    "scheduledEndTime": datetime,  
    "actualStartTime": datetime,  
    "actualEndTime": datetime,  
    "isDefaultBroadcast": boolean,  
    "liveChatId": string  
  },  
  "status": {  
    "lifeCycleStatus": string,  
    "privacyStatus": string,  
    "recordingStatus": string,  
  },  
  "contentDetails": {  
    "boundStreamId": string,  
    "boundStreamLastUpdateTimeMs": datetime,  
    "monitorStream": {  
      "enableMonitorStream": boolean,  
      "broadcastStreamDelayMs": unsigned integer,  
      "embedHtml": string  
    },  
    "enableEmbed": boolean,  
    "enableDvr": boolean,  
    "enableContentEncryption": boolean,  
    "startWithSlate": boolean,  
    "recordFromStart": boolean,  
    "enableClosedCaptions": boolean,  
  },  
}
```

```

    "closedCaptionsType": string,
    "projection": string,
    "enableLowLatency": boolean
},
"statistics": {
    "totalChatCount": unsigned long
}
}

```

Para manipular estos datos el recurso nos proporciona varios métodos, aquí solo se explicarán algunos.

- **Insert**, este método crea un evento. Para ello debe recibir mínimo dos parámetros de entrada, el primero hace referencia a las partes del JSON que serán definidas(snippet, status, contentDetails....), mientras que el segundo argumento que recibe es el propio JSON con los datos. Se deben proporcionar obligatoriamente el título, la hora de inicio y el estado de privacidad del vídeo. Los demás datos si no están definidos tomarán un valor por defecto. Como respuesta a este método obtenemos el mismo JSON pero esta vez todos los campos con su valor asociado, ya sea el que le hemos asignado o uno por defecto. Dentro de las propiedades obtendremos el ID del recurso que será necesario en el método **BIND** para poder asociarlo a un flujo.
- **List**, este método retorna una lista con todos los datos de los *broadcast* pedidos en función de unos parámetros de entrada. Dentro de los parámetros obligatoriamente debemos especificar las propiedades del recurso que queremos recuperar. Adicionalmente podemos aplicar ciertos filtros a los resultados.
 - *broadcastStatus*, devuelve únicamente las emisiones que se encuentren en un estado determinado, activo, todos los estados, completado o sin comenzar
 - *id*, es el filtro más específico devuelve únicamente el recurso asociado a un identificador
 - *maxResults*, limita los resultados obtenidos, por defecto tiene un valor de cinco pero puede tomar valores entre cero y cincuenta.
 - *broadcastType*, con este filtro estipulamos que queremos obtener eventos de un tipo determinado siendo las opciones evento, una retransmisión programada a una hora determinada, persistente, un evento el cual se encuentra continuamente activo, u ambos.
- **Bind**, este método puede realizar dos acciones en función de los parámetros que reciba. Obligatoriamente debe recibir las partes del recurso, y un *id* perteneciente a un *broadcast*. Opcionalmente puede recibir un *id* que representa un recurso **livestream**, si recibe este parámetro el método enlazará ambos recursos quedando asignado a la emisión un flujo de vídeo, si por el contrario este parámetro no es proporcionado el método *bind* desenlazará de la emisión el flujo de vídeo si esta la tuviera.

- **Transition**, una vez creados y enlazados ambos recursos este método nos da la posibilidad de cambiar el estado de la emisión. Es decir podemos hacer pública la emisión, pasar a estado de test o dar por finalizada la emisión. Para ellos deberemos proporcionarle el estado el cual queremos dar a nuestra emisión, el *id* de dicha emisión y las partes del recurso que queremos obtener en la respuesta.

El segundo recurso que nos encontramos es `liveStream` que nos permite configurar las propiedades de la ingestión del vídeo. Los métodos de este recurso son similares a los métodos del `liveBroadcast` con la diferencia de que son asociados a un *stream* y sus propiedades se definen mediante un JSON que se puede ver a continuación.

```
{
  "kind": "youtube#liveStream",
  "etag": etag,
  "id": string,
  "snippet": {
    "publishedAt": datetime,
    "channelId": string,
    "title": string,
    "description": string,
    "isDefaultStream": boolean
  },
  "cdn": {
    "format": string,
    "ingestionType": string,
    "ingestionInfo": {
      "streamName": string,
      "ingestionAddress": string,
      "backupIngestionAddress": string
    },
    "resolution": string,
    "frameRate": string
  },
  "status": {
    "streamStatus": string,
    "healthStatus": {
      "status": string,
      "lastUpdateTimeSeconds": unsigned long,
      "configurationIssues": [
        {
          "type": string,
          "severity": string,
          "reason": string,
          "description": string
        }
      ]
    },
    "contentDetails": {
      "closedCaptionsIngestionUrl": string,
      "isReusable": boolean
    }
  }
}
```

El método `list` es prácticamente igual al explicado para `liveBroadcast` a diferencia que el resultado que obtenemos del mismo es una lista de *streams*. Respecto al

método *insert* su función es la de crear el recurso siendo las propiedades obligatorias a definir el título, el formato y el tipo de ingestión. Esta última propiedad, el tipo de ingestión, establece el protocolo por el cual se transmite el flujo de vídeo a YouTube. En este campo YouTube nos proporciona dos protocolos distintos DASH o RTMP.

3.3. FFmpeg

FFmpeg² es una colección de librerías de software libre capaz de decodificar, codificar, transcodificar, multiplexar, demultiplexar, filtrar y reproducir gran cantidad de archivos en múltiples formatos, también se encuentra disponible para distintos sistemas operativos como Linux, Mac OS X, Windows . . . para la mayoría de sus distribuciones.

FFmpeg presenta cuatro herramientas distintas:

- **ffmpeg**, herramienta que proporciona un rápida conversión de archivos de audio y vídeo a través de la línea de comandos.
- **ffserver**, formado por un servidor de *streaming* para audio y vídeo. Es capaz de soportar varios canales en vivo y *streaming* de archivos.
- **ffplay**, es un reproductor multimedia simple basado en SDL (Simple DirectMedia Layer) y las bibliotecas de ffmpeg.
- **ffprobe**, se encarga de reunir información de streams multimedia como formatos, bitrates, framerates . . .

En este proyecto únicamente se hará uso de ffmpeg como línea de comando ya que mediante esta herramienta podremos enviar flujo de vídeo al servidor de YouTube.

Otro punto importante a la hora de hablar de FFmpeg son las librerías:

- **libavutil**, constituye una biblioteca de apoyo de forma que se simplifica la programación incluyendo estructuras de datos, rutinas matemáticas, utilidades capa multimedia y otras muchas.
- **livavcodec**, esta librería está formada por codificadores y decodificadores de audio y vídeo.
- **libavformat**, es la parte encargada de la multiplexación y demultiplexacion para diferentes formatos multimedia.
- **libavdevice**, esta librería contiene herramientas de entrada y salida para grabar y renderizar el contenido multimedia generado por entornos como *Video4Linux*, *Vfm* o *ALSA*.
- **libavfilters**, proporciona filtros para contenido multimedia, filtros paso bajo, paso alto, compresores , bicuadrados . . .

²<https://ffmpeg.org/>

- **livwscale**, esta librería realiza operaciones altamente optimizadas de escalado de imagen y espacio de color.
- **libswresample**, es capaz de hacer muestreo y conversiones de formato.

Como podemos ver FFmpeg es una herramienta muy potente y que nos proporciona diversas opciones.

3.3.1. FFmpeg streaming

FFmpeg proporciona dos caminos para realizar *streaming* el primero consiste en enviar directamente el flujo de vídeo a un servidor, YouTube en nuestro caso, y este retransmitirlo nuevamente. La otra alternativa consiste en retransmitir directamente a un usuario final, incluso a través de la creación de múltiples salidas retransmitir hacia más de un usuario.

La herramienta ffmpeg a través de la línea de comandos nos permite enviar un flujo de vídeo al servidor de YouTube a través del protocolo RMTP, que es capaz de intercambiar un flujo multimedia entre un reproductor flash y un servidor. El formato del vídeo es *FLV (flash video player)* ya que es el *codec* de vídeo usado por YouTube.

El sistema operativo Linux, ofrece algunas herramientas que complementan a ffmpeg como puede ser *video4linux* , que consiste en un API de captura de vídeo para Linux capaz de capturar la imagen de una webcam. Por otro lado para capturas de audio, *ALSA (Advanced Linux Sound Architecture)* es una buena alternativa, que constituye un controlador de sonido del núcleo de Linux.

Combinando todas estas herramientas es posible enviar hacia los servidores de YouTube un flujo audiovisual.

3.4. Open Broadcaster Software

Esta aplicación conocida por sus siglas OBS³, constituye un software libre y de código abierto para la grabación y retransmisión (streaming) de vídeo. Para este proyecto se ha usado la versión 0.15.1 de OBS.

OBS comenzó como un pequeño proyecto creado por Hugh "Jim" Bailey, pero creció rápidamente con la ayuda de muchos colaboradores que trabajan para mejorar la aplicación. En 2014, comenzó a desarrollarse una nueva versión conocida como *OBS Multiplatform* (más tarde renombrada OBS Studio) para soporte multiplataforma, siendo un programa más completo y con una API más potente. *OBS Studio* es un trabajo en progreso ya que, a febrero de 2016, no ha alcanzado la paridad de características con el original de la OBS, es por eso que el original aun está disponible en el sitio.

Este proyecto usa como lenguaje de programación C y C++, permite capturar fuentes de vídeo en tiempo real, composición de escenas, codificación, grabación y

³<https://obsproject.com/>

retransmisión. Para la retransmisión de contenido en vivo usa RTMP como protocolo de transporte , de forma que es compatible con YouTube o Twitch entre otras plataformas.

Además la comunidad OBS ha desarrollado varios *plugins* que extienden sus funcionalidades como OBS remote que permite acceder remotamente a OBS a través de internet o *CLR Browser Source* que permite usar como fuente de vídeo un recurso web.

Este software permite realizar las mismas operaciones que ffmpeg pero de una forma mas sencilla debido a su interfaz gráfico, por lo que para usuarios menos expertos es una mejor alternativa que ffmpeg.



Figura 3.4: Interfaz OBS

3.5. JdeRobot

JdeRobot⁴ es un entorno de desarrollo para aplicaciones robóticas y de visión por computador. Este entorno, desarrollado por el grupo de robótica de la universidad Rey Juan Carlos, simplifica el acceso a los sensores, actuadores o robots como el drone. JdeRobot ha sido desarrollado en su gran mayoría usando como lenguaje de programación C++, basándose en un entorno de componentes distribuidos que pueden ser escritos en otros lenguajes como Java o Python. Finalmente estos componentes se comunican usando ICE.

ICE hace referencia a las siglas de *Internet Communications Engine* y consiste en un middleware orientado a objetos desarrollado por la empresa ZeroC⁵, que proporciona ayuda a la hora de desarrollar aplicaciones distribuidas ya que se encarga de todas las interacciones con los interfaces de red tales como abrir conexiones de red,

⁴<http://jderobot.org>

⁵<https://zeroc.com>

serializar y deserializar datos o reintentos de conexiones fallidas. Gracias a esto se puede crear conexiones entre distintas máquinas con distintos sistemas operativos o lenguajes de programación.

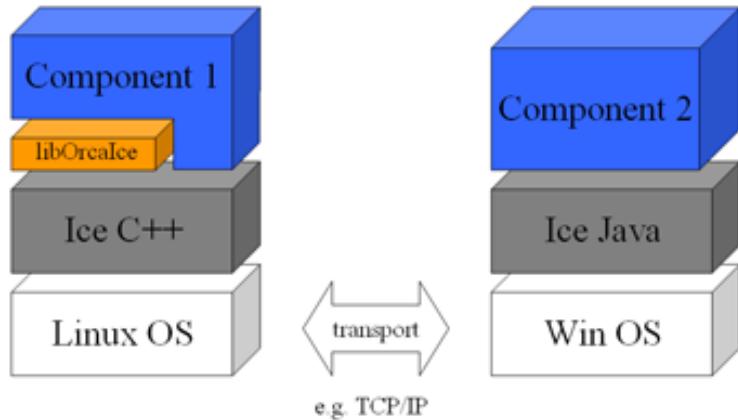


Figura 3.5: Estructura ICE

JdeRobot también usa una distribución de ICE, llamada ICEJs que nos da la opción de conectar navegadores usando JavaScript y los protocolos de ICE a través de *websockets*.

Recientemente JdeRobot ha lanzado su última versión estable JdeRobot 5.4.1. Esta versión aporta varios cambios que facilitan aún más el uso de sus herramientas. A continuación se comentan los cambios más significativos:

- Cambio de SO, la nueva versión de JdeRobot es soportada únicamente por Ubuntu 16.04.
- Cambio en la versión de Gazebo, la versión antigua trabajaba con la versión 5 de Gazebo, con los nuevos cambios se ha pasado a usar la versión 7.
- Actualización de la versión de ICE, este punto es importante ya que anteriormente JdeRobot trabajaba con la versión 3.5 de ICE en la cual no venía incorporado el *plugin* ICEJS lo que implicaba una mayor dificultad a la hora de instalar las librerías así como problemas de compatibilidad. En la nueva versión se ha pasado a usar la versión 3.6 de ICE donde si está incorporado el *plugin* citado.

JdeRobot ha desarrollado distintos componentes y *plugins*, en este proyecto se usarán dos de ellos principalmente *CameraServer* y *ArDroneServer*.

3.5.1. ArDroneServer

ArDroneServer es un componente cuya función es la comunicación a bajo nivel con un drone real. Este componente a través de seis interfaces ICE es capaz de recoger datos y valores provenientes de los sensores y enviar órdenes a los motores y actuadores del drone en función de las instrucciones de movimiento que le lleguen. Las interfaces son las siguientes.

- `cmd_vel`, interfaz para los comando de velocidad
- `navdata`, datos sensoriales como altitud o velocidades
- `ardrone_extra`, interfaz para funciones básicas y extra que presenta el drone como el aterrizaje, despegue, etc.
- `remoteConfig`, interfaz estándar para la transmisión de ficheros XML en Jde-Robot.
- `Pose3D`, esta interfaz recoge datos de posicionamiento en forma vectorial(x,y,z)

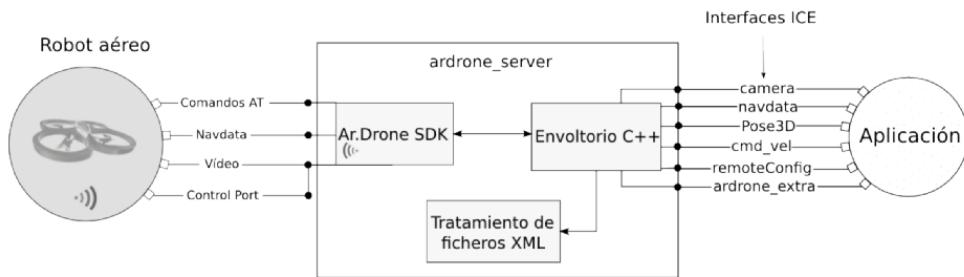


Figura 3.6: Arquitectura ArDroneServer

3.5.2. CameraServer

CameraServer es un componente desarrollado por JdeRobot capaz de servir N cámaras tanto reales como simuladas. Para el manejo de los vídeos, internamente usa *gstreamer*, entorno multimedia libre multiplataforma escrito en el lenguaje de programación C, permite crear aplicaciones audiovisuales, como de vídeo, sonido, codificación etc.

CameraServer permite captar el vídeo de diferentes fuentes como puede ser *video4linux*, un archivo o un recurso web.

CameraServer proporciona una interfaz, **Camera Interface**, esta interfaz aporta los métodos de configuración de la cámara así como los métodos de inicio y final de la toma de imágenes. Dicho interfaz extiende otro llamado **ImageProvider** el cual aporta métodos de configuración de la imagen, como el formato, y métodos que recuperan imágenes de forma cruda, sin comprimir, fotograma a fotograma y son entregados al interfaz *Camera*.

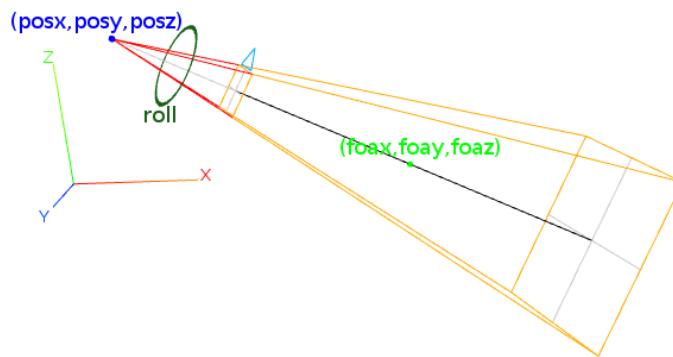


Figura 3.7: Modelo CameraServer

3.5.3. Plugins de Gazebo que incluyen Camera

JdeRobot ha desarrollado varios *plugins* usando Gazebo. Estos *plugins* facilitan la programación a más bajo nivel, como son las conexiones de forma que podemos manejar el drone en un mundo creado por Gazebo y depurar nuestro código.

Algunos de los *plugins* nos permite manejar drones con cámara integrada como es el caso de *Quadrotor2 Gazebo*⁶, el cual simula un quadrotor con una cámara incorporada de la cual se pueden recoger las imágenes fotograma a fotograma a través del interfaz Camera.

Otro ejemplo es *kobuki_driver*, donde se simula un *Turtlebot*, un pequeño robot con ruedas. El *plugin* de JdeRobot permite conducirlo a través de un mundo simulado a la vez que se accede a las imágenes captadas por una cámara incorporada a él.

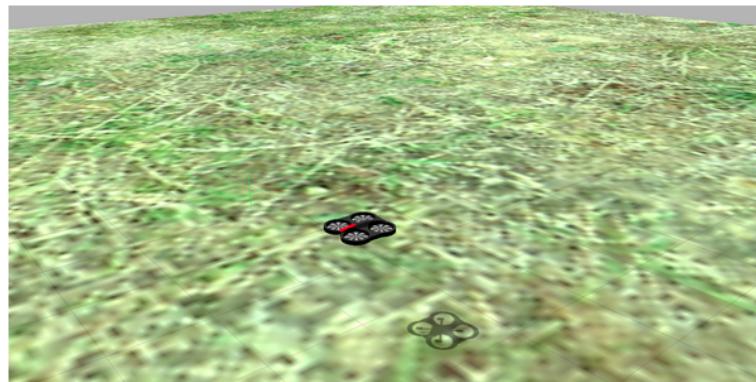


Figura 3.8: Escenario Quadrotor2 Gazebo

3.6. Gazebo

Gazebo es un simulador 3D, que permite crear escenarios en tres dimensiones en tu ordenador con robots, obstáculos y otros muchos objetos. Gazebo fue diseñado

⁶<http://jderobot.org/index.php/DriversQuadrotor2Gazebo>

para evaluar algoritmos y comportamientos de robots en un escenario simulado pero muy cercano a la realidad sin exponer a peligros a los robots. Es esencial testear las aplicaciones de forma que se pueden evitar fallos de batería, de manejo, localización o manejo entre otros. Es un proyecto de código abierto y actualmente cuenta con una gran comunidad de desarrolladores.



Figura 3.9: Escenario creado con gazebo

Gazebo ha sido útil para este proyecto a la hora de simular el comportamiento de un drone con una cámara a bordo que nos proporciona las imágenes necesarias para poder enviar el flujo multimedia a YouTube.

Capítulo 4

Aplicación web de difusión por YouTube

En este capítulo se describe la primera de las dos aplicaciones que se han diseñado, para cumplir los objetivos.

Esta primera aplicación web, maneja y retransmite contenido captado por una cámara local a eventos de YouTube. La aplicación incorpora las siguientes funcionalidades:

1. Permite crear eventos programados y asociados a un canal de YouTube
2. Permite iniciar y detener la retransmisión del evento a través de ffmpeg
3. Permite añadir subtítulos al evento retransmitido.
4. Muestra en su pantalla principal los eventos en directo asociados a un canal de YouTube.

4.1. Diseño

Antes de pasar a analizar el código con detalle se explica la arquitectura global con el objetivo de entender mejor su funcionamiento. La explicación se apoya en la figura 4.1:

a) **Comunicación con YouTube desde NodeJS:** para esta comunicación se usan las librerías de Google *YouTube Live Streaming Api* en su versión desarrollada en Python. En esta comunicación se produce el intercambio de datos entre los servidores de YouTube y nuestro servidor.

b) **Comunicación con ffmpeg desde NodeJS:** Ffmpeg es usado para retransmitir contenido audiovisual hacia los servidores de YouTube. Para el manejo de ffmpeg se ha implementado un script en Python que se encarga de ejecutar ffmpeg a través de la línea de comandos.

c) **Obtención del flujo de audiovisual:** Como fuente de vídeo y audio se usa una cámara local con un micrófono incorporado. El flujo de vídeo se recoge usando *video4linux*, una API de captura de vídeo integrada en Linux.

d) Comunicación con la interfaz del cliente: La aplicación presenta una interfaz para la interacción del cliente. Esta interfaz intercambia datos con el nuestro servidor a través de HTTP. Por otro lado para la visualización de vídeo el cliente tiene dos opciones o ir a la pagina web de YouTube donde se muestra el vídeo en emisión o visualizarlo en la propia interfaz de la aplicación web.

e) Comunicación ffmpeg con YouTube: Para que YouTube pueda comenzar su retransmisión, necesita recibir un flujo de vídeo en sus servidores. Ffmpeg es el encargado de enviar este flujo usando el protocolo *RTMP* (*Real Time Messaging Protocol*)

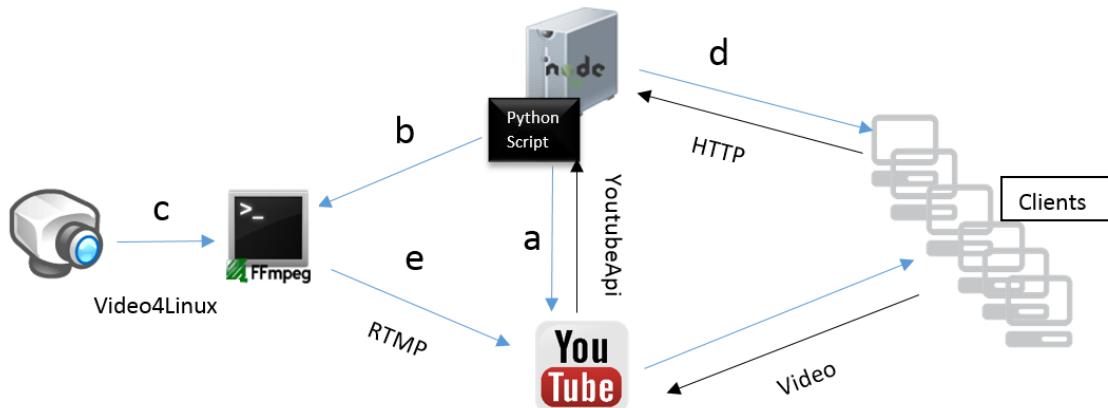


Figura 4.1: Arquitectura SurveillanceApp

4.2. Ejecución de Python en NodeJS

Una de las dificultades encontradas a la hora de desarrollar la aplicación es que las librerías proporcionadas por YouTube no presentan una versión estable para JavaScript. Como se ha comentado anteriormente las funcionalidades relacionadas con YouTube han sido desarrolladas con Python, como se podrá ver más adelante. Para manejar estos scripts se ha recurrido al módulo `child_process`, que forma parte de NodeJS. A través de este módulo NodeJS nos proporciona la opción de crear un proceso hijo de forma asíncrona, es decir sin bloquear el flujo principal del servidor. `Child_process` proporciona varios métodos para realizar esta tarea, en esta aplicación se usara el método `spawn`, que es capaz de generar un nuevo proceso a partir de un comando dado, en nuestro caso el comando será el de ejecución de ficheros Python. El método toma tres argumentos `child_process.spawn(command[, args][, options])`, el primero es el comando a ejecutar, el segundo un array con los argumentos del comando si este los tuviera y por ultimo opciones del comando. De esta forma se crea un proceso paralelo al servidor capaz que ejecuta el comando en un terminal del sistema sin bloquear al servidor. A continuación se incluye un fragmento de código donde se puede ver el uso del módulo dentro de la aplicación. En el podemos observar una primera sentencia donde se crea el proceso haciendo uso de `spawn`, ejecutando el

script de Python. Una vez generado el proceso este proporciona dos salidas `stdout` y `stderr`.

- `stdout`, recoge los datos devueltos por Python. De esta forma si desde Python queremos enviar datos al flujo principal debemos hacerlo mediante el método `print` y serán recogidos por `stdout`.
- `stderr`, aquí se recoge la salida de error tanto para errores de ejecución del comando como errores que puedan surgir dentro del script.

```
exports.retrieveStreamList = function(req,res,status){  
  
    var process = spawn('python',[ './public/python/streamList.py' , ' --  
        status' , status]);  
    processOutput(res,process);  
}  
  
function processOutput(res,process){  
  
    var py_data;  
    var py_err;  
  
    process.stdout.on("data",function(data){  
        py_data += data;  
    })  
  
    process.stdout.on("end",function(){  
  
        if(typeof py_data !== "undefined"){  
            py_data = py_data.substring(9);  
            console.log(py_data)  
            if(py_data.localeCompare("ERROR") == 1){  
                res.status(500).end();  
            }else{  
                res.end(py_data)  
            }  
        }  
    })  
  
    process.stderr.on("data", function(data){  
        py_err += data;  
    })  
    process.stderr.on("end",function(){  
  
        if(typeof py_err !== 'undefined'){  
            res.status(500).end()  
            console.log("Python Error: " + py_err)  
        }  
    })  
}
```

4.3. Servidor NodeJS: Diálogo con YouTube

Para esta comunicación se han usado las librerías proporcionadas por YouTube API. YouTube API proporciona muchas alternativas, para esta aplicación se han elegido dos recursos `livebroadcast` y `livestream` ambos explicados en el capítulo tres. Con estos recursos hemos desarrollado tres funcionalidades relacionadas con YouTube en la aplicación, separadas en tres scripts de Python.

4.3.1. Seguridad

En este fragmento de código aparecen varias cosas importantes para poder entender el funcionamiento de las librerías. Como se ha comentado en el capítulo tres YouTube ha adoptado Oauth 2.0 como sistema de autenticación. Para poder usar las librerías de YouTube primeramente debes registrarte en la consola de desarrolladores de Google y dar de alta tu proyecto. Una vez dado de alta debes habilitar las API que tu proyecto usará. Terminado este proceso obtendrás el secreto de cliente (`client_secrets.json`), es un fichero el cual contiene las credenciales que te identifican como propietario de la cuenta de Google¹ asociada al proyecto.

```
# The CLIENT_SECRETS_FILE variable specifies the name of a file
# that contains
# the OAuth 2.0 information for this application, including its
# client_id and
# client_secret.

CLIENT_SECRETS_FILE = "./private/client_secrets.json"

YOUTUBE_READ_WRITE_SCOPE = "https://www.googleapis.com/auth/
    youtube"
YOUTUBE_API_SERVICE_NAME = "youtube"
YOUTUBE_API_VERSION = "v3"

MISSING_CLIENT_SECRETS_MESSAGE = """
WARNING: Please configure OAuth 2.0

"""

def get_authenticated_service(args):

    flow = flow_from_clientsecrets(CLIENT_SECRETS_FILE,
        scope=YOUTUBE_READ_WRITE_SCOPE,
        message=MISSING_CLIENT_SECRETS_MESSAGE)

    storage = Storage("%s-oauth2.json" % sys.argv[0])
    credentials = storage.get()

    if credentials is None or credentials.invalid:
        credentials = run_flow(flow, storage, args)
```

¹<https://console.developers.google.com>

Otra parte importante es la de los ámbitos o *scope*. Esta parte especifica qué permisos tiene para interactuar con YouTube la aplicación. Por ejemplo, en el código escrito arriba el *scope* permite tanto leer como modificar los recursos asociados a la cuenta de YouTube. Los otros parámetros indican el API que va a ser usada y su versión.

A continuación nos encontramos con el método `get_authenticated_service`, que lleva a cabo el proceso de autenticación. Dicho método es proporcionado por Google en la documentación de la API². Este método busca un fichero con las credenciales que autorizan a la aplicación a acceder a los datos almacenados en la cuenta de YouTube asociada a la misma. Si es la primera vez que se accede estos datos no estarán generados por lo cual se redirigirá automáticamente al usuario a los servidores de autenticación de Google. Si la autenticación ha sido correcta se creará automáticamente un fichero que permite el acceso a la cuenta por parte de la aplicación.

4.3.2. Creación de eventos

Una de las funcionalidades que nos permite la aplicación es la creación de eventos programados. Para ellos a través de un formulario contenido en la interfaz web son recogidos una serie de datos como el título del evento, el horario, la privacidad o la calidad del mismo, todas ellas propiedades de ambos recursos. Estos datos son enviados al servidor en formato JSON y proporcionados como entrada a `createEvent.py`. `CreateEvent` hace uso del método `insert` tanto de `livebroadcast` como de `livestream` para crear ambos recursos. Una vez creados se recupera su ID para proporcionárselo al método `bind` perteneciente a `livebroadcast` de forma que el evento es creado en el canal asociado.

```
# Create a liveBroadcast resource and set configuration
def insert_broadcast(youtube, options):
    insert_broadcast_response = youtube.liveBroadcasts().insert(
        part="snippet,status,contentDetails",
        body=dict(snippet=dict(
            title= args["broadcast_title"],
            scheduledStartTime= args["start_time"]),
        status=dict(
            privacyStatus= args["privacy"]),
        contentDetails=dict(
            monitorStream=dict(
                enableMonitorStream = 'true'))
    )
)
).execute()

snippet = insert_broadcast_response["snippet"]
return insert_broadcast_response["id"]

# Create a liveStream resource and set configuration
def insert_stream(youtube, options):
    insert_stream_response = youtube.liveStreams().insert(
```

²<https://developers.google.com/youtube/v3/live/getting-started>

```

        part="snippet,cdn",
        body=dict(
            snippet=dict(
                title= args["broadcast_title"]
            ),
            cdn=dict(
                format= args["format"],
                ingestionType="rtmp"
            )
        )
    ).execute()

snippet = insert_stream_response["snippet"]
return insert_stream_response["id"]

# Bind the broadcast to the video stream.
def bind_broadcast(youtube, broadcast_id, stream_id):
    bind_broadcast_response = youtube.liveBroadcasts().bind(
        part="id,contentDetails",
        id=broadcast_id,
        streamId=stream_id
    ).execute()

```

4.3.3. Recuperación de información de evento

Dentro de la aplicación en su pantalla principal podemos ver el vídeo de aquellos eventos que se encuentran en directo sin necesidad de acudir a la web de YouTube directamente, esto es posible gracias a que YouTube mediante su propiedad `monitorStream` nos proporciona un i-frame el cual tiene como fuente(`source`) un reproductor de YouTube embebido asociado a nuestro evento.

Por otro lado dentro de la parte privada de la aplicación, únicamente accesible para administradores, se pueden ver los eventos programados e iniciar su retransmisión o los eventos que se están retransmitiendo para detenerlos.

Todas estas funcionalidades requieren de datos proporcionados por los servidores de YouTube. Para recuperar estos datos se recurre al método `list`, perteneciente tanto al recurso `livebroadcast` como al `livestream`. Dicho método admite filtros para buscar los eventos que nos interesan. Se filtra por el estado del evento, activo o programado, y además especificamos las partes del recurso que queremos recuperar por último se limitan los resultados a 50.

```

list_broadcasts_request = youtube.liveBroadcasts().list(
    broadcastStatus= status,
    part="snippet, contentDetails",
    maxResults=50
)

```

En el caso de `livestream` recuperamos únicamente el flujo asociado a la retransmisión mediante su ID, de este recurso nos interesa su propiedad *CDN* ya que de ahí obtenemos el *stream_name*, necesario para la posterior retransmisión ya que este ID

identifica al recurso ³ en los servidores de ingestión de contenido de YouTube, además recuperamos los datos de ingestión como la calidad del evento de forma que luego se pueden calcular datos como bitrate o resolución. Otras propiedades importantes que recuperamos son *monitor* que contiene un *iframe* que posteriormente se inserta en la aplicación web para poder visualizar el evento una vez activo. Una vez recuperados son almacenados en *list_broadcast_request*. Este objeto contiene la información de todos los eventos y de él recuperamos la información que nos interesa de cada uno y formamos un JSON que será la respuesta a la petición web.

```

while list_broadcasts_request:
    broadcast_response = list_broadcasts_request.execute()
    for broadcast in broadcast_response.get("items", []):
        title = broadcast["snippet"]["title"]
        monitorStream = broadcast["contentDetails"]["monitorStream"][
            "embedHtml"]
        stream_id = broadcast["contentDetails"]["boundStreamId"]
        cdn = getStreamKey(youtube, stream_id)
        data_output["data"].append({"title": title, "streamkey": cdn[
            "ingestionInfo"]["streamName"],
            "monitor": monitorStream, "quality": cdn["format"], "broadcastID": broadcast["id"]})

    list_broadcasts_request = youtube.liveBroadcasts().list_next(
        list_broadcasts_request, broadcast_response)

# Retrieve a livestream resource match with stream_id
def getStreamKey(youtube, stream_id):

    list_streams_request = youtube.liveStreams().list(
        part="cdn",
        id=stream_id,
        maxResults=1
    )
    list_streams_response = list_streams_request.execute()
    return list_streams_response["items"][0]["cdn"]

```

4.3.4. Terminar la retransmisión del evento

Para conseguir este objetivo se ha usado el método *transition* perteneciente al recurso *livebroadcast*. Con dicho método recuperamos una emisión en directo a partir de su ID y modificamos su estado de activo a finalizado, de esta forma YouTube dará por terminada la retransmisión y el evento finalizará. También debemos acabar con la ejecución de ffmpeg ya que mientras este se este ejecutando no podremos iniciar la retransmisión de un nuevo evento por que la cámara estará ocupada por ffmpeg , para ello usamos el comando *-pkill ffmpeg*, el cual acabará con el proceso.

³Capítulo 3. Propiedades *livestream* y *livebroadcast*

```
def stop_broadcast(youtube, brID):
    broadcast_request = youtube.liveBroadcasts().transition(
        broadcastStatus= "complete",
        id = brID,
        part = "status")
    broadcast_request.execute()
```

4.4. Servidor NodeJS: Comunicación con Ffmpeg

Ffmpeg es el encargado de enviar el flujo audiovisual a los servidores de ingestión de contenido de YouTube. Ffmpeg presenta varias herramientas pero para este objetivo únicamente hemos usado la funcionalidad que permite manejarlo desde la terminal.

Para ello se ha desarrollado un script de Python que escribe en un fichero .sh el comando de ffmpeg a ejecutar, tras esto mediante la instrucción `os.system` se ejecuta el fichero. Para la construcción del comando ffmpeg, se toma como entrada primeramente la calidad del evento, en función de esta calidad se añade el `bitrate` y la resolución del vídeo según las especificaciones de YouTube. Otro parámetro fundamental que recibe el script es el `stream_key`, que representa un identificador único del `stream` dentro de los servidores de YouTube para que éste pueda asociar el flujo audiovisual recibido al evento.

```
def list_streams(stream_key, resolution, bitrate):

    os.system("chmod +rw ./public/static/ffmpeg.sh")
    outfile = open('./public/static/ffmpeg.sh', 'w')
    outfile.write('ffmpeg -f alsa -ac 2 -i default -f video4linux2
                  + -framerate 15 -video_size ' + resolution +
                  '-i dev/video0 -vcodec libx264 -preset veryfast -minrate ' +
                  bitrate + '-maxrate 1000k -bufsize 1000k' + '-vf "format=
                  yuv420p" -g 30' + '-vf drawtext= "fontfile=/usr/share/fonts
                  /truetype/freefont/FreeSerif.ttf: + fontsize =24:'
                  'fontcolor=yellow:textfile=./public/static/subtitles.txt:reload
                  =1 :x=100:y=50"' +
                  '-c:a libmp3lame -b:a 128k -ar 44100' +
                  '-force_key_frames 0:0:04 f flv rtmp://a.rtmp.youtube.com/
                  live2/' + Stream_key
    outfile.close()
    os.system("chmod 0755 ./public/static/ffmpeg.sh")
    os.system("./public/static/ffmpeg.sh")
```

Ffmpeg presenta una gran variedad de posibilidades a continuación se explican las principales partes del comando usado en el script:

- **Video4Linux y ALSA**, son componentes pertenecientes al SO Linux que se encargan de recoger los flujos de vídeo y audio respectivamente. Como fuente de vídeo usamos la cámara principal del sistema, `dev/video0`, para el audio el micro definido por defecto.

- **Libx264**, *codec* de vídeo H264, tras él aparecen las opciones de codificación del vídeo, el perfil no está especificado pero por defecto ffmpeg usa el perfil básico del *codec*.
- **Drawtext**, opción usada para superponer texto sobre el vídeo, es la equivalencia a los subtítulos. El texto es leído de un fichero .txt, que tiene asociada la opción *reload* encargada de detectar el cambio en el fichero de texto. Las otras opciones lo acompañan se refieren al formato del texto mostrado en pantalla.
- **Force_key_frames**, a la hora de codificar el vídeo se usan *frames* de referencia a partir de los cuales se codifican los siguientes, este comando obliga a enviar estos fotogramas/marcos cada cuatro segundos.
- **RTMP**, protocolo de comunicación usado entre el servidor de YouTube y ffmpeg para el intercambio del contenido.

4.5. Dialogo HTTP e interfaz de cliente

La aplicación cuenta con un interfaz para interactuar con canales de YouTube, pudiendo realizar todas las operaciones descritas anteriormente.

4.5.1. Seguridad

La aplicación tiene acceso a datos privados y acciones de la cuenta de YouTube del usuario por lo que es necesario implementar mecanismos de seguridad para que solo el usuario tenga acceso a ellos. Por ello la aplicación tiene una parte pública y otra privada solo accesible por el administrador.

Como mecanismo de seguridad se ha usado un middleware, implementado en el servidor. El *middleware*, intercepta las peticiones HTTP que dan acceso a la parte privada, donde se encuentran las operaciones de crear, iniciar o detener eventos.

```
//rutas donde se compureba si el usuario tiene permiso
server.use("/createbroadcast",security.middleware)
server.use("/streams",security.middleware)
```

El *middleware* comprueba si el usuario ha iniciado una sesión previamente, en cuyo caso se permitirá el acceso a la dirección requerida. En caso contrario sera redirigido a un formulario de *login*, donde podrá iniciar sesión.

Por último una vez iniciada la sesión se da la opción al usuario de cerrarla.

```
var sess;

exports.middleware = function(req,res,next){
  sess = req.session
  if(sess.auth){
    next(); //if log ok ==> continue
  } else{
```

```

        console.log("Incorrect User")
        res.redirect("/login")

    }

exports.login = function(req,res,session,config){

    if(req.body.user == config.admin.name && req.body.password ==
       config.admin.password){
        sess = req.session;
        console.log("Right User : session started")
        sess.auth = "true";
        res.end("Right user")
    } else{
        res.status(400).end("User or Password Error");
    }
}

exports.logout = function(req,res){
    req.session.destroy();
    res.redirect("/login")
}

```

4.5.2. Diálogo HTTP

La comunicación con el servidor NodeJS se lleva a cabo a través del protocolo HTTP. Esta comunicación ha sido desarrollada con AngularJS, *framework* del lenguaje JavaScript. Principalmente la comunicación HTTP se encarga de transmitir los datos introducidos por el usuario en el interfaz al servidor, que posteriormente se los enviará a YouTube.

Angular usa los llamados controladores para manejar los datos de la aplicación, por ello se ha desarrollado un controlador por cada pantalla que se encarga de recoger y enviar los datos al servidor y mostrar su respuesta en caso de que fuera necesario.

A continuación se muestran dos controladores el primero de ellos es el encargado de mostrar las retransmisiones en directo, el segundo muestra la lista de eventos programados y que pueden empezar su retransmisión.

```

var app = angular.module('app', []);

//index.html controller
app.controller("indexController", ['$scope', '$http', '$sce',function
    ($scope,$http,$sce){
    $scope.setNavbar = "static/navbar.html"

    $scope.videos = {};
    //Add iframe element to index.html
    $scope.addVideo = function(video){
        if(video.length > 0){
            //Extract iframe src

```

```

        src = video.substring(video.indexOf('embed') + 6 ,video.
            indexOf('?'))
        return $sce.trustAsResourceUrl("https://www.youtube.com/embed
            /" + src)
    }
}

$scope.getVideoList = function(){
    $http({
        method:'GET',
        url : '/videolist'
    }).success(function(response){
        if(response.data.length == 0){
            $scope.msg = "No avaialbles events now"
        }else{
            $scope.videos = response.data;
        }
    }).error(function (response){
        $scope.msg = "Can't show the videos"
    })
}
$scope.getVideoList();
})
])

```

```

//stream_list.html controller
app.controller("dataListController", ['$scope', '$http', '$window',
    function($scope,$http,$window){
        $scope.events = {}
        $scope.getList = function(){
            $http({
                method: "GET",
                url: "/streamlist"
            }).success(function(response){
                $scope.events = response;
            }).error(function(response){
                $scope.msg = "Can't show video list"
            })
        }

        $scope.startStream = function(streamkey,quality){
            dataSend = {"streamkey" : streamkey,"quality": quality}
            $http({
                method: "POST",
                url: "/startstreaming",
                data: dataSend,
                headers : {'Content-Type': 'application/JSON'}
            }).success(function (response){
                console.log(response)
                $window.location.href = '/subtitles';
            }).error(function(response){
                $scope.msg = "Can't stop the event"
            })
        }
    }
])

```

```

    }
    $scope.getList();
  ])
)
}

```

4.5.3. Interfaz

La aplicación implementa una sencilla interfaz gráfica desde la cual se puede interactuar con YouTube. Esta interfaz que consta de dos partes, una pública y otra privada.

Usa angular para introducir o recibir los datos en las comunicaciones HTTP gracias al elemento *scope*, un contenedor que almacena el modelo de datos del controlador.

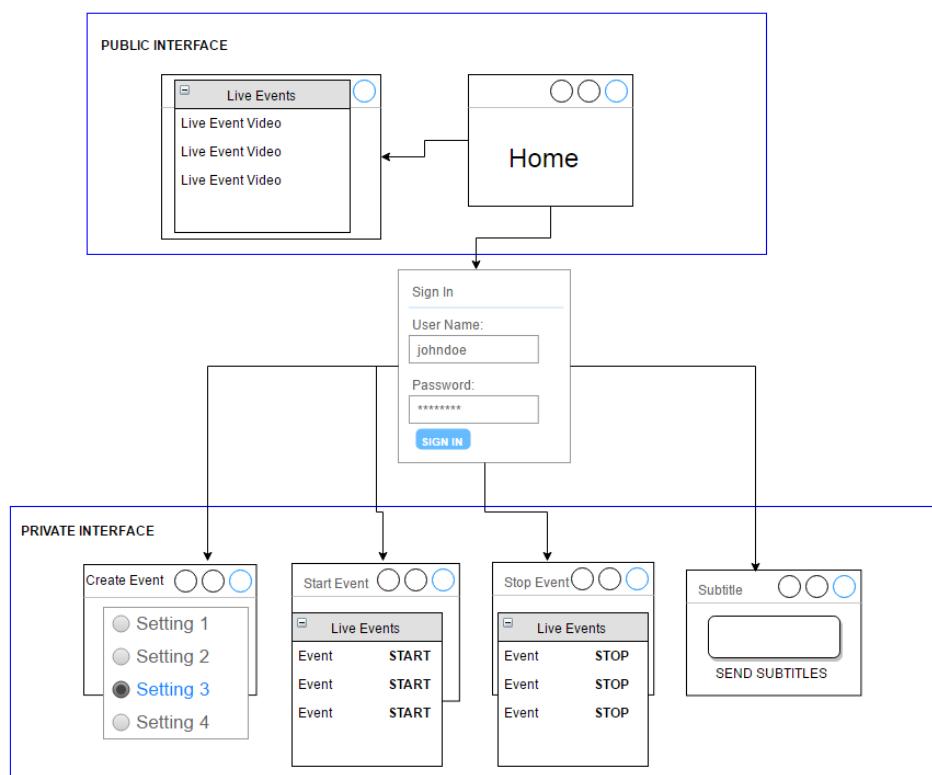


Figura 4.2: Diagrama de la Aplicación

La primera parte, es una parte pública, donde se pueden reproducir los videos de los eventos en directo, usando el elemento de video embebido.

```

<!doctype html>
<html ng-app="app">
  <head>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.3/angular.min.js"></script>
    <script src="//ajax.googleapis.com/ajax/libs/angularjs/1.2.19/angular-cookies.js"></script>
  
```

```
<link rel="stylesheet" type="text/css" href="static/css/
  indexStyle.css">
<script src="app.js"></script>
</head>
<body ng-controller = "indexController">
  <div ng-include = setNavbar ></div>
  
  <div class="container">
    <table>
      <tr>
        <th>Title</th>
        <th> Video</th>
      </tr>
      <tr ng-repeat="video in videos">
        <td>{{video.title}}</td>
        <td><iframe ng-src="{{addVideo(video.monitor)}}"></iframe></td>
      </tr>
    </table>
    <p class="error">{{msg}}</p>
  </div>
</body>
</html>
```

La segunda parte es privada, esta protegida por el middleware comentado anteriormente. Una vez superada la seguridad, desde el interfaz se pueden crear eventos seleccionando la hora de inicio y calidad del evento, se muestra también una lista de todos los eventos programados, junto con la opción de empezar estos eventos. Si dicha opción es seleccionada ffmpeg comenzara la retransmisión de vídeo, y desde el interfaz se ofrecerá la posibilidad de insertar subtítulos en la emisión. Para ello el contenido enviado al servidor es escrito en un fichero de texto que sera leído por ffmpeg. La última funcionalidad implementada es la de dar por finalizado el evento, de forma que aparecerá una lista con los eventos en directo del canal en ese momento dando opción a finalizarlos.

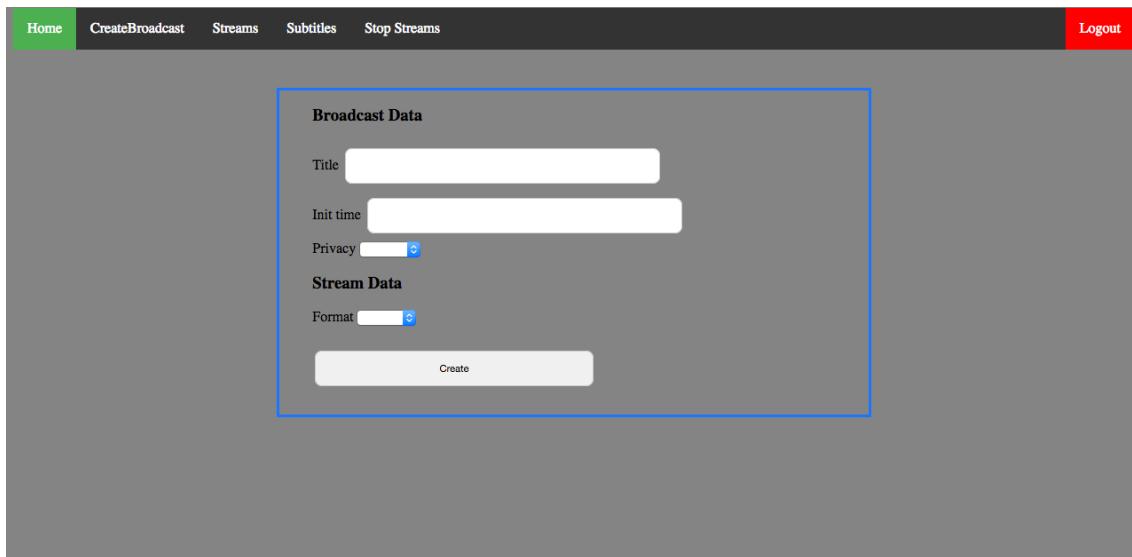


Figura 4.3: Interfaz Creación de Eventos

4.6. Experimentos

Para probar el correcto funcionamiento de la aplicación se ha ejecutado el servidor NodeJS una IP pública. Se ha conectado una cámara web al equipo que capta el flujo de vídeo, que posteriormente es enviado a YouTube. Ffmpeg ha sido configurado para que retransmita el flujo audiovisual a 25fps, suficiente para mostrar un vídeo fluido.

Desde un ordenador remoto, fuera de la red del servidor, se ha accedido a la dirección en la que está ejecutándose el servidor. Una vez conectado se ha creado, iniciado y finalizado eventos de YouTube satisfactoriamente a través de la aplicación web, así como se ha probado la funcionalidad.

En estos experimentos, se han configurado distintas calidades de imagen para el evento. Se ha podido observar que la aplicación no sufre retardos aunque la calidad de la imagen aumente, aunque esta calidad es relativa a la cámara usada para la captación de imágenes.

Con estos resultados se puede decir que la aplicación funciona correctamente⁴.

⁴<http://jderobot.org/Apavo-tfgSurveillanceApp>

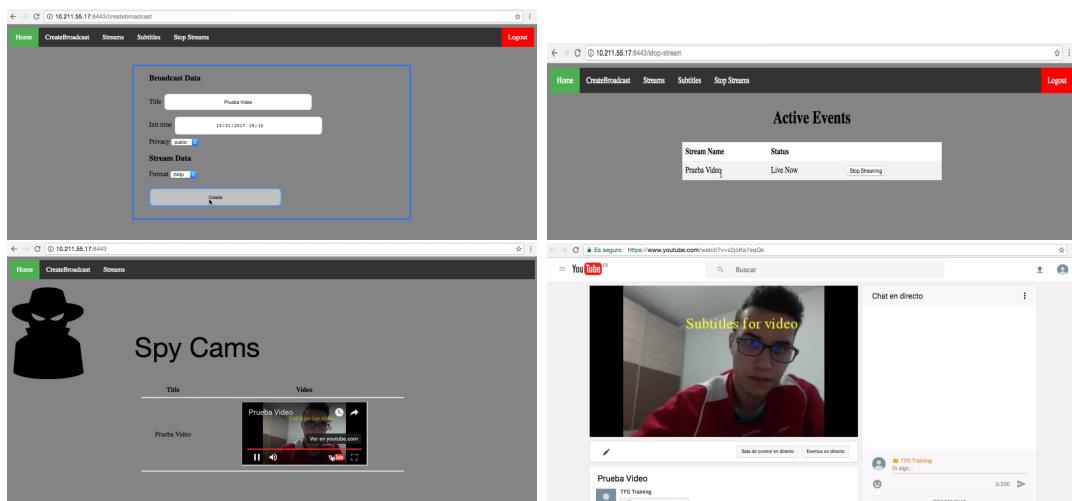


Figura 4.4: Experimento

Capítulo 5

Streaming web con YouTube desde un Drone

Esta segunda aplicación web al igual que la explicada en el capítulo anterior retransmite contenido audiovisual desde YouTube con la diferencia de que el flujo de vídeo esta vez es captado desde la cámara de un drone.

5.1. Diseño

Para esta aplicación se ha reutilizado parte del código desarrollado en el capítulo cuatro. La interfaz usada por los clientes es la misma que en la aplicación anterior, con la diferencia de que la pantalla de creación de eventos desaparece y la de inicio y finalización de eventos se ha unificado. La principal diferencia en el diseño radica en el modo de obtención del flujo de vídeo. Para captar el contenido visual del drone, la aplicación ha tenido que compatibilizarse con las distintas fuentes de vídeo de la plataforma de JdeRobot. Para tal objetivo se ha desarrollado un componente denominado *ffmpegAdapter*, que recoge los fotogramas proporcionados por JdeRobot.

- a) **Comunicación del adaptador con NodeJS:** Una vez el cliente envía al servidor la petición de que se inicie el evento, este ejecuta el adaptador.
- b) **Comunicación del adaptador con ffmpeg:** El adaptador proporciona fotogramas a ffmpeg, que se encarga de generar y enviar el flujo de vídeo para los servidores de YouTube.
- c) **Comunicación del adaptador con JdeRobot:** Para esta comunicación se ha usado ICE, que se encarga de conectar nuestro adaptador a las herramientas de vídeo de JdeRobot. Dichas herramientas envían fotograma a fotograma, al adaptador las imágenes captadas por el drone.

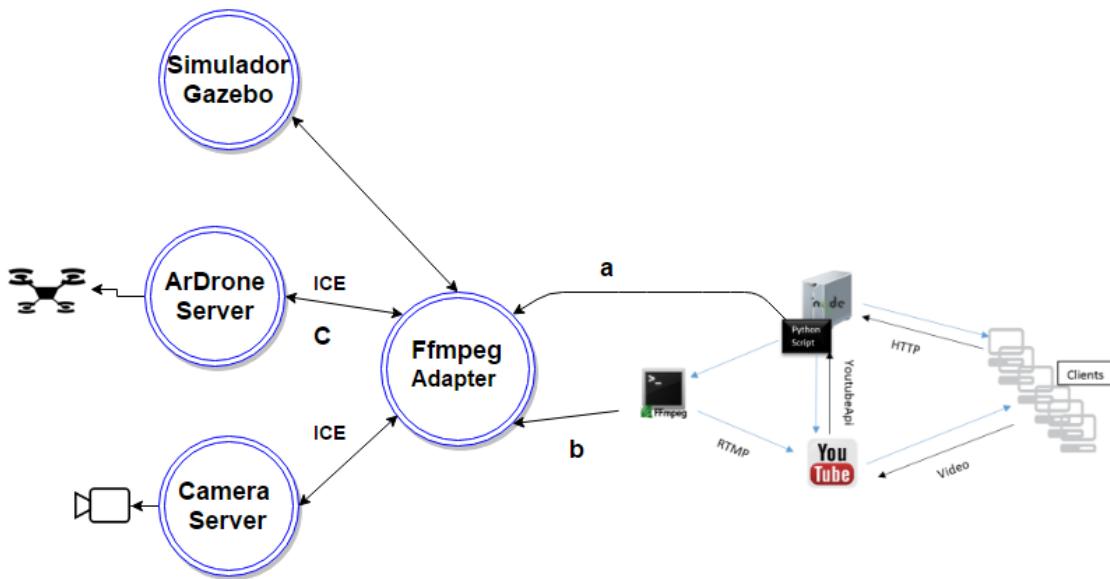


Figura 5.1: Arquitectura Aplicación

5.2. Adaptador ffmpeg a JdeRobot

Tanto *CameraServer* como *ArdroneServer*, herramientas de vídeo de JdeRobot, proporcionan un flujo de imágenes fotograma a fotograma a través de ICE. YouTube no acepta esto sino que debe recibir un flujo de vídeo por *RTMP*.

FFmpeg presenta una opción que permite dadas como fuente de entrada una o varias imágenes, retornar un flujo vídeo. En este caso para no sobrecargar el sistema con archivos, se ha optado por usar como entrada una sola imagen sobreescrita a medida que se reciben nuevos fotogramas. Sobre esta imagen se aplica un bucle infinito con el objetivo de que ffmpeg la use como fuente de entrada. Para entender mejor este proceso se muestra un ejemplo del comando usado.

```
ffmpeg -loop 1 -i image.jpg -v:c libx264 -pix_fmt yuv420p output.mp4
```

Gracias a la opción `-loop 1` se establece un bucle infinito que toma como entrada una imagen, de forma que si esta imagen varía con una frecuencia suficiente se obtendrá un vídeo fluido.

El siguiente paso es encontrar una herramienta de JdeRobot capaz de proporcionar imágenes con una frecuencia suficiente como para poder crear un flujo de vídeo. Para ello se han usado los componentes *cameraserver* y *ArDroneServer*, explicados en el capítulo tres. Dichos componentes captan imágenes a una velocidad de unos 25 fps, suficiente para el propósito. Los componentes están desarrollados en C++, pero pueden interoperar con Python, lenguaje en el que se ha desarrollado el componente.

Una vez encontrado los componentes, la idea es enviar las imágenes capturadas en un UAV a un ordenador local, donde se encontrara el adaptador. Una vez en el adaptador, las imágenes son almacenadas localmente en un solo archivo, de forma que

cada vez que una nueva imagen es recibida la antigua se sobrescribe. Por último ffmpeg accede a esta imagen "dinámica" para formar el flujo de vídeo final.

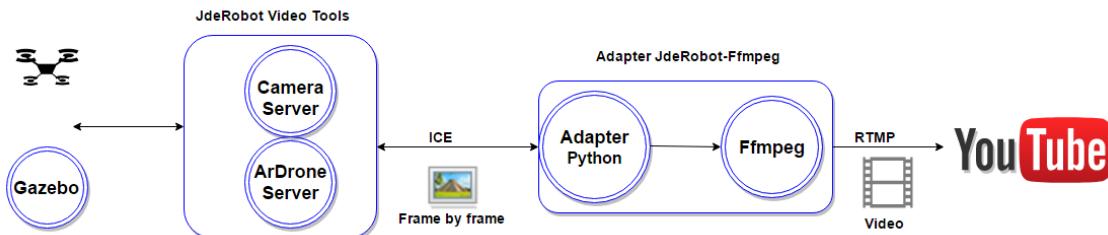


Figura 5.2: Esquema adaptador JdeRobot a ffmpeg

Conexión mediante ICE

JdeRobot incluye distintos componentes que usan ICE para conectarse entre ellos. Para facilitar la conexión ICE en Python, JdeRobot ha desarrollado una serie de librerías que facilitan este proceso. Dentro de este proyecto se hace uso de `easyiceconfig.py`¹ y `parallelIce.py`². `Easyiceconfig` es el encargado de obtener los parámetros necesarios para inicializar la conexión usando la librerías de ICE pertenecientes a ZeroC.

El segundo `parallelICE`, consiste en un modulo de Python que permite recibir la conexión con los interfaicer ICE del dron (*CameraClient, cmdvel, navDataClient, pose3D*). Para nuestro objetivo únicamente necesitaremos conectarlos con la cámara por lo que usaremos `cameraClient.py`.

Este script tiene implementada dos clases `camera` y `cameraClient`. La función de `cameraClient` es crear un hilo de ejecución que inicializa un objeto de clase `camera`, dicho objeto debe recibir como parámetros la conexión ICE inicializada previamente y el nombre del interfaz ICE, con estos datos se establece la conexión con la cámara del dron. Además de esto la clase `camera` tiene implementados métodos que nos devuelven tanto las imágenes del dron como información acerca de ellas, por lo que una vez instanciado el objeto simplemente deberemos llamar a su método `getImage`. Una vez entendidas las conexiones y las librerías usadas se muestra el desarrollo del código.

Como base para desarrollar este adaptador se ha usado como ejemplo de referencia el código de la herramienta `CameraClient.py` y el de la herramienta `colorfilter.py`³, desarrollados en Python.

Para realizar la comunicación entre el ordenador local, donde ejecutamos el adaptador, y la herramientas de JdeRobot se usará una conexión ICE, haciendo uso de las librerías desarrolladas por JdeRobot `easyiceconfig` y `parallelICE`.

```

import sys
import easyiceconfig as EasyIce
from gui.threadGUI import ThreadGUI

```

¹<https://github.com/JdeRobot/JdeRobot/tree/master/src/libs/easyiceconfig.py>

²<https://github.com/JdeRobot/JdeRobot/tree/master/src/libs/parallelIce.py>

³http://jderobot.org/Teaching robotics with JdeRobotColor_filter

```

from parallelIce.cameraClient import CameraClient
from gui.cameraWidget import CameraWidget
from PyQt5.QtWidgets import QApplication

if __name__ == '__main__':
    ic = EasyIce.initialize(sys.argv)
    prop = ic.getProperties()
    remoteCamera = CameraClient(ic, "Introrob.Camera", True)
    app = QApplication(sys.argv)
    camera = CameraWidget()
    camera.setCamera(remoteCamera)
    if (len(sys.argv) == 3 and sys.argv[2] == "GUI"):
        camera.setGUI = True
        camera.initUI()
        camera.show()
    else:
        print("For see the GUI, add to command GUI")

```

Los datos usados para esta conexión son recuperados de un fichero de configuración en el que se especifican la dirección y puerto en la que se esta ejecutando el componente al que se ha de conectar el adaptador.

```
Introrob.Camera.Proxy = cameraA:default -h localhost -p 9999
```

Procesamiento y almacenamiento imágenes: Doble Buffer

Para solucionar el almacenamiento de las imágenes en un ordenador local se ha usado la librería *PIL* de Python, librería que se encarga del procesado de imágenes. Las herramientas de JdeRobot devuelven fotograma a fotograma imágenes en formato *RGB* por lo que son recibidas como un *array* de datos por el adaptador, dicho formato es incompatible con *ffmpeg*, por lo que a través del modulo *image*, perteneciente a la librería *PIL*, se transforma dicho *array* en una imagen, para ser posteriormente almacenada localmente en formato *JPG*, compatible con *ffmpeg*. En este punto nos encontramos con un problema y es que los procesos del adaptador y de *ffmpeg* no están sincronizados, es decir a la vez que *ffmpeg* lee la imagen el adaptador esta escribiendo en ella por lo cual se produce un error. Para este error se proporcionan dos soluciones

- La primera solución implica sustituir como codificador a *ffmpeg* por *OBS*, ya que *OBS* no accede a la imagen si esta está siendo modificada.
- La segunda solución es compatible tanto con *ffmpeg* con *OBS* y consiste en usar la técnica conocida como *doble buffer*. Dicha técnica consiste en usar dos archivos, uno de ellos sera un archivo temporal usado únicamente por el adaptador para escribir datos en él, tras finalizar la escritura se genera una copia del archivo con un nombre distinto que sera usado únicamente para su lectura por *ffmpeg* u *OBS*.

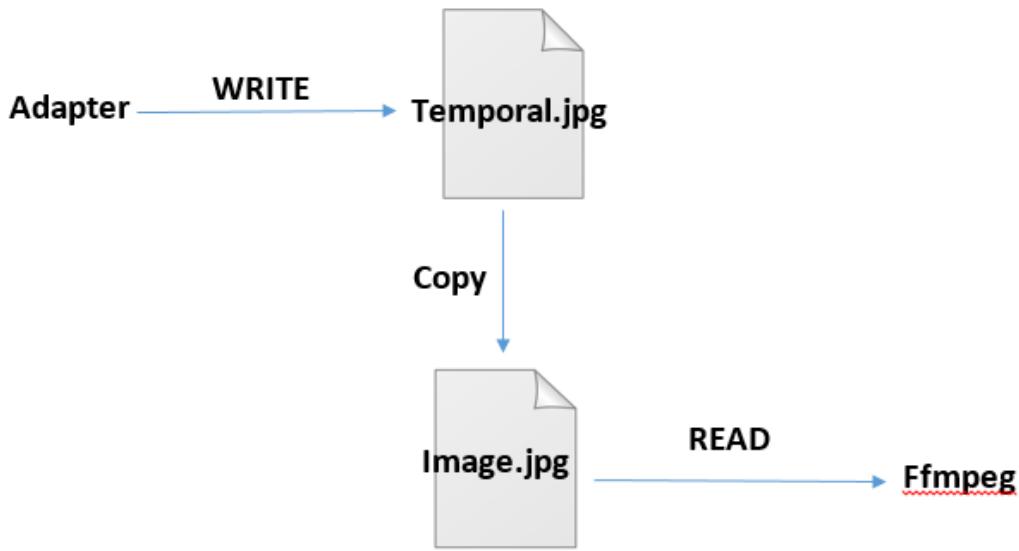


Figura 5.3: Técnica de doble buffer

En el adaptador finalmente se ha elegido la segunda opción, que es compatible con ambos codificadores. A continuación se puede observar el código mediante el cual las imágenes son obtenidas y almacenadas localmente.

```

def updateImage(self):
    img = self.getCamera().getImage()
    if img is not None:
        im = Image.fromarray(img)
        im.save("temp.jpg")
        os.rename("temp.jpg", "imagen.jpg")
        if self.GUI:
            showGUI(im)
  
```

Este método pertenece a la clase `CameraWidget`, dicha clase posee un atributo denominado `camera`, en la instancia de la clase a este atributo se le asigna el valor de la `cámara` obtenida de `JdeRobot`, que a su vez implementa el método `getImage` que devuelve un *array* con los datos de la imagen que posteriormente es convertido a una imagen.

5.3. Comunicacion NodeJS

El servidor NodeJS, se comunica con Python y YouTube de la misma forma que en la aplicación del capítulo anterior.

La novedad en el servidor es que se encarga de ejecutar el adaptador una vez la orden de iniciar la retransmisión es dada por el cliente. Al inicializar el adaptador

también se lanza ffmpeg. Ambos procesos deben ejecutarse en paralelo, ya que las imágenes usadas en el comando ffmpeg son dadas por el adaptador. Se ha recurrido a ejecutar un hilo que se encargue del adaptador mientras que el hilo principal se encarga de ejecutar ffmpeg. A continuación se muestra el código desarrollado para tal propósito.

```
def list_streams(path, stream_key, resolution, bitrate):
    command = 'ffmpeg -f alsa -ac 2 -i default -f image2 -framerate 15 -loop 1 -i ' + path + ' -vcodec libx264 -preset veryfast -minrate ' + bitrate + ' -maxrate 1000k -bufsize 1000k -vf "format=yuv420p" -g 30 -vf drawtext="fontfile=/usr/share/fonts/truetype/freefont/FreeSerif.ttf:fontsize=24:fontcolor=yellow:textfile=./public/static/subtitles.txt:reload=1:x=100:y=50" -c:a libmp3lame -b:a 128k -ar 44100 -force_key_frames 0:00:04 -f flv rtmp://a.rtmp.youtube.com/live2/' + stream_key
    os.system(command)

if __name__ == "__main__":
    try:
        subProcess = Popen(['python3', './public/JdeRobot/ffmpegAdapter/ffmpeg-adapter.py', '--Ice.Config=./public/JdeRobot/ffmpegAdapter/adapter_conf.cfg'])
        list_streams(sys.argv[1], sys.argv[2], sys.argv[3], sys.argv[4])
    except:
        print("ERROR")
```

5.4. Experimentos

Todas estas herramientas han sido desarrolladas con el objetivo final de ejecutarlas junto con un UAV real por ello una vez desarrollados todos los componentes se pasa a la fase de experimentación. Esta fase a su vez se divide en dos partes la simulación y la prueba real.

Experimentos con la cámara de un drone simulado

Debido a que los UAV son aparatos costosos no se puede experimentar con ellos directamente sin antes pasar por un periodo de simulación. De dicha simulación se encarga Gazebo, del que ya hablamos en el capítulo de introducción. Actualmente disponemos de distintos escenarios de simulación compatibles con JdeRobot. Se usa *UAViewer* para teleoperar el Drone.

Los escenarios de Gazebo simulan un UAV que posee los cuatro interfaces que manejas en JdeRobot proporcionados por el *ardroneServer*(todas estas herramientas están explicadas en el capítulo 3), para nuestro adaptador únicamente necesitamos la interfaz Camera pero para el manejo del Drone si necesitaremos los demás interfaces.

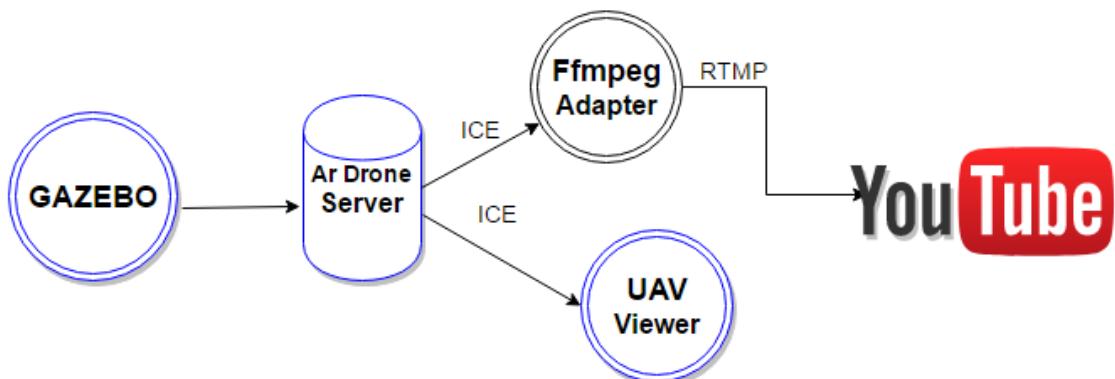


Figura 5.4: Arquitectura Experimento

Para conectar Gazebo con el adaptador y poder recuperar las imágenes proporcionadas por la cámara, únicamente se debe configurar el fichero `adapter.conf.cfg`, con la IP y el puerto correcto donde Gazebo se está ejecutando, una vez conectado a través de ffmpeg, OBS o la aplicación web se comienza la retransmisión hacia YouTube del contenido captado por el UAV. La ejecución de este experimento puede verse en la wiki oficial del proyecto⁴.

Experimentos con Drone real

Para el experimento se usará el modelo *Ar Drone* fabricado por la marca *Parrot* y proporcionado por el equipo de robótica de la URJC, dicho dron es un cuadricóptero totalmente compatible con las herramientas de JdeRobot.



Figura 5.5: arDrone de Parrot

Para realizar las pruebas nos apoyaremos en el *ardrone server* como forma de conexión con el aparato y en *UAVViewer* para el manejo del dron. Por otro lado para nuestro adaptador y posterior comunicación con YouTube únicamente recuperaremos el interfaz de la cámara, como en el experimento del simulador.

⁴http://jderobot.org/Apavo-tfgYouTube_26JdeRobot

ArDrone server posee un fichero de configuración que podemos ver a continuación, en él se encuentran detallados en qué dirección IP y puerto se está recibiendo información de cada uno de los interfaces así como los nombres que toman estos.

```
ArDrone.Camera.Endpoints=default -h 0.0.0.0 -p 9999
ArDrone.Camera.Name=ardrone_camera
ArDrone.Camera.FramerateN=15
ArDrone.Camera.FramerateD=1
ArDrone.Camera.Format=RGB8
ArDrone.Camera.ArDrone2.ImageWidth=640
ArDrone.Camera.ArDrone2.ImageHeight=360
ArDrone.Camera.ArDrone1.ImageWidth=320
ArDrone.Camera.ArDrone1.ImageHeight=240
# If you want a mirror image, set to 1
ArDrone.Camera.Mirror=0

ArDrone.Pose3D.Endpoints=default -h 0.0.0.0 -p 9998
ArDrone.Pose3D.Name=ardrone_pose3d

ArDrone.RemoteConfig.Endpoints=default -h 0.0.0.0 -p 9997
ArDrone.RemoteConfig.Name=ardrone_remoteConfig

ArDrone.Navdata.Endpoints=default -h 0.0.0.0 -p 9996
ArDrone.Navdata.Name=ardrone_navdata

ArDrone.CMDVel.Endpoints=default -h 0.0.0.0 -p 9995
ArDrone.CMDVel.Name=ardrone_cmdvel

ArDrone.Extra.Endpoints=default -h 0.0.0.0 -p 9994
ArDrone.Extra.Name=ardrone_extra

ArDrone.NavdataGPS.Endpoints=default -h 0.0.0.0 -p 9993
ArDrone.NavdataGPS.Name=ardrone_navdatagps
```

Tanto para conectar UAVViewer como nuestro adaptador a *arDroneServer* necesitamos editar los ficheros de configuración de forma que las IPs, puertos y nombres de los interfaces de los que deseamos recuperar información coincidan. A continuación podemos ver la configuración del adaptador para poder recuperar imágenes de la cámara.

```
Introrob.Camera.Proxy = ardrone_camera:default -h 0.0.0.0 -p 9999
```

En la realización del experimento nos encontramos con un problema de red no previsto. El drone crea una red wifi a la cual debemos conectar nuestro PC para establecer una comunicación con él a través de las herramientas desarrolladas. Dicha red no tiene conexión a internet, por lo cual aunque la conexión con el drone es satisfactoria la comunicación con YouTube no se puede llevar a cabo, ya que requiere de conexión a internet.

La solución es crear dentro del mismo PC dos conexiones de red, una conexión wifi para interactuar con el dron y la otra conexión vía *ethernet* con acceso a internet.

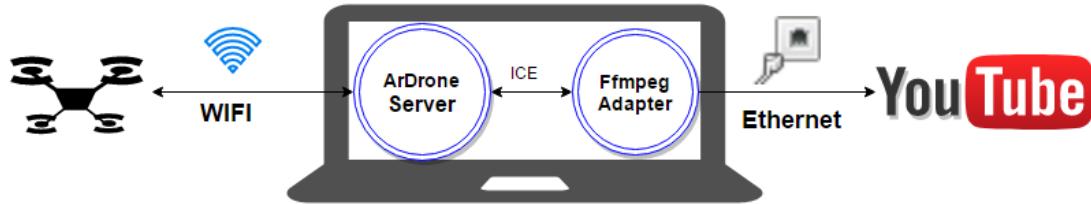


Figura 5.6: Arquitectura Experimento

Debido a que JdeRobot se ejecuta bajo el sistema operativo Linux, para conseguir esta configuración de red hemos accedido manualmente a la configuración de red del sistema aportado una dirección pública vía *ethernet* suministrada por la universidad en la que levantaremos el servidor NodeJS de la aplicación mientras que en la segunda red estableceremos una conexión vía wifi con el UAV.

Una vez realizadas todas las conexiones el experimento se ha llevado a cabo con éxito, este experimento puede verse al completo en la wiki oficial del proyecto⁵ o en el canal de YouTube⁶.

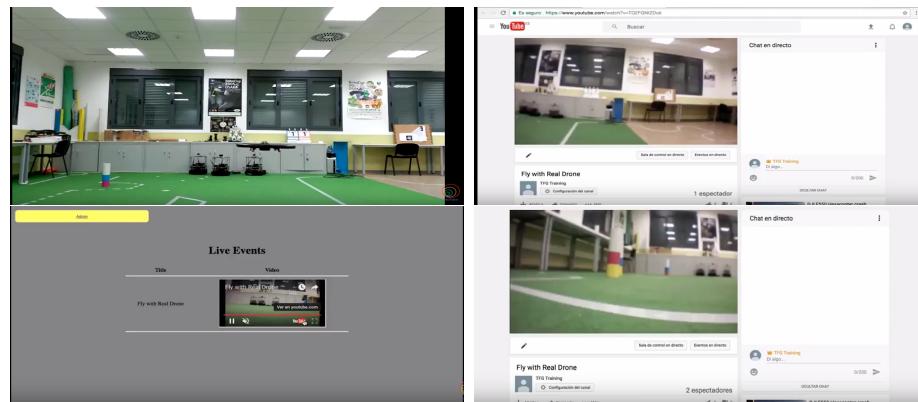


Figura 5.7: Experimento

⁵<http://jderobot.org/Apavo-tfgFlywithRealDrone>

⁶<https://www.youtube.com/watch?v=jo67tP62-Uw>

Capítulo 6

Streaming web con YouTube desde un Drone

Esta segunda aplicación web al igual que la explicada en el capítulo anterior retransmite contenido audiovisual desde YouTube con la diferencia de que el flujo de vídeo esta vez es captado desde la cámara de un drone.

6.1. Diseño

Para esta aplicación se ha reutilizado parte del código desarrollado en el capítulo cuatro. La interfaz usada por los clientes es la misma que en la aplicación anterior, con la diferencia de que la pantalla de creación de eventos desaparece y la de inicio y finalización de eventos se ha unificado. La principal diferencia en el diseño radica en el modo de obtención del flujo de vídeo. Para captar el contenido visual del drone, la aplicación ha tenido que compatibilizarse con las distintas fuentes de vídeo de la plataforma de JdeRobot. Para tal objetivo se ha desarrollado un componente denominado *ffmpegAdapter*, que recoge los fotogramas proporcionados por JdeRobot.

- a) **Comunicación del adaptador con NodeJS:** Una vez el cliente envía al servidor la petición de que se inicie el evento, este ejecuta el adaptador.
- b) **Comunicación del adaptador con ffmpeg:** El adaptador proporciona fotogramas a ffmpeg, que se encarga de generar y enviar el flujo de vídeo para los servidores de YouTube.
- c) **Comunicación del adaptador con JdeRobot:** Para esta comunicación se ha usado ICE, que se encarga de conectar nuestro adaptador a las herramientas de vídeo de JdeRobot. Dichas herramientas envían fotograma a fotograma, al adaptador las imágenes captadas por el drone.

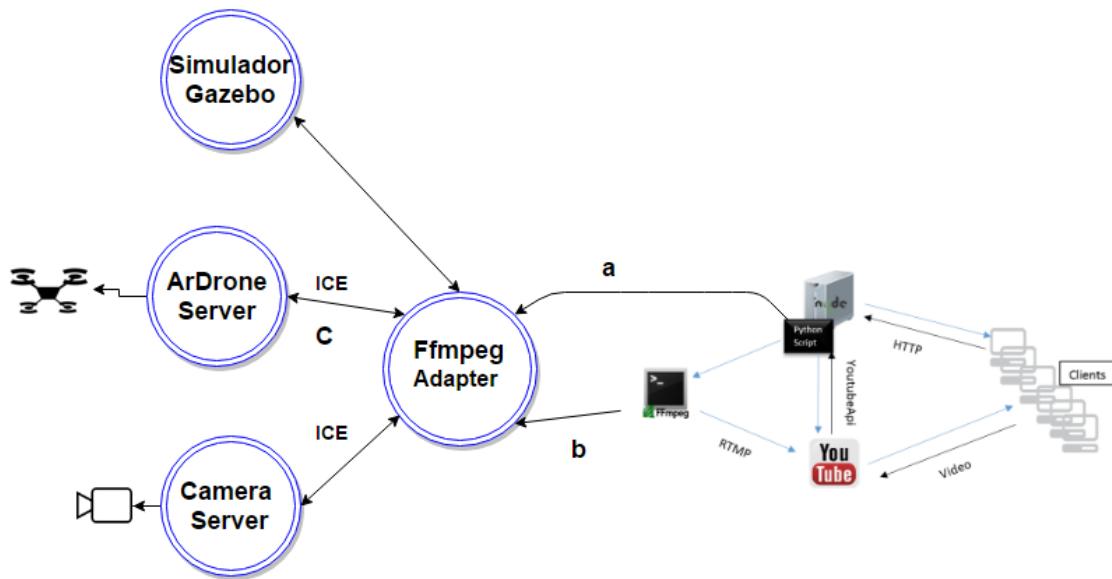


Figura 6.1: Arquitectura Aplicación

6.2. Adaptador ffmpeg a JdeRobot

Tanto *CameraServer* como *ArdroneServer*, herramientas de vídeo de JdeRobot, proporcionan un flujo de imágenes fotograma a fotograma a través de ICE. YouTube no acepta esto sino que debe recibir un flujo de vídeo por *RTMP*.

FFmpeg presenta una opción que permite dadas como fuente de entrada una o varias imágenes, retornar un flujo vídeo. En este caso para no sobrecargar el sistema con archivos, se ha optado por usar como entrada una sola imagen sobreescrita a medida que se reciben nuevos fotogramas. Sobre esta imagen se aplica un bucle infinito con el objetivo de que ffmpeg la use como fuente de entrada. Para entender mejor este proceso se muestra un ejemplo del comando usado.

```
ffmpeg -loop 1 -i image.jpg -v:c libx264 -pix_fmt yuv420p output.mp4
```

Gracias a la opción `-loop 1` se establece un bucle infinito que toma como entrada una imagen, de forma que si esta imagen varía con una frecuencia suficiente se obtendrá un vídeo fluido.

El siguiente paso es encontrar una herramienta de JdeRobot capaz de proporcionar imágenes con una frecuencia suficiente como para poder crear un flujo de vídeo. Para ello se han usado los componentes *cameraserver* y *ArDroneServer*, explicados en el capítulo tres. Dichos componentes captan imágenes a una velocidad de unos 25 fps, suficiente para el propósito. Los componentes están desarrollados en C++, pero pueden interoperar con Python, lenguaje en el que se ha desarrollado el componente.

Una vez encontrado los componentes, la idea es enviar las imágenes capturadas en un UAV a un ordenador local, donde se encontrara el adaptador. Una vez en el adaptador, las imágenes son almacenadas localmente en un solo archivo, de forma que

cada vez que una nueva imagen es recibida la antigua se sobrescribe. Por último ffmpeg accede a esta imagen "dinámica" para formar el flujo de vídeo final.

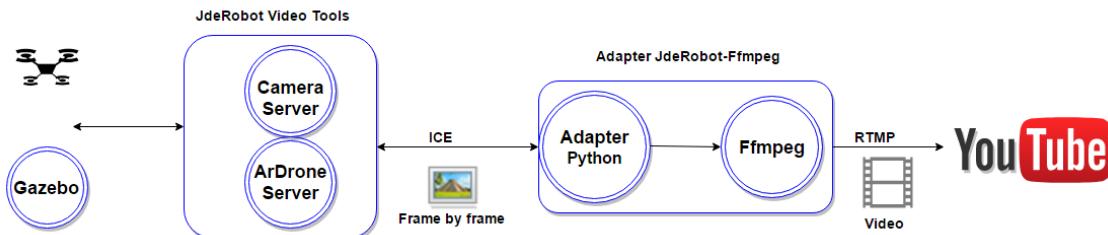


Figura 6.2: Esquema adaptador JdeRobot a ffmpeg

Conexión mediante ICE

JdeRobot incluye distintos componentes que usan ICE para conectarse entre ellos. Para facilitar la conexión ICE en Python, JdeRobot ha desarrollado una serie de librerías que facilitan este proceso. Dentro de este proyecto se hace uso de `easyiceconfig.py`¹ y `parallelIce.py`². `Easyiceconfig` es el encargado de obtener los parámetros necesarios para inicializar la conexión usando la librerías de ICE pertenecientes a ZeroC.

El segundo `parallelICE`, consiste en un modulo de Python que permite recibir la conexión con los interfaccer ICE del dron (*CameraClient, cmdvel, navDataClient, pose3D*). Para nuestro objetivo únicamente necesitaremos conectarlos con la cámara por lo que usaremos `cameraClient.py`.

Este script tiene implementada dos clases `camera` y `cameraClient`. La función de `cameraClient` es crear un hilo de ejecución que inicializa un objeto de clase `camera`, dicho objeto debe recibir como parámetros la conexión ICE inicializada previamente y el nombre del interfaz ICE, con estos datos se establece la conexión con la cámara del dron. Además de esto la clase `camera` tiene implementados métodos que nos devuelven tanto las imágenes del dron como información acerca de ellas, por lo que una vez instanciado el objeto simplemente deberemos llamar a su método `getImage`. Una vez entendidas las conexiones y las librerías usadas se muestra el desarrollo del código.

Como base para desarrollar este adaptador se ha usado como ejemplo de referencia el código de la herramienta `CameraClient.py` y el de la herramienta `colorfilter.py`³, desarrollados en Python.

Para realizar la comunicación entre el ordenador local, donde ejecutamos el adaptador, y la herramientas de JdeRobot se usará una conexión ICE, haciendo uso de las librerías desarrolladas por JdeRobot `easyiceconfig` y `parallelICE`.

```

import sys
import easyiceconfig as EasyIce
from gui.threadGUI import ThreadGUI

```

¹<https://github.com/JdeRobot/JdeRobot/tree/master/src/libs/easyiceconfig.py>

²<https://github.com/JdeRobot/JdeRobot/tree/master/src/libs/parallelIce.py>

³http://jderobot.org/Teaching robotics with JdeRobotColor_filter

```

from parallelIce.cameraClient import CameraClient
from gui.cameraWidget import CameraWidget
from PyQt5.QtWidgets import QApplication

if __name__ == '__main__':
    ic = EasyIce.initialize(sys.argv)
    prop = ic.getProperties()
    remoteCamera = CameraClient(ic, "Introrob.Camera", True)
    app = QApplication(sys.argv)
    camera = CameraWidget()
    camera.setCamera(remoteCamera)
    if (len(sys.argv) == 3 and sys.argv[2] == "GUI"):
        camera.setGUI = True
        camera.initUI()
        camera.show()
    else:
        print("For see the GUI, add to command GUI")

```

Los datos usados para esta conexión son recuperados de un fichero de configuración en el que se especifican la dirección y puerto en la que se esta ejecutando el componente al que se ha de conectar el adaptador.

```
Introrob.Camera.Proxy = cameraA:default -h localhost -p 9999
```

Procesamiento y almacenamiento imágenes: Doble Buffer

Para solucionar el almacenamiento de las imágenes en un ordenador local se ha usado la librería *PIL* de Python, librería que se encarga del procesado de imágenes. Las herramientas de JdeRobot devuelven fotograma a fotograma imágenes en formato *RGB* por lo que son recibidas como un *array* de datos por el adaptador, dicho formato es incompatible con *ffmpeg*, por lo que a través del modulo *image*, perteneciente a la librería *PIL*, se transforma dicho *array* en una imagen, para ser posteriormente almacenada localmente en formato *JPG*, compatible con *ffmpeg*. En este punto nos encontramos con un problema y es que los procesos del adaptador y de *ffmpeg* no están sincronizados, es decir a la vez que *ffmpeg* lee la imagen el adaptador esta escribiendo en ella por lo cual se produce un error. Para este error se proporcionan dos soluciones

- La primera solución implica sustituir como codificador a *ffmpeg* por *OBS*, ya que *OBS* no accede a la imagen si esta está siendo modificada.
- La segunda solución es compatible tanto con *ffmpeg* con *OBS* y consiste en usar la técnica conocida como *doble buffer*. Dicha técnica consiste en usar dos archivos, uno de ellos sera un archivo temporal usado únicamente por el adaptador para escribir datos en él, tras finalizar la escritura se genera una copia del archivo con un nombre distinto que sera usado únicamente para su lectura por *ffmpeg* u *OBS*.

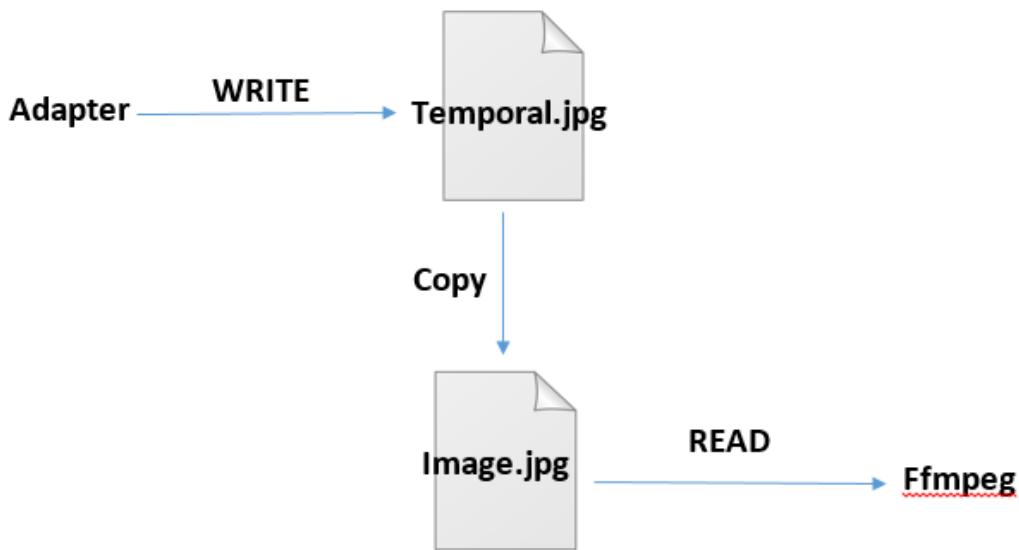


Figura 6.3: Técnica de doble buffer

En el adaptador finalmente se ha elegido la segunda opción, que es compatible con ambos codificadores. A continuación se puede observar el código mediante el cual las imágenes son obtenidas y almacenadas localmente.

```

def updateImage(self):
    img = self.getCamera().getImage()
    if img is not None:
        im = Image.fromarray(img)
        im.save("temp.jpg")
        os.rename("temp.jpg", "imagen.jpg")
        if self.GUI:
            showGUI(im)
  
```

Este método pertenece a la clase `CameraWidget`, dicha clase posee un atributo denominado `camera`, en la instancia de la clase a este atributo se le asigna el valor de la `cámara` obtenida de `JdeRobot`, que a su vez implementa el método `getImage` que devuelve un *array* con los datos de la imagen que posteriormente es convertido a una imagen.

6.3. Comunicacion NodeJS

El servidor NodeJS, se comunica con Python y YouTube de la misma forma que en la aplicación del capítulo anterior.

La novedad en el servidor es que se encarga de ejecutar el adaptador una vez la orden de iniciar la retransmisión es dada por el cliente. Al inicializar el adaptador

también se lanza ffmpeg. Ambos procesos deben ejecutarse en paralelo, ya que las imágenes usadas en el comando ffmpeg son dadas por el adaptador. Se ha recurrido a ejecutar un hilo que se encargue del adaptador mientras que el hilo principal se encarga de ejecutar ffmpeg. A continuación se muestra el código desarrollado para tal propósito.

```
def list_streams(path, stream_key, resolution, bitrate):
    command = 'ffmpeg -f alsa -ac 2 -i default -f image2 -framerate 15 -loop 1 -i ' + path + ' -vcodec libx264 -preset veryfast -minrate ' + bitrate + ' -maxrate 1000k -bufsize 1000k -vf "format=yuv420p" -g 30 -vf drawtext="fontfile=/usr/share/fonts/truetype/freefont/FreeSerif.ttf:fontsize=24:fontcolor=yellow:textfile=./public/static/subtitles.txt:reload=1:x=100:y=50" -c:a libmp3lame -b:a 128k -ar 44100 -force_key_frames 0:00:04 -f flv rtmp://a.rtmp.youtube.com/live2/' + stream_key
    os.system(command)

if __name__ == "__main__":
    try:
        subProcess = Popen(['python3', './public/JdeRobot/ffmpegAdapter/ffmpeg-adapter.py', '--Ice.Config=./public/JdeRobot/ffmpegAdapter/adapter_conf.cfg'])
        list_streams(sys.argv[1], sys.argv[2], sys.argv[3], sys.argv[4])
    except:
        print("ERROR")
```

6.4. Experimentos

Todas estas herramientas han sido desarrolladas con el objetivo final de ejecutarlas junto con un UAV real por ello una vez desarrollados todos los componentes se pasa a la fase de experimentación. Esta fase a su vez se divide en dos partes la simulación y la prueba real.

Experimentos con la cámara de un drone simulado

Debido a que los UAV son aparatos costosos no se puede experimentar con ellos directamente sin antes pasar por un periodo de simulación. De dicha simulación se encarga Gazebo, del que ya hablamos en el capítulo de introducción. Actualmente disponemos de distintos escenarios de simulación compatibles con JdeRobot. Se usa *UAViewer* para teleoperar el Drone.

Los escenarios de Gazebo simulan un UAV que posee los cuatro interfaces que manejas en JdeRobot proporcionados por el *ardroneServer*(todas estas herramientas están explicadas en el capítulo 3), para nuestro adaptador únicamente necesitamos la interfaz Camera pero para el manejo del Drone si necesitaremos los demás interfaces.

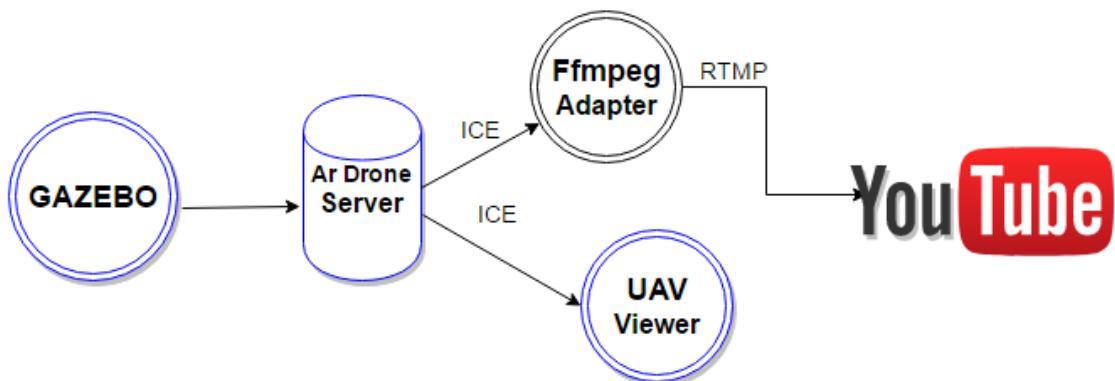


Figura 6.4: Arquitectura Experimento

Para conectar Gazebo con el adaptador y poder recuperar las imágenes proporcionadas por la cámara, únicamente se debe configurar el fichero `adapter.conf.cfg`, con la IP y el puerto correcto donde Gazebo se está ejecutando, una vez conectado a través de ffmpeg, OBS o la aplicación web se comienza la retransmisión hacia YouTube del contenido captado por el UAV. La ejecución de este experimento puede verse en la wiki oficial del proyecto⁴.

Experimentos con Drone real

Para el experimento se usará el modelo *Ar Drone* fabricado por la marca *Parrot* y proporcionado por el equipo de robótica de la URJC, dicho dron es un cuadricóptero totalmente compatible con las herramientas de JdeRobot.



Figura 6.5: arDrone de Parrot

Para realizar las pruebas nos apoyaremos en el *ardrone server* como forma de conexión con el aparato y en *UAViewer* para el manejo del dron. Por otro lado para nuestro adaptador y posterior comunicación con YouTube únicamente recuperaremos el interfaz de la cámara, como en el experimento del simulador.

⁴http://jderobot.org/Apavo-tfgYouTube_26JdeRobot

ArDrone server posee un fichero de configuración que podemos ver a continuación, en él se encuentran detallados en qué dirección IP y puerto se está recibiendo información de cada uno de los interfaces así como los nombres que toman estos.

```
ArDrone.Camera.Endpoints=default -h 0.0.0.0 -p 9999
ArDrone.Camera.Name=ardrone_camera
ArDrone.Camera.FramerateN=15
ArDrone.Camera.FramerateD=1
ArDrone.Camera.Format=RGB8
ArDrone.Camera.ArDrone2.ImageWidth=640
ArDrone.Camera.ArDrone2.ImageHeight=360
ArDrone.Camera.ArDrone1.ImageWidth=320
ArDrone.Camera.ArDrone1.ImageHeight=240
# If you want a mirror image, set to 1
ArDrone.Camera.Mirror=0

ArDrone.Pose3D.Endpoints=default -h 0.0.0.0 -p 9998
ArDrone.Pose3D.Name=ardrone_pose3d

ArDrone.RemoteConfig.Endpoints=default -h 0.0.0.0 -p 9997
ArDrone.RemoteConfig.Name=ardrone_remoteConfig

ArDrone.Navdata.Endpoints=default -h 0.0.0.0 -p 9996
ArDrone.Navdata.Name=ardrone_navdata

ArDrone.CMDVel.Endpoints=default -h 0.0.0.0 -p 9995
ArDrone.CMDVel.Name=ardrone_cmdvel

ArDrone.Extra.Endpoints=default -h 0.0.0.0 -p 9994
ArDrone.Extra.Name=ardrone_extra

ArDrone.NavdataGPS.Endpoints=default -h 0.0.0.0 -p 9993
ArDrone.NavdataGPS.Name=ardrone_navdatagps
```

Tanto para conectar UAVViewer como nuestro adaptador a *arDroneServer* necesitamos editar los ficheros de configuración de forma que las IPs, puertos y nombres de los interfaces de los que deseamos recuperar información coincidan. A continuación podemos ver la configuración del adaptador para poder recuperar imágenes de la cámara.

```
Introrob.Camera.Proxy = ardrone_camera:default -h 0.0.0.0 -p 9999
```

En la realización del experimento nos encontramos con un problema de red no previsto. El dron crea una red wifi a la cual debemos conectar nuestro PC para establecer una comunicación con él a través de las herramientas desarrolladas. Dicha red no tiene conexión a internet, por lo cual aunque la conexión con el dron es satisfactoria la comunicación con YouTube no se puede llevar a cabo, ya que requiere de conexión a internet.

La solución es crear dentro del mismo PC dos conexiones de red, una conexión wifi para interactuar con el dron y la otra conexión vía *ethernet* con acceso a internet.

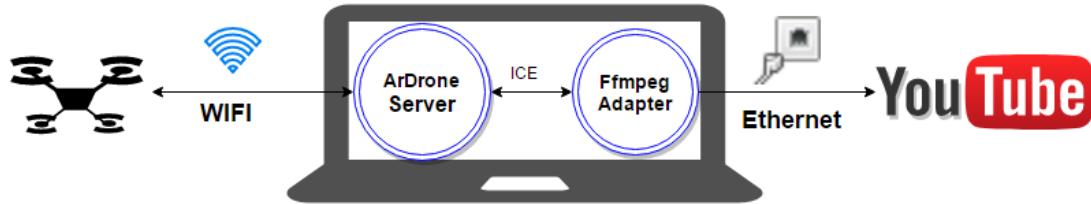


Figura 6.6: Arquitectura Experimento

Debido a que JdeRobot se ejecuta bajo el sistema operativo Linux, para conseguir esta configuración de red hemos accedido manualmente a la configuración de red del sistema aportado una dirección pública vía *ethernet* suministrada por la universidad en la que levantaremos el servidor NodeJS de la aplicación mientras que en la segunda red estableceremos una conexión vía wifi con el UAV.

Una vez realizadas todas las conexiones el experimento se ha llevado a cabo con éxito, este experimento puede verse al completo en la wiki oficial del proyecto⁵ o en el canal de YouTube⁶.

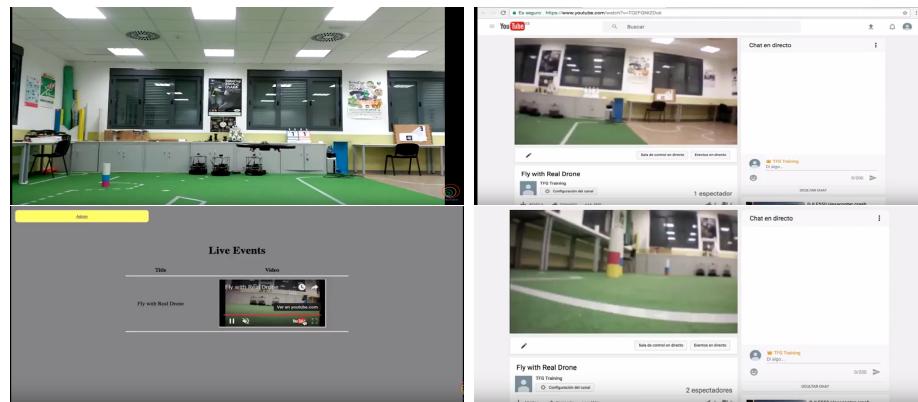


Figura 6.7: Experimento

⁵<http://jderobot.org/Apavo-tfgFlywithRealDrone>

⁶<https://www.youtube.com/watch?v=jo67tP62-Uw>

Capítulo 7

Servidor Imágenes de vídeos en la red

La tercera aplicación lleva a cabo el proceso inverso a las aplicaciones presentadas en los capítulos cuatro y cinco. Esas dos aplicaciones enviaban vídeos(cámara local o cámara de un drone) a YouTube, esta aplicación descarga en tiempo real, el flujo de vídeo de un evento en directo de YouTube y lo muestra a través de las herramientas de visualización de JdeRobot. Sirve los fotogramas, uno a uno, a las aplicaciones JdeRobot de procesamiento de imágenes. Es por ello un driver de flujos de vídeo dentro del entorno JdeRobot, con la peculiaridad de que la fuente original está en una URL de YouTube, de la que se va descargando el flujo como cliente streaming. A este servidor lo llamamos YouTubeServer.

7.1. Diseño

La aplicación consiste en un servidor *ICE* programado en Python, que se encarga de descargar el vídeo de YouTube y enviarlo fotograma a fotograma a las aplicaciones de JdeRobot. Esta construido con cuatro bloques funcionales.

- a) **Conexión YouTube:** YouTube a través de *youtube-dl*¹ proporciona al servidor la lista de direcciones de descarga de la retransmisión, típicas de la descarga por streaming adaptativo por *MPEG-DASH* desde los servidores YouTube de vídeo.
- b) **Descarga con ffmpeg:** Con la dirección facilitada por *youtube-dl*, ffmpeg descarga el flujo de vídeo de los servidores de YouTube.
- c) **Extracción de frames:** JdeRobot no trabaja con vídeo solo con imágenes sueltas, es decir fotograma a fotograma, por lo que ffmpeg, descompone el vídeo en imágenes que almacena localmente.
- d) **Comunicación ICE:** envía los fotograma almacenados a las aplicaciones de procesamiento de imágenes utilizando estándezar de JdeRobot para ello, el interfaz Camera.

¹<https://rg3.github.io/youtube-dl/>

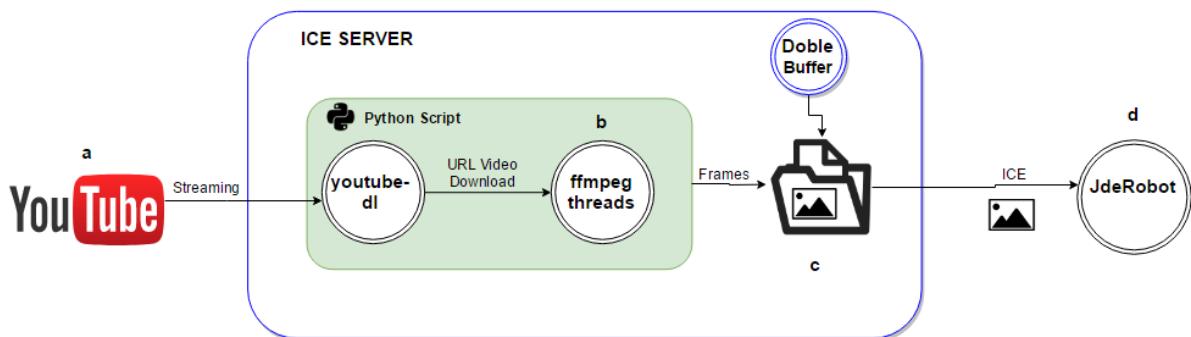


Figura 7.1: Arquitectura YouTubeServer

7.2. Comunicación con YouTube

Los eventos en directo de YouTube tienen asociada una lista, que contiene las direcciones web del flujo de vídeo en distintas calidades(240p,360p,720p....), almacenado en los servidores de YouTube. *Youtube-dl* proporciona direcciones de esta lista con siguiente comando:

```
youtube-dl -f 92 -g URL
```

La opción *-f* determina la calidad del flujo de vídeo, definido en la URL, que queremos obtener de manera progresiva. En el ejemplo, el parámetro URL es definido en el fichero de configuración del driver y se devolverá la dirección de descarga del vídeo en calidad 240p, que corresponde con el 92.

Una vez obtenida la dirección de descarga, *ffmpeg* se encarga de descargarla. Al ser un evento en directo el vídeo no está completo, por lo cual la descarga se debe realizar por fragmentos, *transports streams (.ts)* que son un tipo de archivo definido en la especificación del estándar *MPEG-2* y usados para la descarga de contenido *streaming*.

Tanto *ffmpeg* como *youtube-dl* son ejecutados en la terminal desde un *script* de Python que se encuentra en el servidor.

```

def setFileList(self):
    command = shlex.split('youtube-dl -f 92 -g ' + self.url)
    process= Popen(command ,stdout=PIPE, stderr=PIPE)
    self.fileList=process.stdout.read()
    process.stdout.close()

def downloadVideo(self):
    data = self.fileList.splitlines()
    data= data[0].decode('utf-8')
    command=shlex.split('ffmpeg -i ' + data + ' -c copy output.ts')
    process= Popen(command, stdout=PIPE, stderr=PIPE)

```

7.3. Extracción de fotogramas y comunicación ICE

Las herramientas de JdeRobot que permiten visualizar contenido, como *uav_viewer* o *cameraview*, no trabajan con flujos de vídeo sino con fotogramas, por lo que el vídeo tiene que ser descompuesto en fotogramas antes de enviarlo a aplicaciones JdeRobot.

De nuevo ffmpeg será el encargado de descomponer el vídeo previamente descargado en imágenes. Al ser una descarga en tiempo real, el archivo de vídeo esta cambiando constantemente, por lo que para extraer correctamente los fotogramas sueltos, antes de ejecutar el comando de ffmpeg se debe extraer la duración del fragmento de vídeo en ese momento. Para ello se usa *ffprobe* una herramienta de ffmpeg que extrae datos de archivos, en este caso del fragmento de vídeo.

El proceso que lleva a cabo la extracción de fotogramas recibe dos parámetros de entrada que son *init_time*, indica el instante del vídeo a partir del cual se deben empezar a extraer imágenes, de esta forma se evita extraer imágenes de partes del vídeo ya procesadas, y *end_time* que representa el final del fragmento de vídeo.

Los fotogramas son extraídos uno a uno, se almacenan localmente y una vez son enviados se sobrescriben. Para que no provocar errores de lectura-escritura simultanea se usa la técnica del *doble buffer* descrita en el capítulo anterior.

```

def getImage(self, init_time, end_time):
    init_time = datetime.strptime(init_time, '%H:%M:%S')
    end_time = datetime.strptime(end_time, '%H:%M:%S')
    command = shlex.split("ffmpeg -i output.ts -start_number 0 -vf fps
                           =5 -ss " + init_time + " -to " + end_time + " -f image2 -
                           updatefirst 1 temp.jpg")
    process= Popen(command, stdout=PIPE, stderr=PIPE)

def changeName(self):
    if os.path.isfile('./temp.jpg'):
        os.rename("temp.jpg","image.jpg")

def getVideoDuration(self):
    command = shlex.split('ffprobe -show_entries format=duration -
                           sexagesimal output.ts')
    process = Popen(command ,stdout=PIPE, stderr=PIPE)
    time = process.stdout.read()
    time = time.decode('utf-8')
    time = time.split('\n')[1].split('=')[1]
    time = datetime.strptime(time.split('.')[0], '%H:%M:%S')
    process.stdout.close()
    return time

```

Una vez almacenadas localmente las imágenes son enviadas a las aplicaciones JdeRobot usando el middleware de comunicaciones ICE y su bibliotecas. JdeRobot tiene implementados distintos interfaces ICE, que definen las operaciones asociadas a un objeto. Para esta aplicación se han usado el interfaz *imageprovider*² y el interfaz

²<https://github.com/JdeRobot/JdeRobot/blob/master/src/interfaces/slice/jderobot/image.ice>

`camera`³ que hereda de `imageprovider`.

Para darles funcionalidad las operaciones de los interfaces deben ser sobrescritas en el servidor. Aunque solo dos operaciones `getImageDescription` y `getImageData` son usadas se deben sobrescribir todas las operaciones del interfaz.

```
class ImageProviderI(jderobot.Camera):
    .....
    def getImageDescription(self, current=None):
        self.imageData = jderobot.ImageDescription()
        if os.path.isfile('./image.jpg'):
            self.image= Image.open('./image.jpg')
            self.imageData.width = self.image.width
            self.imageData.height = self.image.height
            self.format = 'RGB'
        return self.imageData

    def getImageData_async(self, cb, formato, current=None):
        job = Job(cb, formato)
        return self.workQueue.add(job)
```

La operación `getImageData` es una operación asíncrona, es decir se ejecuta de forma paralela al servidor ICE sin interrumpir su flujo principal. Para tratar esta operación se ha implementado una cola *first in first out*, que almacena los datos de cada petición del cliente. Paralelamente al hilo principal del servidor se ejecuta otro hilo que se encarga de procesar las operaciones encoladas por orden de entrada.

```
class WorkQueue(threading.Thread):
    def __init__(self):
        self.callbacks = []
        threading.Thread.__init__(self)

    def run(self):
        if not len(self.callbacks) == 0:
            self.callbacks[0].execute()
            del self.callbacks[0]

    def add(self, job):
        self.callbacks.append(job)
        self.run()

class Job(object):
    def __init__(self, cb, formato):
        self.cb = cb
        self.format = formato
```

³<https://github.com/JdeRobot/JdeRobot/blob/master/src/interfaces/slice/jderobot/camera.ice>

```

    self.imageDescription = jderobot.ImageData()

def execute(self):
    if not self.getData():
        print("No data")
        #self.cb.ice_exception(jderobot.Image.DataNotExistException())
        return
    self.cb.ice_response(self.imageDescription)

def getData(self):
    if os.path.isfile('./image.jpg'):
        self.imageDescription = jderobot.ImageData()
        self.imageDescription.description = ImageProviderI.
            getImageDescription(self)
        self.im = Image.open('./image.jpg', 'r')
        self.im = self.im.convert('RGB')
        self.imRGB = list(self.im.getdata())
        self.pixelData = []
        for pixelList in self.imRGB:
            for pixel in pixelList:
                self.pixelData.append(pixel)
        self.imageDescription.pixelData = self.pixelData
        return True
    else:
        return False

```

7.4. Experimentos

Para verificar el funcionamiento del driver de YouTube, se ha creado e iniciado una retransmisión en directo en YouTube. La URL de este evento ha sido añadida en el fichero de configuración del driver y la calidad del evento elegida ha sido 240p, siendo el *framerate*, configurado tanto en la descarga como en la transferencia de imágenes es de 25fps.

La aplicación elegida para la prueba es la herramienta **UAVviewer**, que simplemente muestra por pantalla las imágenes recibidas y se conecta driver a través de ICE.

El resultado del experimento⁴ ha sido satisfactorio, cumpliendo el objetivo de desarrollar un driver que muestre imágenes en el interfaz de JdeRobot.

Aunque la conexión ha sido satisfactoria se han observado ciertos retardos en la transferencia de imágenes a JdeRobot. Por otro lado, la calidad de la imagen se ha visto resentida también respecto a las imágenes originales, aunque difícilmente visible por el ojo humano. Esta perdida de calidad es provocada por la compresión de las imágenes a la hora de descargar el vídeo desde YouTube.

⁴<http://jderobot.org/Apavo-tfgYouTubeServerJdeRobot>

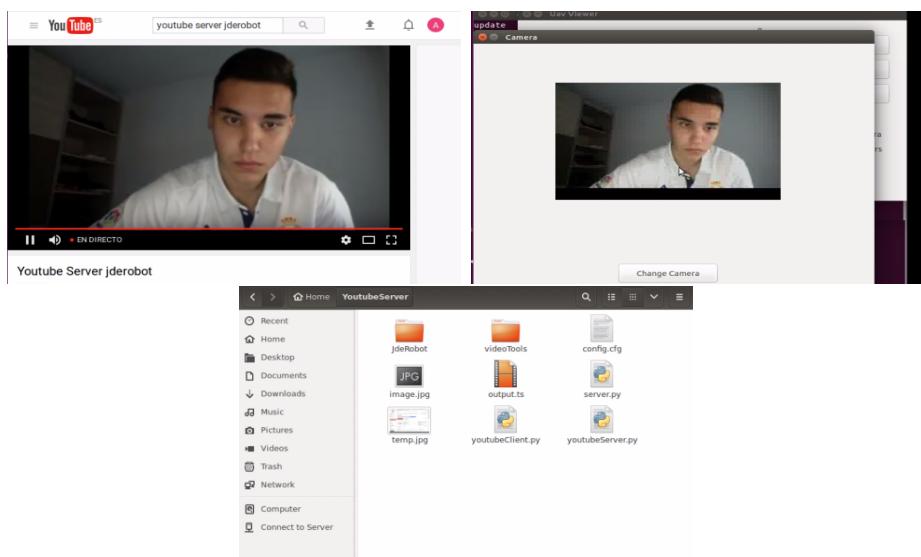


Figura 7.2: Vídeo de YouTube mostrado en un cliente web normal y en la aplicación UAVviewer de JdeRobot usando el driver YouTubeServer

Capítulo 8

Conclusiones

Una vez llegados a este punto y finalizado el aprendizaje y desarrollo de aplicaciones es la hora de evaluar el trabajo realizado volviendo a los objetivos marcados en el capítulo dos y valorando en qué medida se han conseguido. Como valoración global del proyecto consideramos que los objetivos presentados a principios de el capítulo dos han sido conseguidos de forma satisfactoria.

El objetivo principal de interconectar las tecnologías JdeRobot y YouTube se ha conseguido con el desarrollo de dos aplicaciones web y un driver para JdeRobot.

- El primer subobjetivo de crear una aplicación web que retransmita el contenido de una cámara web vía YouTube, se ha conseguido desarrollando un componente en Python que maneja el API de YouTube y emplea ffmpeg para capturar imágenes de una cámara local. Este flujo visual es enviado a los servidores de YouTube, quien se encarga posteriormente de retransmitir el contenido a miles de clientes potenciales tal y como se describe en el capítulo cuatro. Con este punto se ha conseguido aprender la interfaz de programación de YouTube, así como el manejo de eventos, a parte de la gestión de contenido audiovisual con ffmpeg y los servidores de YouTube.
- El segundo subobjetivo planteado fue crear una aplicación web que retransmitiera las imágenes captadas por un drone a través de YouTube, y se ha satisfecho con el desarrollo en Python de un adaptador que recibe las imágenes captadas por una cámara a bordo de un drone a través de los interfaces de JdeRobot y ffmpeg se encarga de enviarlas a los servidores de YouTube para su retransmisión como se especifica en el capítulo cinco. El aporte fundamental en este punto comparado con el anterior es que se pasa de tomar las imágenes de una cámara local a poderlas tomar a través de una cámara remota la cual puede estar a bordo de un drone. Además se ha adaptado la toma de imágenes para compatibilizar la aplicación web con las aplicaciones de la plataforma robótica de JdeRobot.
- El tercer subobjetivo de visualizar contenido YouTube en las interfaces de JdeRobot, se ha completado con el desarrollo de un driver en Python que recibe un flujo audiovisual de YouTube, lo descompone en fotogramas y mostrarlo a través del interfaz estándar de JdeRobot, explicado en el capítulo seis. La novedad en este punto con respecto a los dos anteriores es que se implementa el

camino opuesto, es decir se usa YouTube como fuente de vídeo. Se ha conseguido conectar flujos de vídeo en directo de YouTube con las aplicaciones de JdeRobot, añadiendo de esta forma una nueva fuente de vídeo a las ya existentes en la plataforma.

Todo esto se ha validado experimentalmente y documentado, como se puede observar en los experimentos llevados acabo a lo largo del desarrollo todos ellos disponibles en la wiki oficial del proyecto ¹.

Echando la vista atrás y analizando los proyectos antecesores se ha conseguido dar un gran salto en la difusión del contenido audiovisual ya que en esta ocasión gracias a la inclusión de la tecnología de YouTube disponemos de una capacidad de difusión prácticamente infinita pudiendo llegar este contenido a millones de personas, todo ello en tiempo real.

Por otro lado se ha añadido una gran heterogeneidad de plataformas e infraestructuras como YouTube, software robótico, ffmpeg o librerías de visualización.

8.1. Lineas Futuras

Tras el punto y final de este trabajo se abre un gran abanico de posibilidades de cara a futuros proyectos. Una línea de crecimiento futuro es la inclusión del audio en el dron ya que a día de hoy este es captado localmente en el servidor.

Por otro lado, y una vez YouTube haya madurado estas características se podría hacer compatible la aplicación web con vídeos 360º o la retransmisión inmediata de contenido sin necesidad de un codificador intermedio.

Otro punto de mejora sería optimizar los tiempos y anchos de bandas de retransmisión buscando mejorar así la calidad de las retransmisiones y aumentar el numero de fotogramas por segundo, tanto en la subida como bajada de contenido a YouTube.

¹<http://jderobot.org/Apavo-tfg>

Bibliografía

- [1] MediaWiki del proyecto <http://jderobot.org/Apavo-tfg>
- [2] Repositorio Github del proyecto <https://github.com/RoboticsURJC-students/2016-tfg-alberto-pavo/>
- [3] Pagina oficial JdeRobot <http://jderobot.org>
- [4] Repositorio Github de JdeRobot <https://github.com/JdeRobot>
- [5] MediaWiki del proyecto de Aitor Martinez Fernandez <http://jderobot.org/Aitormf-tfg>.
- [6] MediaWiki del proyecto de Iván Rodríguez-Bobada Martín <http://jderobot.org/Irodmar-tfg>.
- [7] Pagina Youtube Live Streaming API <https://developers.google.com/youtube/v3/live/getting-started>.
- [8] Soporte oficial de Google sobre YouTube <https://support.google.com/youtube>
- [9] Página oficial de desarrolladores de Google sobre Python <https://developers.google.com/api-client-library/python/>
- [10] Página oficial de FFmpeg <https://ffmpeg.org/>
- [11] Pagina oficial Open Broadcaster Software <https://obsproject.com/>
- [12] Información sobre OBS https://en.wikipedia.org/wiki/Open_Broadcaster_Software
- [13] Pagina oficial de AngularJS <https://angularjs.org/>
- [14] Repositorio oficial de youtube-dl <https://rg3.github.io/youtube-dl/>