Índice

1. Funcionamiento del software y código.

El software consiste en la lectura y procesamiento de un vídeo guardado en el disco duro del ordenador personal y en formato MP4. El vídeo es de un conjunto de helióstatos que van entrando uno por uno en un panel solar y fusionándose todos entre sí. Y tras esto, los helióstatos van saliendo uno por uno de ese panel solar, hasta que no quede ninguno.

El código desarrollado es el siguiente:

Bibliotecas requeridas para este software. import cv2 import argparse import time import numpy as np

Para que funcione todo el código expuesto y elaborado a continuación, hay que importar en este caso las librerías 'cv2', 'argparse', 'time' y 'numpy' (esta última se ha asignado un nombre propio: 'np').

'cv2' permite el uso de distintas funcionalidades de Python. En este proyecto, se ha usado para la captura y lectura de un vídeo guardado en el sistema, convertirlo de color a escala de grises, aplicarle un umbral (diferenciar solo dos niveles de grises: u oscuros o claros), mostrar el vídeo original y umbralizado en pantalla y en reproducción (conforme el programa lo va analizando fotograma a fotograma), realizar pausas (breves o prolongadas) de la ejecución del código, detectar y localizar contornos (helióstatos) en el vídeo, calcular el ancho y alto en píxeles de los contornos, así como sus valores de área, calcular la sumatoria parcial y total de los valores de todos los píxeles BGR al cuadrado de cada helióstato, y dibujar un rectángulo verde o de otro color alrededor del contorno en el vídeo. 'argparse' es usado para proporcionar distintos parámetros desde la consola de Windows al ejecutar este código, de tal forma que el programa sea ejecutado de una forma u otra según la petición de datos del usuario, como las dimensiones del helióstato que el programa analizará, y así ignorar los helióstatos con dimensiones ajenas a las deseadas por el usuario. 'time' permite medir tiempos de ejecución de todo el código o de fragmentos de código, en segundos. Se utilizará para medir la tasa de 'frames' (fotogramas) por segundo de la lectura y procesado del vídeo de helióstatos, y así comprobar la eficiencia de la ejecución del programa. 'numpy' permite el manejo de arrays o vectores con el fin de recorrer y analizar una serie de datos en secuencia, como una matriz de dos dimensiones y con muchos valores numéricos guardados. Además, usar 'numpy' para dicho fin es mucho más eficiente que otros métodos de análisis de datos en secuencia que no sean en arrays o vectores, como los típicos bucles 'for'.

start_time = time.time() # Obtener el tiempo de ejecución inicial de este programa. frame_counter = 0 # Contador de fotogramas totales del vídeo. Se irá incrementando progresivamente en líneas de código posteriores.

Obtener en segundos el tiempo de ejecución inicial del programa. Inicial porque esta línea de código se ubica en el comienzo del software. Declarar e inicializar a cero un contador de fotogramas del vídeo. Ambos datos serán usados para calcular los FPS (fotogramas por segundo) del vídeo de helióstatos, al final de este código.

Argumentos o parámetros necesarios para ejecutar este programa a través de la consola de Windows. parser = argparse.ArgumentParser(description='Parametros del programa.') # Dar un nombre al conjunto de parámetros y asignarlo a la variable 'parser'. parser.add_argument('directorioVideoHeliostatosCargar', type=str) # Crear el argumento 1: ruta o directorio del vídeo a cargar en el PC. parser.add_argument('anchoMinimo type=int) # Crear el argumento 3: ancho mínimo del helióstato para su análisis. parser.add_argument('altoMinimoHeliostato', type=int) # Crear el argumento 4: alto mínimo del helióstato para su análisis. parser.add_argument('umbralVideoHeliostatos', type=int) # Crear el argumento 5: umbral o nivel de color mínimo del vídeo de helióstatos a partir del cual podría estar detectándose un helióstato. parser.add_argument('numeroHelios type=int) # Crear el argumento 6: número máximo de helióstatos a detectar y analizar en cada fotograma del vídeo de helióstatos. args = parser.parse_args() # Devuelve información de los parámetros definidos previamente.

Permite que a la hora de ejecutar el programa desde la terminal de comandos de Windows, solicite al usuario la ruta del vídeo a cargar del sistema, el ancho y alto mínimos del helióstato para ser detectado y analizado por el programa, el umbral (o tonalidad del color) del vídeo a partir del cual el programa detectará los helióstatos, y el número máximo de helióstatos que el programa deberá detectar y analizar, para cada fotograma del vídeo de helióstatos. Por ejemplo: 'C: \Users \Pc \Desktop \TFG¿miCodigo.py Videos/varios_heliostatos.mp4 50 50 127 2'.

'parser = argparse.ArgumentParser(description='Parametros del programa.')'. Esta línea de código permite asignar un conjunto de parámetros o argumentos y de darles un nombre. Es guardado en la variable 'parser'. Partiendo de la anterior variable 'parser', se van designando los distintos argumentos, junto a sus nombres y tipos, como cadena o entero (según si la entrada del parámetro hay que escribir letras y/o caracteres, o simplemente números). 'args = parser.parse_args()'. Permite que, desde la variable 'args', cargar el argumento concreto, almacenado previamente cuando el programa solicitó al usuario los argumentos. Por ejemplo, si se desea comprobar cuál fue el ancho del helióstato que el usuario solicitó por parámetro en la consola, se cargará y obtendrá el segundo argumento con la línea de código siguiente: 'args.anchoMinimoHeliostato'. Así, el programa realizará las medidas y operaciones oportunas de acuerdo al valor de este parámetro deseado por el usuario.

Mostrar en la consola este aviso de cuando se va a ejecutar el programa. print("") print("Iniciando programa...") print("")

Al iniciar la ejecución del programa, mostrará por consola que se ha iniciado su ejecución, con el aviso 'Iniciando programa...'. Este aviso solo aparecerá una vez durante toda su ejecución.

Leer secuencia de imágenes del vídeo a partir del directorio especificado por parámetro. camara = cv2.VideoCapture(args.directorioVideoHeliostatosCargar)

Partiendo del directorio especificado por el usuario, el programa leerá y cargará el archivo concreto, que deberá ser el vídeo de helióstatos. De lo contrario, la ejecución del programa no se realizará correctamente.

Declarar estos arrays con el fin de almacenar toda la información sobre los resultados de los helióstatos analizados en el vídeo de helióstatos. # Además, son usados especialmente con el fin de mostrar, para cada información, hasta dos resultados distintos (uno para cada helióstato) en una misma línea de texto, en la consola. heliostato = [] anchoAlto = [] areaTotal = [] sumaBGRparcial = [] sumaBGRtotal = []

Arrays que permiten almacenar y mostrar en consola toda la información y resultados sobre los helióstatos analizados en el vídeo de helióstatos. El uso concreto de estos arrays se debe a que con ellos es posible mostrar en consola y para cada información, hasta dos resultados distintos (uno para cada helióstato) en una misma línea de texto, de la forma que se explicará posteriormente. Así se evita el uso de demasiadas líneas de texto en la consola, y compactar toda la información posible en una sola. Estos arrays son vaciados una vez que se haya guardado la información correspondiente al helióstato (o helióstatos) del fotograma actual (en análisis) del vídeo de helióstatos, y posteriormente mostradas dichas informaciones de ese helióstato (o helióstatos) en consola. Todo este procedimiento se repite igual para los siguientes fotogramas de dicho vídeo de helióstatos.

Iteración 'while True' para cada fotograma del vídeo, hasta completar todos los fotogramas y llegar al final del vídeo (cambiaría automáticamente de True a False y el bucle 'while' finaliza). while True:

A partir de aquí y prácticamente hasta el final del código, todas las demás instrucciones se aplicarán para cada fotograma del vídeo de helióstatos.

Obtener frame. Para ello, se toma un fotograma del vídeo, se guarda en 'frame', y si se ha hecho esta acción correctamente, 'grabbed' valdrá true (verdadero), y viceversa. (grabbed, frame) = camara.read()

Si se ha llegado al final del vídeo, romper la ejecución de este bucle 'while' y finalizar el programa. if not grabbed: break

Estas líneas de código se encargarán de indicar al programa si quedan más fotogra-

mas por leer o no del vídeo de helióstatos, y así saber hasta cuándo dicho programa se mantendría en ejecución. Primero, 'camara.read()' obtiene el fotograma actual del vídeo, y lo guarda en la variable 'frame'. Si se ha hecho esta acción correctamente (debido a que quedan más fotogramas por leer del vídeo y no se ha llegado al final del mismo), 'grabbed' (justo al lado izquierdo de 'frame') valdrá 'true', y viceversa. Si se ha llegado al final del vídeo, 'grabbed' valdrá 'false' (porque ya no quedan más fotogramas por leer), y romperá la ejecución del presente bucle 'while' para que deje de leer más fotogramas del vídeo y finalice la ejecución de este programa. Esta ruptura es producida porque se cumple la condición de 'if not grabbed', así que desencadena la instrucción 'break'.

Convertir a escala de grises el fotograma actual del vídeo. Para ello, con la variable 'frame' (fotograma del vídeo) capturada anteriormente, se llama a la función 'cv2.COLOR_BGR2GRAY'. img = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

Aplicar un umbral a ese fotograma del vídeo. Parámetros de este método: imagen fuente en escala de grises, valor de umbral para clasificar los valores de píxeles de esa imagen, # valor máximo a ser representado si el valor del píxel supera al valor del umbral, aplicar un tipo concreto de umbralización (0 porque no se desea hacer esto). # NOTA: la variable 'ret' que recibe como resultado en este método no es usada en este programa así que se puede ignorar, esto es debido a que no se está aplicando umbralización de Otsu. ret, thresh = cv2.threshold(img, args.umbralVideoHeliostatos, 255, 0)

cv2.imshow("Camara2", thresh) # Mostrar vídeo umbralizado en una ventana.

cv2.waitKey(1) # El programa hará una pequeña pausa (1 milisegundo) para que de tiempo a que se muestren los vídeos y fotogramas en las dos ventanas que se han creado en este código para tal fin.

Buscar y detectar todos los contornos o helióstatos del fotograma actual del vídeo. # Parámetros del siguiente método: imagen umbralizada, devolver todos los contornos y crear una lista completa de jerarquía de familia, marcar la mínima cantidad de puntos (no todos) # que forman (delimitan) la figura (helióstato). Argumentos que devolverá dicho método: imagen fuente (sobra), modo de devolución del contorno, método de aproximación del contorno (sobra). im2, contours, hierarchy = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

Las siguientes líneas de código se encargarán de convertir cada fotograma del vídeo a escala de grises, aplicarle un umbral y detectar los contornos o helióstatos, así como mostrar en pantalla la visualización en vivo (al mismo tiempo que la ejecución del programa) del vídeo de helióstatos umbralizado.

Convertir a escala de grises el fotograma actual del vídeo. Basta con tomar la ante-

rior variable ‘frame’ (fotograma actual del vídeo a color), y aplicar la línea de código ‘cv2.COLOR_BGR2GRAY’. El fotograma convertido a escala de grises se guarda en la variable resultado ‘img’. Aplicar un umbral al fotograma actual del vídeo en escala de grises (variable ‘img’ anterior). Un vídeo umbralizado en escala de grises permite diferenciar únicamente dos niveles de color: gris oscuro y gris claro. Así se facilitan las tareas de análisis y detección de contornos (helióstatos) en el vídeo. En este caso, el umbral definido ha sido de ’args.umbralVideoHeliostatos’, siendo esta expresión el valor numérico proporcionado por parámetro por el usuario al ejecutar el programa (por ejemplo, 127), y el máximo típico de 255. Es decir, si el píxel del vídeo es gris oscuro, en una escala de grises 0-127, será tratado y pintado como negro. En otros casos (128-255), como blanco. Así para todos los píxeles de cada fotograma del vídeo, quedando un vídeo en blanco y negro puros. Blanco es el helióstato, y negro el fondo. El fotograma umbralizado se guarda en la variable resultado ‘thresh’. Después, con el método ‘imshow’, se muestra en tiempo de ejecución y en una ventana el vídeo de helióstatos umbralizado. Es importante realizar después de las operaciones anteriores una pausa muy breve de un milisegundo para que el programa le de tiempo a mostrar los vídeos y fotogramas actualizados en las dos ventanas: vídeo normal y umbralizado. Para ello, se usa el método ‘waitKey(1)’, siendo ‘1’ el tiempo de espera deseado en milisegundos. Buscar, detectar y delimitar todos los contornos o helióstatos del fotograma actual del vídeo umbralizado. Para ello, el método ‘findContours’ requiere de los parámetros necesarios para saber cuál es el fotograma umbralizado a tratar (variable ‘thresh’ obtenida previamente), cómo devolverá el o los resultados (en este caso devolverá todos los contornos detectados, y almacenados en lista completa de de jerarquía de familia), y cómo delimitará el contorno (en este caso usando la mínima cantidad de puntos). Los contornos detectados se guardarán en la variable resultado ‘contours’. Las otras dos variables resultado ‘im2’ e ‘hierarchy’ no son usadas para este proyecto.

Guardar en los distintos arrays los siguientes textos, para luego ser mostrados por consola junto con los respectivos resultados de los helióstatos. heliostato.append(” ”) anchoAlto.append(”Ancho y alto WH del helióstato en píxeles: ”) areaTotal.append(”Área del helióstato en píxeles: ”) sumaBGRparcial.append(”Sumatorias BGR al cuadrado de todos sus píxeles:”) sumaBGRtotal.append(”Suma total BGR al cuadrado helióstato completo: ”)

Guardar en los respectivos arrays ’heliostato’, ’anchoAlto’, ’areaTotal’, ’sumaBGRparcial’ y ’sumaBGRtotal’ los textos ’Ancho y alto’, ’Área’, ’Sumatorias BGR parcial’ y ’Sumatorias BGR total’, relacionados con los datos que se obtendrán más adelante de los helióstatos analizados en el vídeo de helióstatos.

Recorrer solo los dos primeros contornos, los más grandes (siguiente bucle ’for’), para cada fotograma del vídeo (bucle ’while’ ejecutándose actualmente). # Al no re-

correr los demás contornos, estos serán descartados porque no son muy grandes ni importantes o son falsos. # Siendo 'args.numeroHeliostatosAnalizar' el número de contornos deseado por el usuario por parámetro en la consola que se quiere analizar como máximo para cada fotograma. for i in range(0,args.numeroHeliostatosAnalizar):

Este bucle 'for' que va del número 0 al 'args.numeroHeliostatosAnalizar' (sin incluirse este último número) se encargará de analizar únicamente los 'x' primeros contornos más grandes, para cada fotograma del vídeo. De esta forma, se ignorarán los falsos contornos y menos importantes. Además, en dicho bucle se abarcan y realizan operaciones como detectar las coordenadas de cada helióstato en el vídeo, así como calcular sus áreas y sumatorias de los valores de las componentes RGB al cuadrado de todos sus píxeles. Siendo 'args.numeroHeliostatosAnalizar' el número de contornos deseado por el usuario por parámetro en la consola que se quiere analizar como máximo para cada fotograma.

Obtener las coordenadas del contorno. (x, y, w, h) = cv2.boundingRect(contours[i])
xy: coordenadas de un punto, w: ancho, h: altura.

Calcular el área del contorno numero 'i', en el fotograma actual del vídeo. 'i' es el iterador del bucle 'for' actual. area = cv2.contourArea(contours[i])

Para cada contorno, se obtendrán sus coordenadas en el vídeo, y su valor de área, aparte de mostrar el vídeo de helióstatos normal en pantalla (en tiempo de ejecución), o de actualizarlo al siguiente fotograma que se analizará.

Obtener las coordenadas del contorno usando el método 'boundingRect': XY, correspondientes a su esquina superior izquierda en el vídeo, y WH, correspondientes a su ancho (horizontal) y su altura (vertical). Es posible que si el helióstato todavía no ha terminado de entrar en el vídeo (desde el lado izquierdo), pues que no se muestre su esquina superior izquierda. Así que en este caso, se mostrará la esquina superior izquierda del helióstato mostrado en dicho vídeo hasta el momento. Respectivamente para el ancho y la altura. Para el contorno número 1 y/o 2 del fotograma actual del vídeo, el programa detectará y calculará su área total, gracias al método 'contourArea' de la biblioteca de OpenCV.

Si el contorno tiene un ancho y alto mayores a los especificados por parámetros, este será analizado y reencuadrado en un rectángulo verde en el vídeo. if (w ¿ args.anchoMinimoHeliostato and h ¿ args.altoMinimoHeliostato):

Esta condición 'if' solo será ejecutada si los valores que el usuario proporcionó por parámetro desde consola de ancho y alto del helióstato son menores al ancho y alto del helióstato que se va a analizar justo ahora. Si se cumplen todas, se accederá a este 'if' para calcular y mostrar en consola la sumatoria acumulativa de los valores de las componentes RGB al cuadrado de todos los píxeles del contorno (helióstato),

además del valor de área y de reencuadrarlo en un rectángulo verde en el vídeo. Y también, se mostrarán sus coordenadas (ubicación) en el vídeo, su ancho y alto, etcétera. Todo esto se irá aplicando a cada contorno.

Si se está analizando el contorno número uno en el fotograma actual del vídeo, hacer. if (i == 0):

Dibujar un rectángulo verde alrededor del contorno, en el vídeo. # Parámetros: fotograma actual vídeo, esquina superior izquierda, esquina inferior derecha (width: ancho, height: altura), rectángulo color verde, grosor del rectángulo 2 píxeles. cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)

Si se está analizando el helióstato número dos en el fotograma actual del vídeo (en caso de que ya exista el otro helióstato en ese mismo fotograma del vídeo), hacer. else:

En este caso, ahora se reencuadra el contorno en un rectángulo rojo, en vez de verde. Así, ambos contornos podrán ser diferenciados si se muestran en el mismo fotograma del vídeo. cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 0, 255), 2)

Aquí puede suceder los siguientes casos:

Si solo hay un contorno en el fotograma actual del vídeo, o hay dos pero relativamente juntos entre sí (rozándose, fusionándose o separándose), se reencuadrará en verde dicho contorno o doble contorno en el vídeo usando el método 'rectangle'. Si se muestran dos contornos separados entre sí en un mismo fotograma del vídeo, se analizarán primero uno y después el otro. En el vídeo, cada contorno se reencuadrará en colores distintos: el de la izquierda en verde y el de la derecha en rojo. Si en el fotograma actual del vídeo no hay contornos, no se hará ningún procedimiento. Simplemente se pasará inmediatamente al siguiente fotograma del vídeo para seguir analizando más helióstatos que pudieran existir en ellos.

Además, conforme el helióstato se vaya desplazando por el vídeo, también lo hará el rectángulo verde o rojo para mantener el reencuadre.

Respecto a los parámetros introducidos en el método 'rectangle', aclarar que 'x+w' hace referencia a la esquina superior derecha del contorno, porque a 'xy', que es su esquina superior izquierda, se le suma su ancho 'w'. Respectivamente para 'y+h' que es su esquina inferior izquierda: a 'xy' (esquina superior izquierda) se le suma su altura 'h'. La combinación de las esquinas superior derecha y la inferior izquierda, '(x+w, y+h)', resulta la esquina inferior derecha.

Leer y analizar todos los píxeles del helióstato. def vectorial(frame, x, y): # Del fotograma actual del vídeo, se leerá únicamente donde haya un helióstato (su ancho y

alto), y así con todos los helióstatos de cada fotograma del vídeo. i = frame[y+2:y+h-1, x+2:x+w-1]

Matrices BGR resultado de la lectura de ese helióstato. mB = i[:, :, 2] mG = i[:, :, 1] mR = i[:, :, 0]

Elevar al cuadrado cada dato BGR del helióstato. mB2 = np.power(mB, 2) mG2 = np.power(mG, 2) mR2 = np.power(mR, 2)

Realizar la sumatoria acumulativa de cada BGR al cuadrado de ese helióstato. sumB = np.sum(mB2) sumG = np.sum(mG2) sumR = np.sum(mR2)

Sumar las anteriores tres componentes entre sí, para obtener la sumatoria total de los valores de las tres componentes RGB entre sí de todos los píxeles al cuadrado del contorno entero. sumaRGB = sumR+sumG+sumB

Ir introduciendo en los arrays las informaciones de los resultados de los helióstatos, con el fin de mostrarlas después por consola. # Al ser arrays acumulativos, si en un mismo fotograma del vídeo se obtienen datos de dos helióstatos, para cada array se guardarán los datos de esos dos helióstatos a la vez. # De esta forma, se compactará más la información mostrada en consola al estar esta a dos columnas: helióstato verde y helióstato rojo, para cada línea de texto o array. anchoAlto.append(w) anchoAlto.append(h) anchoAlto.append(" ")

areaTotal.append(area) areaTotal.append(" ")

sumaBGRparcial.append(sumB) sumaBGRparcial.append(sumG) sumaBGRparcial.append(sumR) sumaBGRparcial.append(" ")

sumaBGRtotal.append(sumaBGR) sumaBGRtotal.append(" ")

Llamar a la función definida 'vectorial(frame, x, y)', siendo 'frame' el fotograma actual del vídeo a tratar, y XY las coordenadas de la esquina superior izquierda del helióstato. vectorial(frame, x, y)

Estos dos bucles 'for' compactados en vectores son los encargados de analizar píxel a píxel el helióstato. Primero se analizarán los píxeles de columnas, y luego los de filas. En resumen, se realizan las siguientes operaciones: medir el ancho, alto y área del helióstato en píxeles, calcular para cada píxel del helióstato las componentes RGB y RGB al cuadrado (cada componente por separado), y con estas últimas, realizar una sumatoria acumulativa de cada componente RGB por separado de todos los píxeles que componen el helióstato, y después, lo mismo pero sumando o unificando las tres componentes entre sí.

Esta sección de código se ha trabajado sobretodo con NumPy (abreviado como 'np' en el código) y con arrays, con el fin de reducir considerablemente el tiempo de

ejecución del programa. La función 'vectorial()' es llamada y con los parámetros 'fotograma actual del vídeo', y las coordenadas XY de la esquina superior izquierda del helióstato, que es desde donde se empezarán a analizar cada helióstato, hasta llegar a su esquina inferior derecha.

En la línea de código 'i = frame[y+2:y+h-1, x+2:x+w-1]', se obtendrá, de ese fotograma, el helióstato. Para ello, en la primera entrada de 'frame', que hace referencia al número de filas, se indica el número de las mismas, es decir, desde la altura máxima del helióstato 'y' hasta su base inferior 'y+h'. Y en la segunda entrada de 'frame', correspondiente al número de columnas, se proporciona también cuántas tiene: desde el lado izquierdo 'x' hasta el lado derecho del helióstato 'x+w'. Es importante no confundir la longitud/altura del helióstato ('w' y 'h', respectivamente), con las coordenadas o ubicación del helióstato en el vídeo, midiéndose desde su esquina superior izquierda ('XY'). Para seleccionar o cargar el helióstato (variable 'i') del fotograma actual del vídeo (variable 'frame'), se parte desde la esquina superior izquierda de ese helióstato con la coordenada 'X', y para definir el límite derecho (lado derecho) del helióstato, a esa 'X', que es donde está ubicado el helióstato en el vídeo, se le suma su longitud 'w'. Y respectivamente para la altura. Indicar que, con el fin de evitar leer sobre el rectángulo verde o rojo que reencuadra al helióstato en el vídeo, se le han puesto en 'frame' unos números '+2' y '-1'. Son las medidas exactas para evitar que esto suceda.

En 'mB = i[:, :, 2], mG = i[:, :, 1], mR = i[:, :, 0]', se obtendrán las salidas BGR (o números del 0-255) de ese helióstato cargado previamente en la variable 'i'. Para ello, se indica, en cada matriz 'mB', 'mG' y 'mR' (matrices de píxeles azules, verdes y rojos del helióstato, respectivamente), que se desean obtener todas las filas y columnas de 'i', es decir, todos los valores BGR del helióstato. Se indican con los dos puntos ':', en las dos primeras entradas de esas matrices. Y en la tercera entrada, que correspondería a la tercera dimensión, se obtienen los valores azules 'B' con el número 2, los verdes 'G' con el 1, y los rojos 'R' con el 0.

Con esos valores BGR ('mB', 'mG' y 'mR'), se elevarán cada uno de ellos al cuadrado con el método 'np.power'. Así, se dispondrán en 'mB2', 'mG2' y 'mR2' todos los valores BGR del helióstato elevados al cuadrado. Los valores RGB solo pueden ser representados del 0 al 255. Al elevar al cuadrado uno de estos valores, nunca se sobrepasará del 255, y pasará del 255 al cero directamente, y así sucesivamente. Esto lo realiza el programa automáticamente.

Con todos estos valores BGR del helióstato elevados al cuadrado, se realizará una sumatoria acumulativa de todos ellos, para cada componente BGR por separado. Los resultados de las sumatorias se guardarán en 'sumB', 'sumG' y 'sumR'. Se trataría de cumplir la siguiente fórmula:

E = sumatoria R al cuadrado + sumatoria G al cuadrado + sumatoria B al cuadrado Siendo 'sumatoria' la sumatoria acumulativa de todos los píxeles del helióstato.

Para cada helióstato, se obtendría entonces: Er, Eg y Eb.

Finalmente, en 'sumaRGB = sumR+sumG+sumB', esta acción consiste en sumar o unificar los resultados 'sumB', 'sumG' y 'sumR' obtenidos previamente (las anteriores sumatorias al cuadrado de todos los píxeles del contorno). Así, lo que se estaría haciendo para cada helióstato sería lo siguiente:

Etotal = Er + Eg + Eb

Es decir, obtener la sumatoria total de los valores de las tres componentes RGB entre sí de todos los píxeles al cuadrado del contorno entero.

Ir guardando los valores y resultados del helióstato en sus respectivos arrays (dependiendo del tipo de información: ancho/alto, área, etcétera), para luego mostrarlos por consola.

Cuando se analiza otro helióstato (independientemente de si se salta o no al siguiente fotograma del vídeo), los valores de área y sumatorias pasarán automáticamente a valer cero de partida. De esta forma, no se obtendrán valores de un helióstato acumulados (erróneos) del helióstato analizado previamente.

Mostrar vídeo original en una ventana/actualizar fotograma. cv2.imshow("Camara", frame)

Mostrar el vídeo de helióstatos normal en pantalla con el método 'imshow()', y reproducirse al mismo tiempo que el programa va analizando cada uno de sus fotogramas. Con 'reproducirse', quiere decir además que se actualizará o cambiará el fotograma del vídeo al actual.

Mostrar en consola el valor del área del helióstato en píxeles, su ancho y alto también en píxeles, # los valores de las sumatorias acumulativas RGB al cuadrado (cada componente por separado) de todos los píxeles del helióstato, y esto mismo pero sumando esta vez las tres componentes entre sí. print(heliostato) print(anchoAlto) print(areaTotal) print(sumaBGRparcial) print(sumaBGRtotal)

Borrar el contenido de los arrays, ya que se ha mostrado toda la información en consola del helióstato (o helióstatos) para el fotograma actual del vídeo de helióstatos. del heliostato[:] del anchoAlto[:] del areaTotal[:] del sumaBGRparcial[:] del sumaBGRtotal[:]

Mostrar en consola todos los datos obtenidos del helióstato (o helióstatos) del fotograma actual del vídeo, y que fueron almacenados (temporalmente) en los arrays

'heliostato', 'anchoAlto', 'areaTotal', 'sumaBGRparcial' y 'sumaBGRtotal'. Es decir, el valor del área del helióstato en píxeles, su ancho y alto también en píxeles, los valores de las sumatorias acumulativas RGB al cuadrado (cada componente por separado) de todos los píxeles del helióstato, y esto mismo pero sumando esta vez las tres componentes entre sí.

Después, tras mostrarse toda esta información en la consola, vaciar los arrays, con el fin de reutilizarlos para el análisis de los siguientes fotogramas.

Al alcanzar esta línea de código, ya se habrá leído y analizado el fotograma actual del vídeo. Antes de pasar al siguiente fotograma, hacer: frame_counter += 1 # Incrementar el contador de fotogramas leídos del vídeo a uno. end_time = time.time() # Obtener el tiempo de ejecución tras haber leído el fotograma actual del vídeo. fps = frame_counter / float(end_time - start_time) # Calcular los FPS (fotogramas por segundo del vídeo) dividiendo el contador de fotogramas actual por la diferencia entre ambos tiempos medidos. print("FPS:", fps) # Mostrar en consola los FPS (fotogramas por segundo del vídeo). Se mostrará cada vez que se haya analizado el fotograma actual del vídeo.

Esta sección de código se encargará de calcular los FPS (fotogramas por segundo) del vídeo de helióstatos, cada vez que se lea y analice un fotograma completo de ese vídeo (o el equivalente a haber alcanzado estas líneas de código). Para ello, se incrementa en uno el contador de fotogramas leídos, tomar el tiempo de ejecución final tras haber procesado el fotograma actual, calcular los FPS con una fórmula que consiste en dividir el contador de fotogramas leídos actual entre la diferencia del tiempo final de la lectura del fotograma actual y el comienzo de la ejecución del programa, y mostrar dicho resultado en consola, cada vez que se haya leído y procesado un fotograma del vídeo.

Cuando el bucle 'while' inicial finalice, mostrar en consola que el programa finalizó su ejecución (el vídeo fue leído y analizado completamente). print("Programa terminado.")

Cuando se hayan terminado de leer todos los fotogramas del vídeo, es decir, se ha leído el vídeo completo, se mostrará en consola 'Programa terminado.', finalizando la ejecución del programa.