# Índice general

**1. Streaming web con YouTube desde un Drone** **3**
1.1. Diseño . . . . . 3
1.2. Adaptador ffmpeg a JdeRobot . . . . . 4
1.3. Comunicacion NodeJS . . . . . 7
1.4. Experimentos . . . . . 8

# Índice de figuras

1.1. Arquitectura Aplicación . . . . . 4
1.2. Esquema adaptador JdeRobot a ffmpeg . . . . . 5
1.3. Técnica de doble buffer . . . . . 7
1.4. Arquitectura Experimento . . . . . 9
1.5. arDrone de Parrot . . . . . 9
1.6. Arquitectura Experimento . . . . . 11
1.7. Experimento . . . . . 11

# Capítulo 1

# Streaming web con YouTube desde un Drone

Esta segunda aplicación web al igual que la explicada en el capitulo 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.

## 1.1. 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.

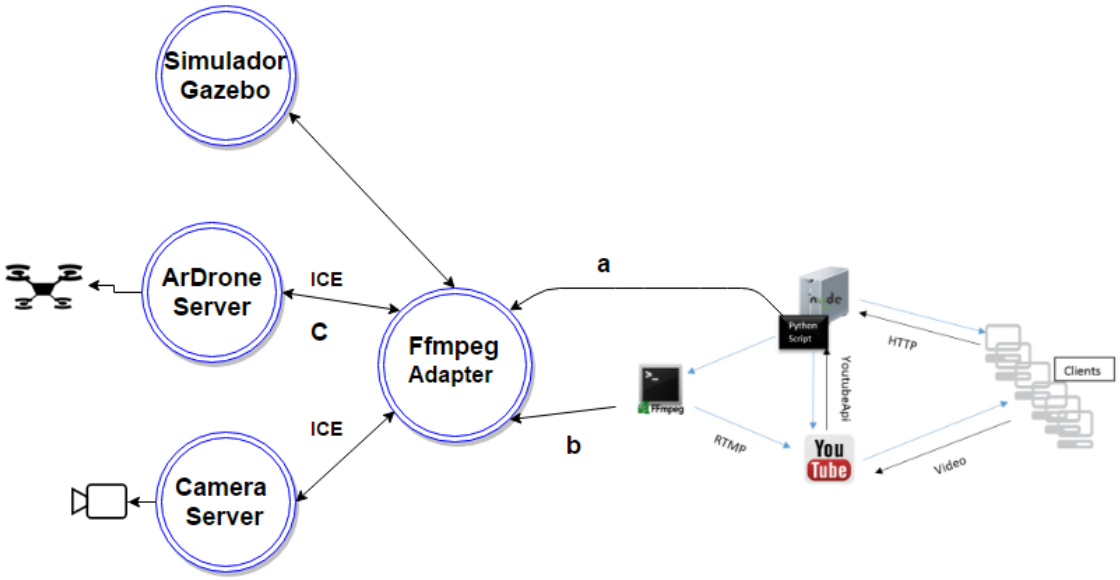


Figura 1.1: Arquitectura Aplicación

## 1.2. Adaptador ffmpeg a JdeRobot

Tanto CameraServer como ArdroneServer, herramientas de vídeo de JdeRobot, proporcionan un flujo de imágenes fotograma a fotograma a traves 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 solo imagen sobrescrita 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 capitulo 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.

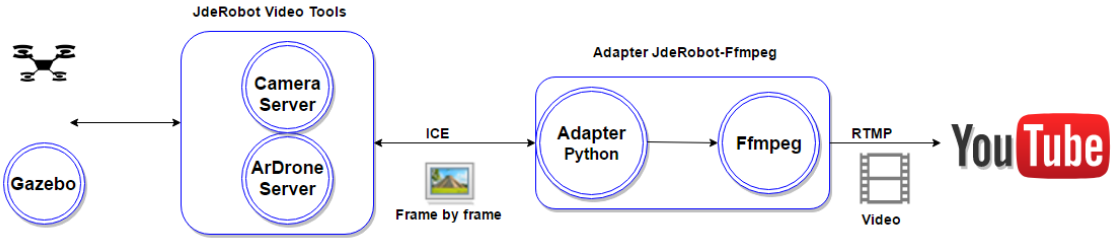


Figura 1.2: Esquema adaptador JdeRobot a ffmpeg

### 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 inicializa 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 interfacer ICE del drone (CameraClient, cmdvel, navDataClient,pose3D). Para nuestro objetivo únicamente necesitaremos conectarnos 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 codigo 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$_py$

[3] http://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.

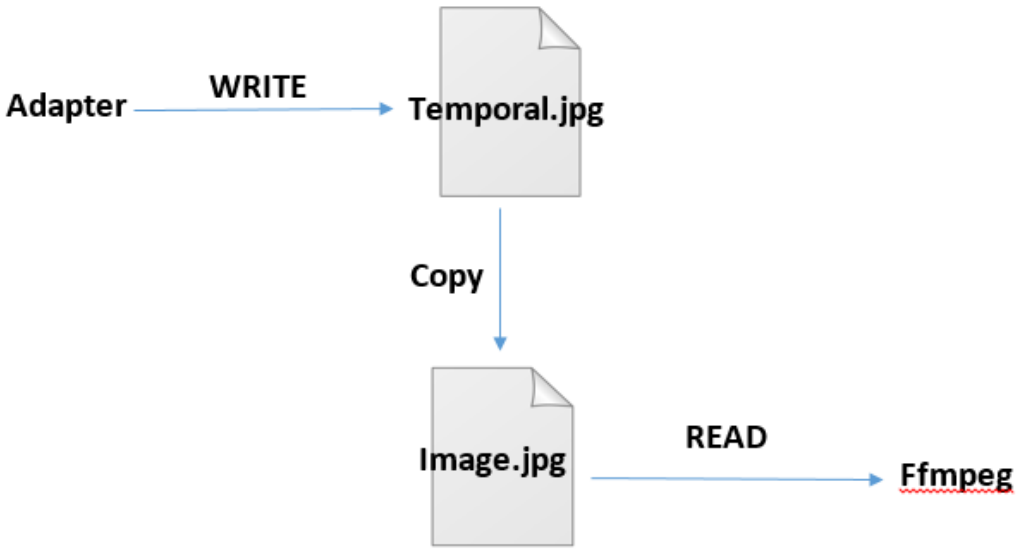


Figura 1.3: Técnica de doble buffer

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(img)
```

Este método pertenece a la clase CameraWidget, dicha clase posee un atributo denominado camera, en la instanciación 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.

## 1.3. Comunicacion NodeJS

El servidor NodeJS, se comunica con Python y YouTube de la misma forma que en la aplicación del capitulo 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")
```

## 1.4. 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 manejamos en JdeRobot proporcionados por el *arDroneServer*(todas estas herramientas están explicadas en el capitulo 3), para nuestro adaptador únicamente necesitamos la interfaz Camera pero para el manejo del Drone si necesitaremos los demás interfaces.

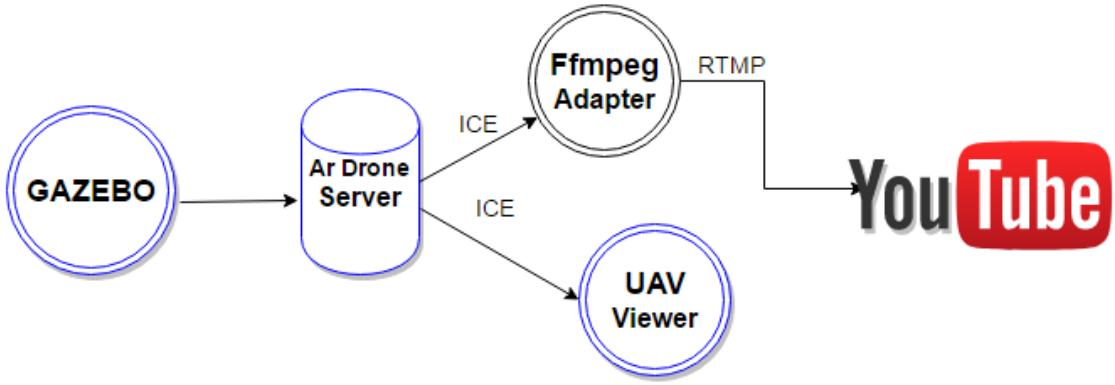


Figura 1.4: Arquitectura Experimento

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 esta 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 proyecto [4].

### 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 cuadricoptero totalmente compatible con las herramientas de JdeRobot.

Figura 1.5: 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.

[4] http://jderobot.org/Apavo-tfgYouTube_{,26,JdeRobot}

*ArDrone server* posee un fichero de configuración que podemos ver a continuación, en él se encuentran detallados en que 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 UAViewer 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 problema es que 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.

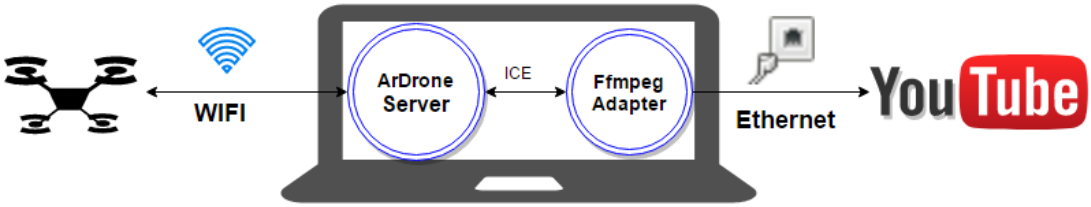


Figura 1.6: Arquitectura Experimento

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 proyecto [5] o en el canal de YouTube[6].

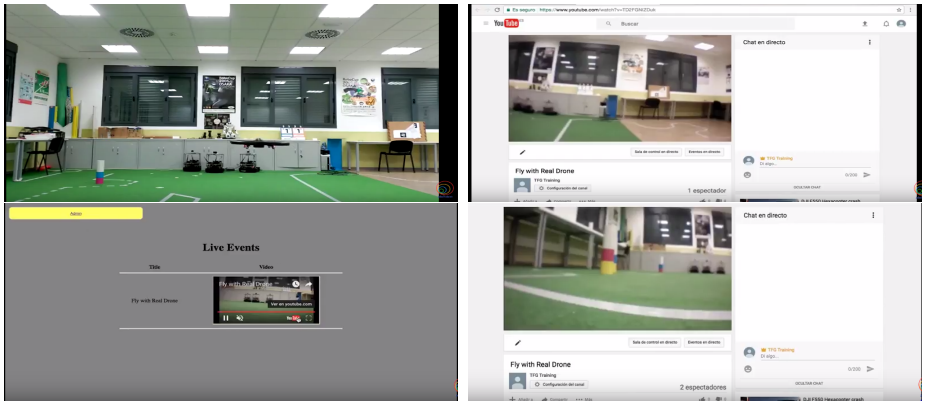


Figura 1.7: Experimento

[5] http://jderobot.org/Apavo-tfg$Fly_{with}Real_{D}rone$
[6] https://www.youtube.com/watch?v=jo67tP62-Uw