Índice general

Índice de figuras

Capítulo 1

Visualización YouTube streaming en JdeRobot

La tercera aplicación lleva a cabo el proceso inverso a las aplicaciones presentadas en los capítulos cuatro y cinco. La 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.

1.1. Diseño

La aplicación consta de un servidor *ICE* programado en Python, que se encarga de descargar el vídeo de YouTube y enviarlo fotograma a fotograma a las herramientas de JdeRobot.

a) Conexión YouTube servidor ICE: YouTube a través de *youtube-dl*[1] proporciona al servidor la lista de direcciones de descarga del la retransmisión.

b) Descarga con ffmpeg: Con la dirección facilitada por *youtube-dl*, ffmpeg descargar el flujo de vídeo de los servidores de YouTube.

c) Extracción de frames: JdeRobot no trabaja con vídeo solo con imágenes por lo que ffmpeg, descompone el vídeo en imágenes que almacena localmente.

d) Comunicación servidor JdeRobot: *ICE* envía los fotograma almacenados a las herramientas de JdeRobot para que estas los reproduzcan.

[1] https://rg3.github.io/youtube-dl/

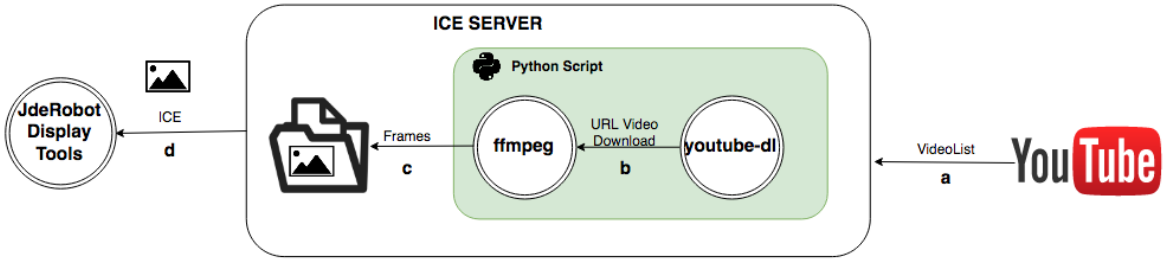

Figura 1.1: Arquitectura YouTubeServer

1.2. Comunicación servidor ICE 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 que queremos obtener, en el caso del ejemplo devolverá la dirección web donde se encuentra almacenado el vídeo en calidad 240p.

Una vez obtenida la dirección de descarga, ffmpeg se encarga de descargarla. Al ser un evento en directo el vídeo no esta 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)
```

1.3. Comunicación servidor ICE con JdeRobot

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 JdeRobot.

De nuevo ffmpeg será el encargado de descomponer el vídeo previamente descargado en imágenes. Ya que es una descarga en tiempo real, el archivo de vídeo esta cambiando constantemente. Es por ello que para extraer correctamente los fotogramas, 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.strftime(init_time,'%H:%M:%S')
    end_time = datetime.strftime(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 JdeRobot con ICE. JdeRobot tiene implementados distintos interfaces ICE, estos interfaces definen operaciones de los objetos ICE. Para esta apliación se han usado el interfaz imageprovider[2] y el interfaz camera[3] que hereda de imageprovider.

[2]https://github.com/JdeRobot/JdeRobot/blob/master/src/interfaces/slice/jderobot/image.ice
[3]https://github.com/JdeRobot/JdeRobot/blob/master/src/interfaces/slice/jderobot/camera.ice

Para darles funcionalidad las operaciones de los interfaces deben ser sobrescritos 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,curren=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
    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 pixeList in self.imRGB:
        for pixel in pixeList:
          self.pixelData.append(pixel)
      self.imageDescription.pixelData = self.pixelData
      return True
    else:
      return False
```

1.4. Experimentos

Para verificar el funcionamiento de la aplicación, se ha creado ha iniciado una retransmisión en directo en YouTube. La url de este evento ha sido añadida en el fichero de configuración del servidor ICE. El elemento seleccionado para la visualización del vídeo es UAVviewer que conecta al servidor a través de ICE. La calidad del evento en YouTube es 240p. El resultado del experimento[4] ha sido satisfactorio, cumpliendo el objetivo de desarrollar un driver que muestre imágenes en el interfaz de JdeRobot.

[4]http://jderobot.org/Apavo-tfgYouTubeServer*JdeRobot*

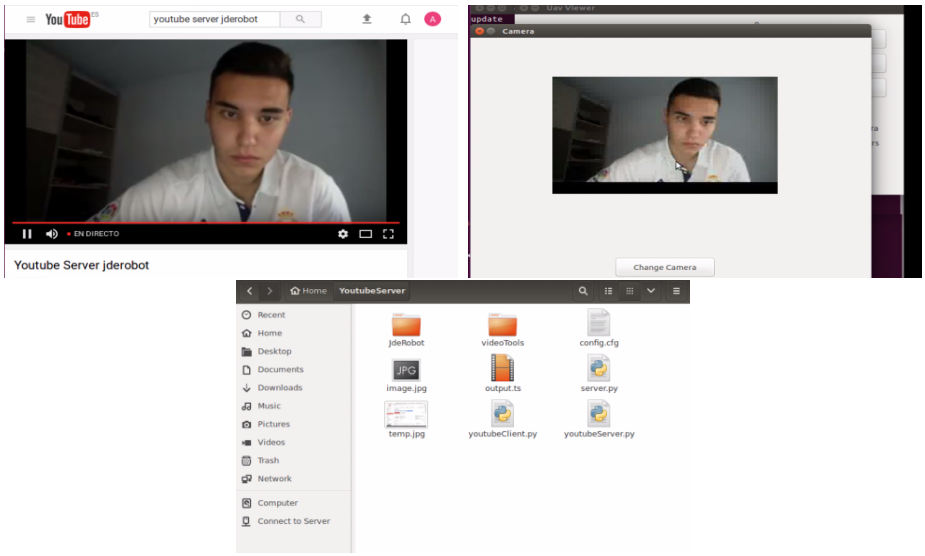

Figura 1.2: Experimento