# Interpolación de imágenes en vídeo

**Tecnologías Multimedia**


UNIVERSIDAD DE ALMERÍA

1/02/2018

Antonio Navarro Duarte, Jesús Pérez Escámez y Juan José Camacho Hidalgo

# Tabla de contenido

# Introducción

Este proyecto tiene como objetivo la realización de una aplicación en el lenguaje Python cuya finalidad es la generación de imágenes situadas entre imágenes conocidas de una secuencia de vídeo. Esta técnica consiste en, dadas dos imágenes consecutivas A y C, construir una imagen intermedia (desconocida) B, usando como datos los píxeles de A y C.

La técnica de interpolación que usamos es la estimación del flujo óptico (Optical Flow), siendo ésta una de las que mejores resultados proporciona.

Nuestra idea era la de una vez obtenida la aplicación, incluir esta característica en el proyecto FFMPEG, pero el resultado obtenido tras el periodo asignado a este proyecto no es lo suficientemente avanzado para ser incluido.

Nuestra aplicación aplica la técnica de “optical flow” para cualquier vídeo dado, obteniendo otro vídeo con el seguimiento del movimiento dentro de éste. Además, duplicamos la tasa de fotogramas por segundo, pero solo lo hemos conseguido para casos con fotogramas con características muy concretas.

# Nuestra planificación

A nivel de planificación, el proyecto sufrió varias modificaciones en forma de retrasos debido a la comprensión del problema y a temas de investigación acerca de todo lo relacionado con la interpolación de imágenes. Así, se dedicaron algo más de dos meses a este tipo de tareas. La implementación ha sumado aproximadamente un mes.

La organización del proyecto se presentó de la siguiente forma:

Hito 1. Análisis y Diseño: 25 de Septiembre - 22 de Octubre

Hito 2. Implementación Primera Fase: 23 de Octubre - 26 de Noviembre

Hito 3. Implementación Segunda Fase: 27 de Noviembre - 21 de Diciembre

Hito 4. Refinamiento de la Implementación y Pruebas: 22 de Diciembre - 14 de Enero

Sin embargo, el hito 1 finalmente comenzó con retraso, sobre el 20 de octubre, y se extendió hasta finales de diciembre, lo que nos llevó a planificar de nuevo nuestro calendario. La implementación quedó cubriendo todo el mes de Enero.

# Sistemas de interpolación

Un sistema de interpolación es una técnica que, haciendo uso de los fotogramas de un vídeo dado, es capaz de generar nuevos fotogramas los cuales, en principio, se corresponderían con fotogramas intermedios entre otros dos existentes y consecutivos en el vídeo objeto de la técnica. El propósito de esta técnica de cara al vídeo resultante sería hacer el movimiento más fluido, así como compensar el desenfoque de movimiento.

Para lograr este cometido, los sistemas de interpolación hacen uso de distintos medios de interpolación.

Se conoce como interpolación, en el campo del análisis numérico, al método por el que se pueden construir nuevos puntos dentro del rango de un conjunto discreto de puntos que son ya conocidos. En el caso de archivos de vídeo, se pretende utilizar este método para construir nuevas imágenes dado que estas están formadas por píxeles que son tomados como puntos para hacer uso de estos métodos.

A nivel matemático, existen cuatro métodos de interpolación muy conocidos, que procedemos a explicar con brevedad.

### Interpolación constante por piezas

También conocida como Interpolación del vecino más cercano. En este método de interpolación se considera que los puntos desconocidos más cercanos a los ya conocidos tienen el mismo valor que estos. Es el método de interpolación más sencillo, pero el menos usado dada su imprecisión:

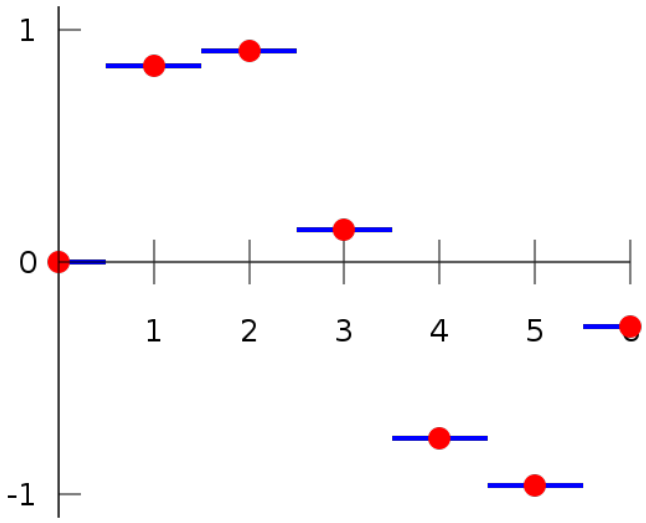



*Ilustración 1: Interpolación constante por piezas*

## Interpolación lineal

Se trata también de un método simple. En este caso se considera que los puntos desconocidos intermedios entre dos puntos conocidos consecutivos pertenecen a la línea recta que une ambos puntos.

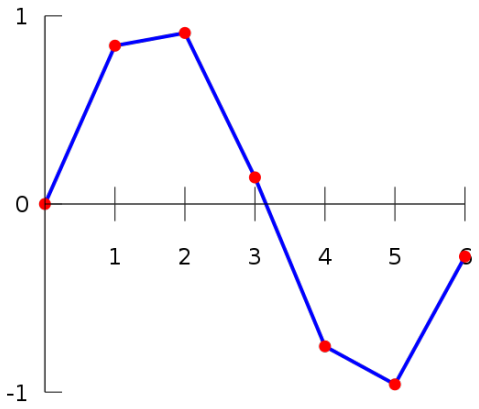



*Ilustración 2: Interpolación lineal*

## Interpolación polinómica

Este método de interpolación es la generalización de la interpolación lineal. Si los puntos desconocidos entre otros dos en el caso de la interpolación lineal podían ser expresados en forma de una función lineal –una función polinómica de grado uno o cero–, los puntos desconocidos de la interpolación polinomial formarán parte de una función polinómica de grado mayor que uno. Visualmente el resultado suele corresponderse con una curva.

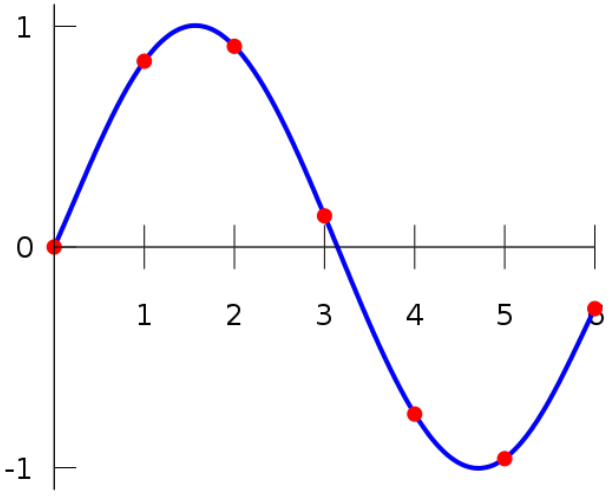



*Ilustración 3: Interpolación polinómica*

## Interpolación spline

Por su traducción, interpolación de ranura o interpolación de astilla. Del mismo modo que la interpolación lineal tomaba cada uno de los intervalos entre dos puntos conocidos como una función lineal, esta técnica de interpolación tomará cada uno de esos intervalos, pero como una función polinómica de grado mayor que uno, de forma que cada "pieza" encaje de forma suavizada con las otras. La diferencia de esta técnica con la interpolación polinómica es que, mientras la primera toma cada uno de los intervalos entre los puntos conocidos como una función distinta, la interpolación polinómica trata de abarcar todos los puntos con la misma función. Es la interpolación que menor margen de error presenta.

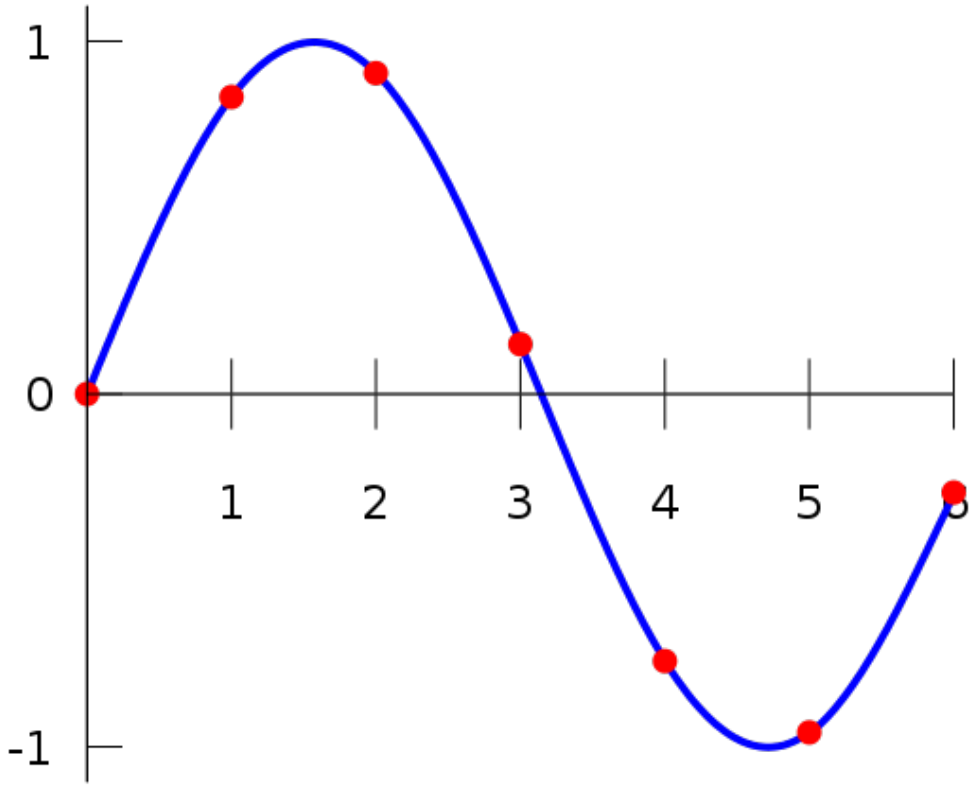
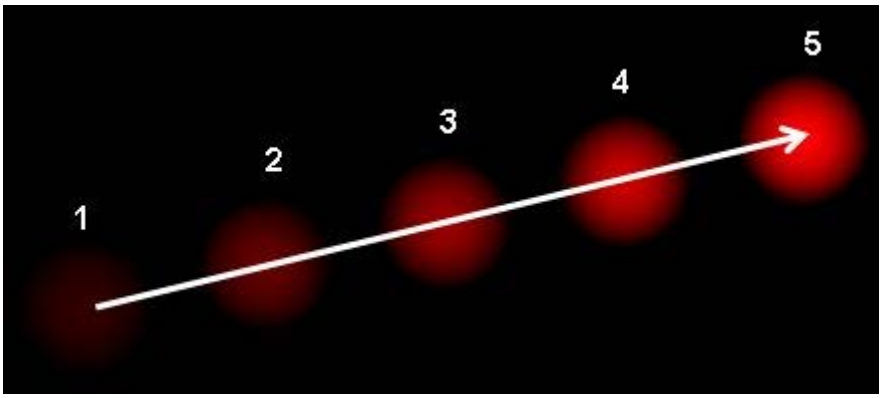



*Ilustración 4: Interpolación spline*

# Interpolación basada en el flujo óptico

El flujo óptico es un patrón de movimiento aparente de objetos, superficies y contornos en una imagen y es causado por el movimiento relativo entre el observador y la escena. Se trata de un vector de desplazamiento que muestra el movimiento de dos puntos del primer fotograma al segundo.




El flujo óptico trabaja con dos asunciones: que la intensidad de píxeles de un objeto no varía de entre dos fotogramas consecutivos y que píxeles vecinos tendrán movimiento similar.

Si consideramos un pixel $I(x,y,t)$ en el primer fotograma (utilizamos el tiempo como tercera dimensión dado que hablamos de un vídeo), este se moverá en una distancia $(dx,dy)$ para el siguiente fotograma en un tiempo $dt$. Hechas las asunciones previas, podemos decir que:

$$I(x,y,t)=I(x+dx,y+dy,t+dt)$$

Utilizando la aproximación de la serie de Taylor en el lado derecho de la igualdad para eliminar los términos comunes y dividiendo por $dt$, entonces obtenemos la siguiente ecuación:

$$f_x u+f_y v+f_t=0$$

donde:

- $f_x=\partial f/\partial x$
- $f_y=\partial f/\partial y$
- $u=dx/dt$
- $v=dy/dt$

La ecuación de arriba es llamada la Ecuación del Flujo Óptico. $f_x$ y $f_y$ son gradientes de imagen, del mismo modo que $f_t$ es el gradiente de tiempo. No obstante, los valores de vector $(u,v)$ son desconocidos. Como no es posible resolver esta ecuación con dos variables desconocidas, existen varios métodos para estimarlas. Estos son:

- Correlación de fase
- Métodos basados en bloques
- Métodos diferenciales
    - Método de Lucas-Kanade
    - Método de Horn-Schunck
    - Método de Buxton-Buxton
    - Método de Black-Jepson
    - Métodos de varianzas generales
- Métodos de optimización discreta

Para el desarrollo de nuestro programa hemos hecho uso del método de Lucas-Kanade, de modo que nos centraremos en explicar ese.

### Método de Lucas-Kanade

Basándonos en la asunción previa de que todos los pixeles vecinos tendrán movimiento similar, este método toma una parcela de 3x3 alrededor de un punto. Así que se considera que los nueve puntos tienen el mismo movimiento. Es posible calcular $(f_x,f_y,f_t)$ para estos nueve puntos, así que nuestro problema ahora es resolver nueve ecuaciones con dos variables desconocidas u y v. La mejor solución obtenida es con el método de los mínimos cuadrados. El resultado de esto es la siguiente ecuación:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \sum_i f_{x_i}{}^2 & \sum_i f_{x_i} f_{y_i} \\ \sum_i f_{x_i} f_{y_i} & \sum_i f_{y_i}{}^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i f_{x_i} f_{t_i} \\ -\sum_i f_{y_i} f_{t_i} \end{bmatrix}$$

En resumen, desde la vista del usuario, este da unos puntos que rastrear y recibe el vector de flujo óptico de los mismos. Pero este método falla al aplicarlo a movimientos de grandes distancias –de modo que si quisiéramos usarlo con éstas, habría que despreciar las distancias pequeñas y pasar a considerar las grandes distancias como pequeñas–.

## Lenguaje de programación elegido: Python

En esta sección hablamos sobre el lenguaje de programación elegido para nuestro desarrollo. En concreto usamos Python, en la versión 2.7, un lenguaje que soporta orientación a objetos, programación imperativa, y programación funcional, aunque esta es minoritaria. Es un lenguaje interpretado, usa tipado dinámico y es multiplataforma. La filosofía del lenguaje es ofrecer legibilidad en la sintaxis.

Como IDE, es decir, el entorno de desarrollo, usamos PyCharm. Y, además, usamos las APIs de OpenCV, numpy y tkinter, que integramos en el programa.

# Nuestro programa

Nuestro programa se divide en dos partes:

Primero tenemos la interfaz gráfica que se encarga de interactuar con el usuario y en segundo lugar tenemos el algoritmo que calcula el “Optical Flow”.

Para la interfaz gráfica decidimos utilizar la librería “Tkinter” por varias razones:

- Es una librería que viene preinstalada con Python.
- Es simple y fácil de aprender.
- Hay una gran documentación.

La interfaz creada es muy sencilla, consta de tres botones:


El botón “cargar archivo” que permite al usuario escoger un video. El botón “exportar archivo” que indica donde guardar el video como resultado (debe acabar en .avi para qué funcione). Por último, el botón “procesar video” el cual ejecuta el algoritmo sobre el video escogido.

Tanto el cargar archivo como el exportar archivo hacen uso de la clase “tkFileDialog”.

La parte del algoritmo, la cual posee el algoritmo de “Optical Flow”, tiene varios métodos, los más destacados son:

“parámetros_exportar()”, donde se inicializa el video de salida, ingresando el códec a utilizar, la resolución del video y la ubicación del archivo.

"parámetros_detector()", donde se adjunta los parámetros que utilizara el algoritmo de detección de esquinas.

"aplicar_filtro()", donde se realiza el Optical Flow y se guarda el resultado.

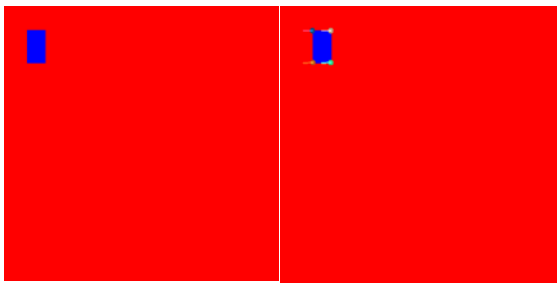
Para la detección de esquinas se ha utilizado el método de la librería "OpenCV" goodFeaturesToTrack() el cual utiliza el algoritmo de "Shi-Tomasi", una modificación del "Harris Corner Detection".

Veamos paso a paso como funciona nuestro algoritmo:

Se toma el primer *frame* del video y aplicamos el método anteriormente mencionado para obtener las esquinas. Luego se toma el segundo frame del video, que junto con el anterior *frame* hacemos uso del método de la librería: "calcOpticalFlowPyrLK()". Esto nos devuelve un seguimiento de las esquinas, pero solo nos quedamos con los puntos buenos, para después dibujar la trayectoria que ha tenido esos puntos respecto al anterior frame. Finalmente guardamos el nuevo *frame* en el video de salida y volvemos hacer de nuevo el mismo proceso con el resto de *frames* del video.

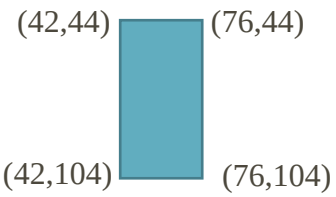
Como se ha comentado anteriormente, no hemos conseguido crear una imagen intermedia con un video cualquiera, por lo tanto, hemos creado un caso particular para poder mostrar cómo funciona nuestro algoritmo.

En este caso tenemos 7 imágenes de 512x512 pixeles, donde tenemos un fondo de color rojo un rectángulo de color azul, el cual se mueve por la imagen. (1.png, 2.png, 3.png…).

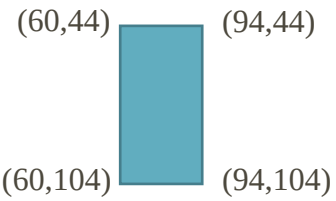


Para que resulte más sencillo solo tendremos en cuenta la primera imagen y la segunda.

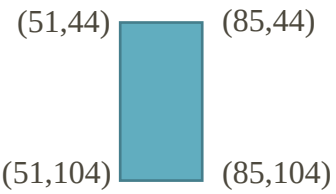
El rectángulo de la primera imagen tiene las siguientes coordenadas:




En la segunda imagen tiene:




Ahora comprobamos la imagen intermedia que ha creado nuestro algoritmo:




Como podemos ver ha creado un *frame* donde el objeto en movimiento está en una posición intermedia entre el frame 1 y el frame 2. (Las imágenes: resInt2.png, resInt3.png… corresponde cona las imágenes intermedias)

En este caso particular es muy sencillo puesto que sabemos de antemano que objeto es el que se mueve, además de que el algoritmo Optical Flow maneja de forma perfecta los puntos por donde se mueve el rectángulo.

# Mejoras posibles

En esta sección, comentamos las posibles mejoras y observaciones respecto a nuestro proyecto.

En primer lugar, partiendo de que hemos logrado el objetivo para un caso concreto y sencillo, lograr aplicarlo para un vídeo cualquiera de la siguiente forma: en lugar de considerar solo las esquinas, tomar el contorno completo de las formas en movimiento y en lugar de reconstruir la forma basándonos solo en sus vértices, reproducir cada uno de los píxeles presentes en el interior de dicha forma.

Por otro lado, otra mejora a tener en cuenta sería la mejora de la eficiencia optimizando el código, obteniendo un resultado de forma más rápida. El procesamiento de cada fotograma se realiza píxel a píxel, siguiendo un esquema de forma bruta. Usando otro esquema algorítmico, mejoraríamos este aspecto.

Por último, comentar la posibilidad de combinar la técnica del “optical flow” con otras técnicas de interpolación en la búsqueda de un resultado más preciso.

# Conclusiones

Como consecuencia de este proyecto, extraemos varias conclusiones:

En primer lugar, el proyecto nos ha resultado más complejo de lo esperado inicialmente, ya que nos esperábamos mayores facilidades en cuanto a recursos, información relevante, de cara al desarrollo del proceso.

A pesar de esto, nos ha servido para comprender las diversas técnicas de interpolación, el lenguaje Python, así como sus diversas librerías multimedia.

No obstante, hemos logrado un avance significativo de cara a la consecución de los objetivos propuestos, es decir, al lograr el objetivo en un caso sencillo, pensando con mayor abstracción, podríamos lograrlo para casos más generales.

# Referencias


1. https://w3.ual.es/~vruiz/Investigacion/video_interpolation/index.html
2. https://github.com/Sistemas-Multimedia/MCDWT
3. https://en.wikipedia.org/wiki/Motion_interpolation
4. https://es.wikipedia.org/wiki/Sistema_de_interpolaci%C3%B3n
5. https://en.wikipedia.org/wiki/Interpolation
6. https://en.wikipedia.org/wiki/Linear_interpolation
7. https://en.wikipedia.org/wiki/Linear_function
8. https://en.wikipedia.org/wiki/Least_squares
9. https://en.wikipedia.org/wiki/Optical_flow
10. https://docs.opencv.org/3.3.1/d7/d8b/tutorial_py_lucas_kanade.html


# Apéndice

A continuación, se muestra el código completo:

Interfaz_grafica.py

```
# -*- coding: utf-8 -*-
import tkFileDialog
import ttk
from Ventana import Frame
from OpticalFlow import FlujoOptico
from Tkinter import PhotoImage


# Metodo que permite abrir un fichero.
def abrir_fichero():
    ventana.filename =
tkFileDialog.askopenfilename(initialdir="/home/antonio/", title="Seleccione
un video",
                                                    filetypes=(("Archivos
avi", "*.avi"), ("Todos", "*.*")))

    flujo.cambiar_ruta(ventana.filename)
    flujo.capturar()


# Metodo que permite guardar un fichero.
def guardar_fichero():
    ventana.filename = tkFileDialog.asksaveasfilename(title="Seleccione
ubicación",
                                                      filetypes=(("Archivo
avi", "avi"), ("Todos", "*.*")))

    btnProcesar['state'] = 'normal'
    flujo.ruta_guardar_archivo(ventana.filename)


# Metodo que procesa el video
def procesar_fichero():
    flujo.parametros_detector(100, 0.3, 7, 7)
```

```
    flujo.parametros_exportar()


# Se crea un objeto de tipo Frame la cual contiene la ventana principal
ventana = Frame(800, 400)
flujo = FlujoOptico("")
cont = 0

# Creamos los label como titulo de la aplicacion y los autores.
titulo = ttk.Label(ventana.window, text="Interpolación de imágenes en
video")
titulo.config(font=("Calibri", 25))  # Se ajusta el tipo de fuente y el
tamaño de fuente.
autores = ttk.Label(ventana.window, text="Antonio Navarro Duarte | Jesús
Pérez Escámez | Juan José Camacho Hidalgo")

# Logo de la ual.
img = PhotoImage(file='logo.gif')
logo = ttk.Label(ventana.window, image=img)
logo.place(x=20, y=240)

# Creamos los botones
btnLoad = ttk.Button(ventana.window, text="Cargar archivo",
command=abrir_fichero)
btnExp = ttk.Button(ventana.window, text="Exportar archivo",
command=guardar_fichero)
btnProcesar = ttk.Button(ventana.window, text="Procesar video",
command=procesar_fichero)
btnProcesar['state'] = 'disabled'


# Le damos una posición a los botones en la ventana.
btnLoad.place(x=73, y=60, width=120, height=25)
btnProcesar.place(x=339, y=60, width=120, height=25)
btnExp.place(x=605, y=60, width=120, height=25)

# Añadimos los label
titulo.pack()
autores.place(x=330, y=375)

ventana.bucle_ventana()  # Se mantiene en bucle hasta que el usuario cierre
la ventana.
```

Ventana.py

```
# -*- coding: utf-8 -*-
import Tkinter


class Frame:
    window = Tkinter.Tk()  # Crea una ventana Tk root.
    filename = ""

    def __init__(self, ancho, alto):
        # Damos un nombre al titulo de la ventana.
        self.window.title("Optical Flow")

        # Obtenemos el ancho y alto de la pantalla.
        ancho_pant = self.window.winfo_screenwidth()  # Ancho de la
pantalla.
        alto_pant = self.window.winfo_screenheight()  # Altura de la
```

```
pantalla.

        # Calculamos las coordenas (x e y) para establecer la posicion de
la pantalla.
        x = (ancho_pant / 2) - (ancho / 2)
        y = (alto_pant / 2) - (alto / 2)

        # Establecemos las dimensiones de la pantalla y lo colocamos.
        self.window.geometry('%dx%d+%d+%d' % (ancho, alto, x, y))

    def bucle_ventana(self):
        self.window.mainloop()
```

OpticalFlow.py

```
# -*- coding: utf-8 -*-
import numpy as np
import cv2


class FlujoOptico:
    # Ruta donde se encuentra el archivo de video
    rutaOpen = ""
    # Parametros para el ShiTomasi
    feature_params = ""
    # Parametros para el optical flow
    lk_params = dict(winSize=(15, 15),
                     maxLevel=2,
                     criteria=(cv2.TERM_CRITERIA_EPS |
cv2.TERM_CRITERIA_COUNT, 10, 0.03))
    # Captura de video original
    cap = ""
    # Captura de video de salida
    out = ""
    # Ruta del archivo de salida
    rutaSave = ""

    # Creamos unos colores aleatorios para los puntos y lineas.
    color = np.random.randint(0, 255, (100, 3))

    def __init__(self, path):
        self.rutaOpen = path

    # Metodo para empezar a capturar el video.
    def capturar(self):
        self.cap = cv2.VideoCapture(self.rutaOpen)

    # Metodo que cambia la ruta del archivo.
    def cambiar_ruta(self, path):
        self.rutaOpen = path

    # Metodo para salvar el archivo
    def ruta_guardar_archivo(self, path):
        self.rutaSave = path

    # Metodo que configura los parametros del archivo a exportar
    def parametros_exportar(self):
        # Se define el codec y se crea un objeto VideoWriter
        fourcc = cv2.VideoWriter_fourcc(*'XVID')

        # Obtenemos el ancho y alto del video original
```

```
        width = self.cap.get(cv2.CAP_PROP_FRAME_WIDTH)
        height = self.cap.get(cv2.CAP_PROP_FRAME_HEIGHT)

        # Abrimos un flujo donde se guardara el video
        self.out = cv2.VideoWriter(self.rutaSave, fourcc, 30, (int(width),
int(height)))

    # Metodo que define los parametros necesarios para el algoritmo de
detección de esquinas (ShiTomasi)
    def parametros_detector(self, max_corner, quality_level, min_distance,
block_size):
        self.feature_params = dict(maxCorners=max_corner,
                                   qualityLevel=quality_level,
                                   minDistance=min_distance,
                                   blockSize=block_size)

    def aplicar_filtro(self):
        # Primero comprobamos si el cap esta abierto
        if not self.cap.isOpened():
            self.cap = cv2.VideoCapture(self.rutaOpen)

        # Tomamos el primer frame del video
        ret, old_frame = self.cap.read()

        # Convertimos el frame de RGB a blanco y negro
        old_gray = cv2.cvtColor(old_frame, cv2.COLOR_BGR2GRAY)

        # Encontramos las esquinas del frame con el algoritmo de ShiTomasi
        p0 = cv2.goodFeaturesToTrack(old_gray, mask=None,
**self.feature_params)

        # Creamos una mascara donde se dibujaran los puntos y el
seguimiento de estos.
        mask = np.zeros_like(old_frame)

        # Ahora empezamos a recorrer los siguientes frames del video
        while True:
            # Capturamos un frame del video
            ret, frame = self.cap.read()

            # Si ocurre algún error al leer el frame paramos el bucle.
            if not ret:
                break

            # Convertimos el frame leido de RGB una escala de grises
            frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

            # Calculamos el flujo optico entre el primer frame que hemos
leido y este último
            p1, st, err = cv2.calcOpticalFlowPyrLK(old_gray, frame_gray,
p0, None, **self.lk_params)

            # Seleccionamos solo los buenos puntos.
            good_new = p1[st == 1]
            good_old = p0[st == 1]

            # Dibujamos el camino que une los puntos (seguimiento del
movimiento)
            for i, (new, old) in enumerate(zip(good_new, good_old)):
                a, b = new.ravel()
                c, d = old.ravel()
                mask = cv2.line(mask, (a, b), (c, d),
self.color[i].tolist(), 2)
```

```
                frame = cv2.circle(frame, (a, b), 5,
self.color[i].tolist(), -1)

            # Guardamos la nueva imagen generada y la mostramos
            img = cv2.add(frame, mask)
            cv2.imshow("Previsualizacion", img)

            # Guardamos el frame en el archivo
            self.out.write(img)
            # Definimos como parar el proceso.
            k = cv2.waitKey(30) & 0xff
            if k == 27:
                break

            # Por último actualizamos el frame anterior por el nuevo y los
anteriores puntos por los nuevos
            old_gray = frame_gray.copy()
            p0 = good_new.reshape(-1, 1, 2)

        # Destruimos todas las ventanas
        cv2.destroyAllWindows()
        self.cap.release()
        self.out.release()
```