

# TRABAJO FIN DE GRADO

UNIVERSIDAD DE ALMERIA

ESCUELA SUPERIOR DE INGENIERÍA

Medición de la radiación solar proyectada

**Tecnologías de la Información**

**Curso 2018/2019**

---

**Víctor Eugenio Rodríguez Ruiz**



**Directores:**  
Vicente González Ruiz  
Luis José Yebra Muñoz

# Índice general

<b>1. Descripción del problema a resolver</b>	<b>3</b>
1.1. Introducción . . . . .	3
1.2. Objetivos y justificación . . . . .	3
1.3. Helióstatos . . . . .	3
1.4. Helióstato con sensor de reflexión . . . . .	4
1.5. Funcionamiento de la central solar de receptor central o de tipo torre . . . . .	5
1.6. Ejemplos de centrales de receptor central . . . . .	7
<b>2. El lenguaje de programación Python</b>	<b>17</b>
2.1. Historia . . . . .	17
2.2. Modo interactivo . . . . .	18
2.3. Usos de Python . . . . .	19
2.4. Versiones de Python . . . . .	20
2.5. Transición de Python 2 a Python 3 . . . . .	21
<b>3. La biblioteca OpenCV</b>	<b>24</b>
3.1. Diseño del proyecto . . . . .	24
3.2. Historia . . . . .	25
3.3. Funciones . . . . .	25
3.4. Aplicaciones . . . . .	26
3.5. Lenguaje de programación . . . . .	26
<b>4. Medición de la radiación solar proyectada</b>	<b>27</b>
4.1. Variables de entorno . . . . .	27
4.2. Instalación del software Python . . . . .	27
4.3. Cómo usar el software . . . . .	35
4.4. Diagrama de flujo . . . . .	36
4.5. Funcionamiento del software y código . . . . .	36
4.6. Cronograma de tareas . . . . .	52
4.7. Descripción de los requerimientos/requisitos . . . . .	52
4.8. Entradas y salidas . . . . .	54
4.9. Análisis de los recursos consumidos . . . . .	57
4.10. Plataformas que son capaces de ejecutar el software . . . . .	78
4.11. Ejemplos de ejecuciones y comentarios sobre sus resultados . . . . .	78
4.12. Validación cualitativa de la función de estimación de potencia . . . . .	81
<b>5. Conclusiones y posibles líneas de trabajo futuro</b>	<b>85</b>

**6. Referencias bibliográficas**

**85**

# **Capítulo 1**

## **Descripción del problema a resolver**

### **1.1. Introducción**

Los sistemas de control de plantas termosolares de receptor central son sistemas complejos que tienen entre sus objetivos concentrar energía solar reflejada por el campo de helióstatos en una serie de puntos. Obtener una medida de la radiación reflejada en un punto del receptor es una tarea complicada debido a la dificultad de medir directamente dicha radiación concentrada. Este trabajo plantea como objetivo general el aproximarse a esta medida mediante la utilización del tratamiento digital de imágenes de la proyección de la radiación solar concentrada por un helióstato.

La obtención en tiempo real de parámetros matemáticos obtenidos del análisis de dichas imágenes, así como su correlación con variables físicas, permitirán una estimación óptima de la distribución de radiación solar concentrada por un helióstato en un receptor. Esta estimación de la distribución podría ser aplicada a la proyección de un conjunto de helióstatos, así como utilizada en tareas diarias de operación y mantenimiento de un campo de helióstatos.

### **1.2. Objetivos y justificación**

Este proyecto tiene como objetivo fundamental obtener los parámetros matemáticos de una imagen de ejemplo de la proyección de la radiación solar reflejada por un helióstato de la Plataforma Solar de Almería (CIEMAT) sobre una diana. Con dichos parámetros se construirá un estimador de la cantidad de radiación solar concentrada.

Para ello se utilizarán como herramientas la librería de código abierto OpenCV a través de su interfase en lenguaje Python/C/C++ y sobre el sistema operativo UNIX/Linux. Se utilizarán las primitivas que dicho sistema ofrece para obtener la medida del tiempo de cómputo de cada parámetro ante diferentes configuraciones de computador (concurrencia, paralelismo, ...). [10] [18] [24]

### **1.3. Helióstatos**

Un helióstato es un conjunto de espejos que establecen una superficie grande y se mueven sobre uno o dos ejes, normalmente en montura acimutal, lo que permite, con los movimientos apropiados, mantener el reflejo de los rayos solares que inciden sobre él, se fijen en todo momento

en un punto o superficie. Haciendo esto, los rayos que refleja el helióstato pueden ser dirigidos hacia un solo punto (o superficie) durante todo el día.

Se utilizan fundamentalmente en observaciones astronómicas para mantener fija la imagen del Sol o de un astro sobre el aparato de observación, en cuyo caso suelen ser de pequeñas dimensiones. La aplicación de este proyecto es en el uso de centrales solares termoeléctricas para concentrar la energía solar sobre el receptor, y conseguir así altas temperaturas. Estos helióstatos suelen ser grandes, llegando a tener más de 120 m<sup>2</sup>.

En experimentación y pruebas de materiales a altas temperaturas, un conjunto suficientemente alto de helióstatos puede concentrar los rayos solares hasta conseguir temperaturas de más de 2000 °C.

Los primeros helióstatos considerados como elementos industriales se desarrollaron a los inicios de la década de los ochenta para las plantas experimentales termosolares de receptor central, con el propósito de probar la viabilidad de la energía solar térmica en los procesos de producción de electricidad a escala industrial. Las figuras 1 y 2 muestran, respectivamente, un campo de helióstatos y un helióstato.

#### 1.4. Helióstato con sensor de reflexión

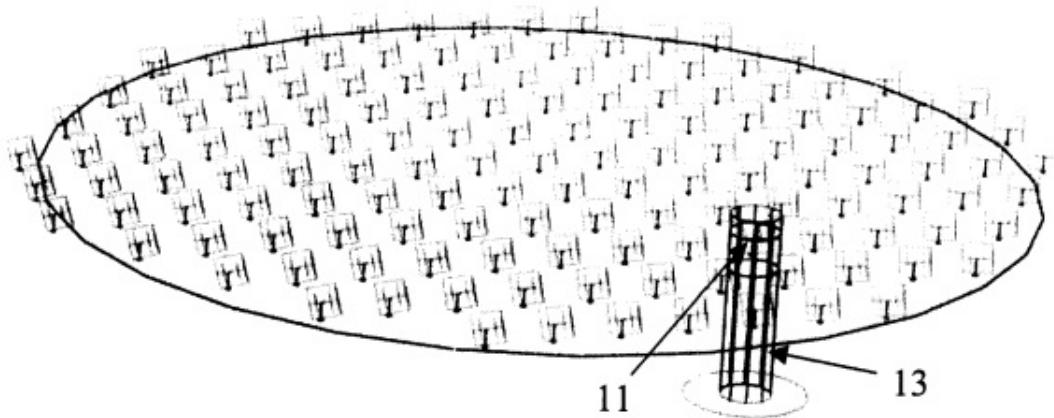


Figura 1.1: Un campo de heliostatos. [9]

El helióstato con sensor de reflexión (figura 1.1) es un helióstato perteneciente a un campo solar que refleja los haces de luz que llegan a él dotado de un mecanismo de seguimiento solar. Se trata de una invención que pertenece dentro del área de la termotecnia, al campo de la producción de energía a partir de la radiación solar.

Desde mediados del siglo XX se vienen realizando investigaciones para intentar transformar esa energía en electricidad. Es por esto que se han desarrollado helióstatos (figura 1.2) que concentran haces de luz sobre un receptor central que contiene un fluido logran alcanzar temperaturas suficientes como para producir grandes cantidades de vapor de agua que genera electricidad a través de una turbina, normalmente en un ciclo de Rankine.

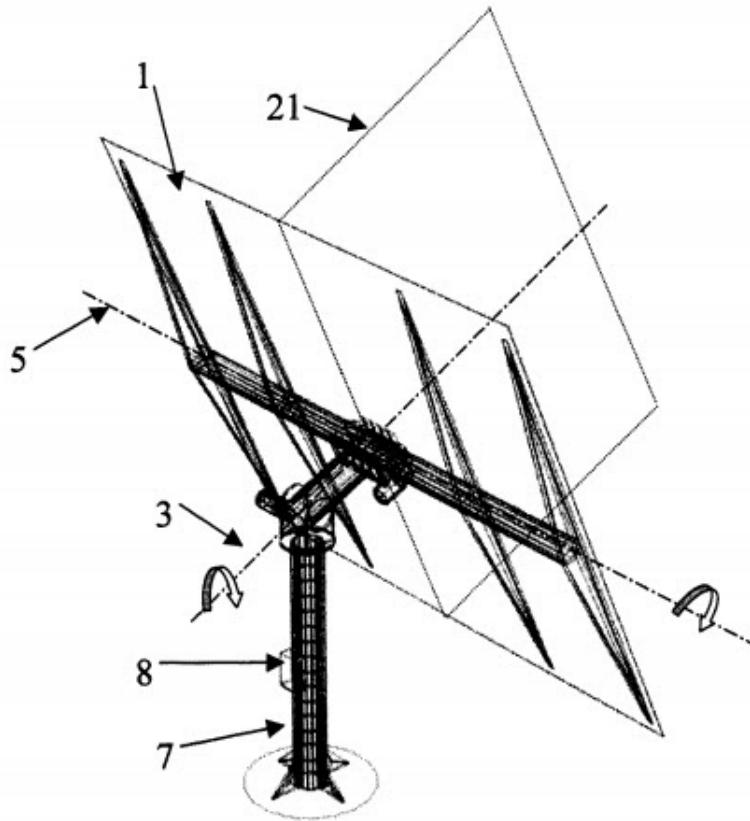


Figura 1.2: Un helióstato. [9]

### 1.5. Funcionamiento de la central solar de receptor central o de tipo torre

Una central solar de tipo torre central, está formada por un campo de helióstatos que reflejan la luz del sol y concentran los haces reflejados en una caldera situada sobre una torre de gran altura (figura 1.3).

En la caldera, el aporte calorífico obtenido es transferido a un fluido térmico. Dicho fluido es conducido hacia un generador de vapor donde transfiere el calor a agua se convierte en vapor, y acciona los álabes del grupo turbina-alternador generando energía eléctrica. El vapor es posteriormente condensado en un aerocondensador para repetir el ciclo.

La producción de una central térmica depende de una serie de factores:

- La cantidad de horas que este fluido esté expuesto al sol.
- El lugar donde la fábrica esté situada.
- La calidad de los depósitos de almacenamiento del fluido.

La energía producida, después de ser transformada, es transportada mediante líneas a la red general. [16]

La potencia de la torre central va de los 10 a los 200 MW, habiéndose instalado ya diferentes

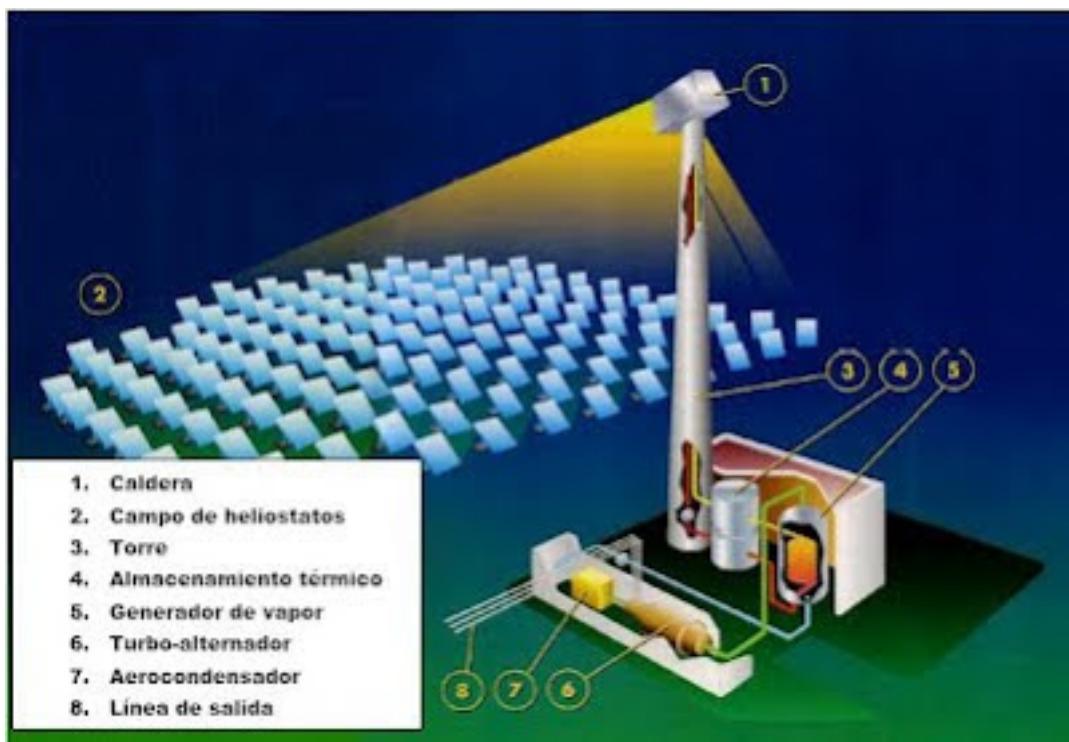


Figura 1.3: Componentes de una central solar.

plantas comerciales.

Su buen funcionamiento para obtener el máximo aprovechamiento depende de la latitud y climatología (situación geográfica) a la que está sometida, así como los procedimientos de mantenimiento, operación y control de la planta.

Hay fundamentalmente dos tipos de configuración: una en la que los heliostatos rodean completamente a la torre central, y otra en la que los heliostatos están colocados al norte o sur de la torre.

#### 1.5.1. Beneficios

Conforme se haya hecho la inversión y la instalación, ya no generará gastos mayores, el único será el del mantenimiento.

No requiere el uso de combustibles, lo que evita el riesgo de almacenar combustibles.

No contamina, ya que la energía solar no produce desechos, residuos ni vapores.

#### 1.5.2. Inconvenientes

Se debe de instalar en lugares en donde la radiación del sol sea durante el mayor tiempo posible durante todo el día.

Todo su sistema mecánico es más complejo que otros sistemas.

Se necesitan acumuladores de calor para poder usar este dispositivo para cuando no haya radiación.

En la turbina se genera la energía eléctrica, y de ahí pasa a un generador y un transformador y finalmente a una subestación; mientras que el vapor de agua pasa a un condensador y a un sistema de enfriamiento para posteriormente pasar por una bomba a la caldera donde el ciclo se repite. [7] [19] [3]

## 1.6. Ejemplos de centrales de receptor central

### 1.6.1. Plantas PS10 y PS20

#### 1.6.1.1. Planta PS10

La figura 1.4 muestra la torre de energía solar sevillana PS10.



Figura 1.4: Planta Solúcar PS10.

En Sanlúcar la Mayor, a 18 kilómetros de Sevilla, la empresa española Abengoa construyó una estación solar de generación de electricidad (figura 1.5).

Inaugurada en el año 2007, la PS10 fue la primera central solar termoeléctrica (de carácter comercial) de torre central y campo de heliostatos instalada en el mundo. Utiliza paneles de espejo para generar 11 megavatios.

Sólo PS10, PS20 y Solnovoas, que se reúnen en la conocida como Planta Solúcar, operan comercialmente un total de 183 MW, produciendo energía anual equivalente a 94.000 hogares, y evitando así la emisión de más de 114.000 toneladas anuales de CO<sub>2</sub> a la atmósfera.

La PS10 está formada por 624 heliostatos y una torre solar de 114 metros de altura. El campo de espejos o heliostatos refleja la luz solar sobre un receptor en la parte superior de la torre (figura 1.6).

Esta torre se encuentra en el centro de la estación PS10 Solúcar. En la parte superior, el receptor solar consiste en una serie de paneles de tubos que operan a muy alta temperatura y por los que circula agua a presión. Este receptor se calienta por efecto de la luz solar y genera vapor saturado a 257 °C. El vapor que se produce es almacenado parcialmente en unos tanques acumuladores para ser utilizado cuando no haya suficiente producción; el resto es enviado a accionar una turbina que genera la electricidad (figura 1.7).

Los 624 heliostatos circundantes, cada uno con una superficie de 120 metros cuadrados, producen el reflejo para que el sistema pueda convertir alrededor del 17 por ciento de la energía de la luz



Figura 1.5: Campo de heliostatos y torre.



Figura 1.6: Resplandor de la torre PS10.

solar en 11 megavatios de electricidad. Como punto de comparación, la planta PS20 (que se encuentra al lado) produce 20 MW de potencia mediante su torre de 160 metros de altura sobre un campo de 1.255 heliostatos.

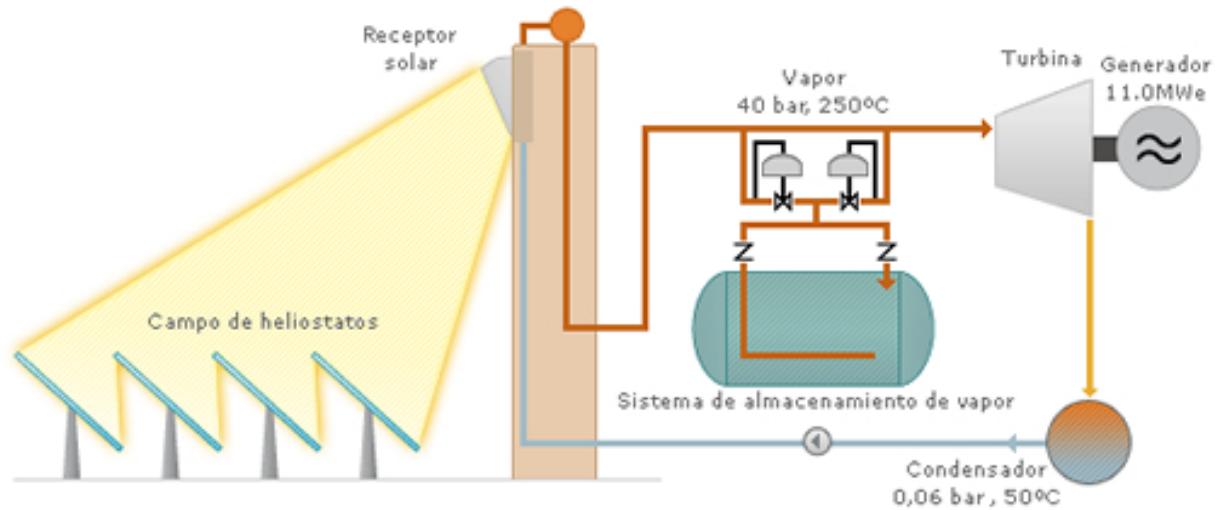


Figura 1.7: Esquema de funcionamiento de la PS10.

Por lo tanto, esta planta funciona calentando agua con luz solar y, con el vapor generado, mueve turbinas para crear electricidad (figura 1.8).



Figura 1.8: Campo y torre PS10.

[17]

#### 1.6.1.2. Planta PS20

La PS20 (Planta Solar 20) es una planta comercial situada al lado de la PS10. A diferencia de la PS10, la PS20 cuenta con 85 hectáreas y con 1.255 heliostatos que reflejan los rayos al receptor situado en la parte superior de la torre de 160 metros de altura.

La PS20 dispone con un sistema de almacenamiento de 1 hora, con un receptor más eficiente y un sistema de control y operación mejor que le permiten producir 20 MW que abastecen a 10.000 hogares anualmente (figuras 1.9 y 1.10).



Figura 1.9: Helióstato PS20.



Figura 1.10: Helióstato PS20.

[1]

### 1.6.2. Planta termosolar Gemasolar

Gemasolar es una planta de energía solar por concentración con sistema de almacenamiento térmico en sales fundidas situada en Sevilla, España. Es la primera a escala comercial en aplicar la tecnología de receptor de torre central y almacenamiento térmico en sales fundidas. La superficie de esta planta ocupa aproximadamente 158 hectáreas. Características:

- Potencia eléctrica de 19,9 MW.
- La producción neta esperada es de 110 GWh/año.
- Capaz de suministrar energía limpia y segura que reduce en más de 50.000 toneladas al año de

emisiones de CO<sub>2</sub>.

La planta está formada por:

2.650 helióstatos de 120 m<sup>2</sup> que forman círculos concéntricos alrededor de la torre que disponen de un mecanismo que posiciona con precisión la superficie de los espejos.

Una torre que en lo alto se encuentra el receptor de haz de luz compuesto por paneles.

2 tanques: uno de sales frías y otro de sales calientes.

Un cambiador de calor; un generador y transformador.

Su funcionamiento se ilustra en la siguiente imagen (figura 1.11):

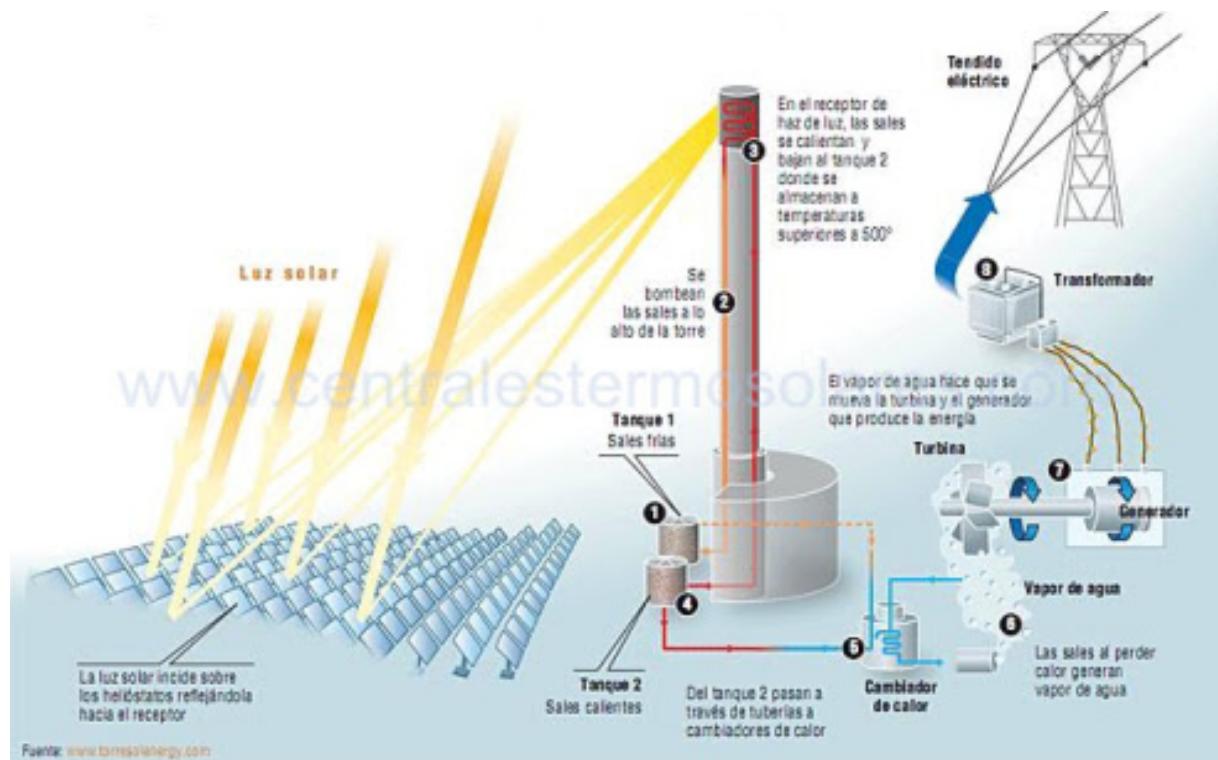


Figura 1.11: Funcionamiento de la Gemasolar.

1. Tanque 1: Sales frías.
2. Se bombean las sales a lo largo de la torre.
3. En el receptor de haz de luz, las sales se calientan y bajan al tanque 2 donde se almacenan a temperaturas superiores a 500°.
4. Tanque 2: Sales calientes.
5. Cambiador de agua.
6. Las sales al perder calor generan vapor de agua.
7. El vapor de agua hace que se mueva la turbina y el generador que produce la energía.
8. Transformador. La energía pasa al tendido eléctrico.

La planta no solo funciona como una central solar sino también como una central térmica (cuando no hay luz solar) gracias a su sistema de concentración de sales y a partir de ahí también generar electricidad. También podemos decir que gracias a esto la planta asegura obtención de energía las 24 horas del días en situaciones de baja insolación, en las madrugadas y varios meses al año.

Es tal el éxito de la planta sevillana que muchos profesionales del sector buscan aplicar las energías limpias a las pequeñas urbanizaciones y así hacer posible su aplicación a la vida cotidiana. [11]

### **1.6.3. Planta termosolar de receptor central de Ivanpah**

El Sistema de generación eléctrica solar Ivanpah es una planta termosolar concentrada en el desierto de Mojave. Está ubicado en la base de Clark Mountain en California, a través de la línea estatal desde Primm, Nevada. La planta tiene una capacidad bruta de 392 megavatios (MW). Su campo solar está formado por 173,500 helióstatos, de 14.05 m<sup>2</sup> cada uno proyectando en tres torres la energía solar reflejada. La primera unidad del sistema se conectó a la red eléctrica en septiembre de 2013 para una prueba de sincronización inicial. La instalación abrió formalmente el 13 de febrero de 2014. En 2014, fue la estación de energía solar térmica más grande del mundo.

Los campos de espejos de helióstatos enfocan la luz solar en los receptores ubicados en torres de energía solar centralizadas. Los receptores generan vapor para accionar turbinas de vapor especialmente adaptadas.

Para la primera planta, se ordenó el mayor grupo generador de turbina de vapor con energía solar, con una turbina de recalentamiento de caja única Siemens SST-900 de 123 MW. Las plantas no tienen almacenaje, y calientan el vapor a 550 °C directamente en los receptores. La potencia total instalada de la planta de Ivanpah es de 377 MW eléctricos.

La instalación, con un costo de \$ 2.2 mil millones, fue desarrollada por BrightSource Energy y Bechtel. [23]

### **1.6.4. Proyecto de Google RE<C**

Introducción.

RE<C fue una iniciativa de Google para impulsar la innovación en energía renovable, con el objetivo de hacer que la energía renovable sea lo suficientemente barata para competir cara a cara con las centrales eléctricas de carbón.

Las plantas de energía solar concentradas (CSP) usan espejos o lentes para enfocar una gran cantidad de luz solar sobre un objetivo que absorbe calor, llamado receptor. El intercambiador de calor del receptor crea vapor a alta presión, que luego impulsa una turbina para alimentar un generador eléctrico. La refrigeración por agua en spray se utiliza normalmente para condensar el vapor (figura 1.12).

La CSP modular de "torre de energía" utiliza un motor de turbina de gas más pequeño (Brayton) para realizar la conversión de energía. Los motores Brayton no necesitan refrigeración por rociado con agua y, de ese modo, se adaptan mejor a los ambientes secos del desierto.

El otro componente principal de una planta de energía CSP es un campo de espejos controlados, llamados helióstatos. Este campo tiene miles de metros cuadrados de helióstatos que concentran la energía solar en el receptor de la central eléctrica.

Este helióstato compone de los elementos descritos a continuación.



Figura 1.12: Un helióstato prototípico manteniendo la luz en el objetivo. [22]

#### 1.6.4.1. El módulo reflector

Cada helióstato tenía un espejo de enfoque de 2m x 3m articulado en la parte superior de su marco. El módulo reflector del espejo estaba hecho de vidrio.

#### 1.6.4.2. El marco del helióstato y la base

Muchos marcos y bases de helióstatos existentes son estructuras sólidas montadas sobre una base de hormigón vertido en un sitio plano. Utilizan accionamientos de precisión y grandes actuadores para realizar apuntamientos rígidos. Fue sujetado por un anclaje de tierra. Montados en el bastidor había dos accionadores de cable que usaban pequeños motores baratos.

#### 1.6.4.3. El diseño del campo

Cada uno de nuestros sistemas modulares de conversión de potencia del motor Brayton fue diseñado para producir una salida eléctrica planificada de 890kW por torre.

Establecimos un tamaño de campo de 862 helióstatos alrededor de una torre de 44 m, cada helióstato es de aproximadamente 6m<sup>2</sup>. Los helióstatos tienen disposición hexagonal.

#### 1.6.4.4. Requisitos de focalización del sistema de control

Para convertir la energía de manera eficiente, el motor Brayton requiere un receptor de cavidad de temperatura más alta que el receptor típico de una planta de vapor.

Un receptor de mayor temperatura requiere una abertura más pequeña para reducir la pérdida de calor radiante.

#### 1.6.4.5. Sistema de detección y control

El sistema de control es capaz de controlar simultáneamente los puntos de luz desde múltiples helióstatos a un alto grado de precisión a un lugar deseado en un objetivo.

Se utilizó un acelerómetro de 3 ejes montado en helióstato de bajo costo combinado con un sistema central de fotometría multiscópica para resolver las posiciones de puntos de luz individuales en el objetivo.

#### 1.6.4.6. Mitigación del viento

Las áreas de tierra grandes y planas donde es más probable que se construyan los helióstatos son también las áreas más propensas al viento.

Los helióstatos a lo largo del borde exterior de un campo protegen a los helióstatos en el medio de gran parte del impacto del viento. Las cercas de viento simples también reducen el impacto del viento en los helióstatos. [5]

### 1.6.5. Torres CESA y CRS de la Plataforma Solar de Almería

#### 1.6.5.1. La instalación CESA-1 de 7MWt



Figura 1.13: Instalación CESA. [12]

El proyecto CESA-I fue promovido por el Ministerio de Industria y Energía de España e inaugurado en mayo de 1983 para demostrar la viabilidad de las plantas solares de receptor central y para permitir el desarrollo de la tecnología necesaria. En la actualidad CESA-I ya no produce

electricidad, sino que se opera, con un alto grado de flexibilidad, como una instalación de ensayo de componentes y subsistemas como helióstatos y receptores solares.

La instalación capta la radiación solar directa por medio de un campo de 300 helióstatos (figura 1.13).

La torre es de hormigón y tiene una altura de 80 m, siendo capaz de soportar una carga de 100 toneladas. A lo largo de la torre hay tres niveles de ensayo: Una cavidad adaptada para su uso como horno solar y ensayo de materiales. Una cavidad con un banco calorimétrico de ensayo de receptores volumétricos presurizados.

Una instalación de ensayo de receptores volumétricos atmosféricos. La torre se completa con una grúa en la parte superior con 5 toneladas de capacidad y un elevador montacargas con capacidad para 1.000 kg. [14]

#### 1.6.5.2. La instalación SSPS-CRS de 2,5 MWt



Figura 1.14: Instalación CRS. [13]

La planta SSPS-CRS fue inaugurada como parte del proyecto SSPS (Small Solar Power Systems) de la Agencia Internacional de la Energía en Septiembre de 1981. Es una instalación de ensayos dedicada al ensayo de pequeños receptores solares. El campo de helióstatos está formado por 91 unidades (figura 1.14), aunque también existe un segundo campo con 20 helióstatos.

El campo de helióstatos CRS ha sido recientemente mejorado con la conversión de todos sus helióstatos en unidades autónomas. En la actualidad, la instalación CRS dispone del primer campo de helióstatos autónomos, que no precisa del uso de zanjas ni cableados.

La torre, de 43 m de altura, es metálica y dispone de tres plataformas de ensayo. La infraestructura de la torre se completa con una grúa con capacidad para 4.000 kg y un elevador de cremallera con capacidad para 1.000 kg.

Para la caracterización del mapa de flujo de la radiación solar concentrada en ambas torres, se utilizan dos sistemas de medida de flujo: directo e indirecto. El sistema de medida directo consiste en una serie de calorímetros con tiempos de respuesta de microsegundos; están dispuestos en una barra móvil y montada frente a la apertura del receptor. El sistema de medida indirecto consta

de una cámara CCD que utiliza un sensor de flujo para la calibración convirtiendo la escala de gris de las imágenes tomadas por la cámara en valores de flujo. [15]

## Capítulo 2

# El lenguaje de programación Python

### 2.1. Historia

Python fue creado a finales de los ochenta por Guido van Rossum en el Centro para las Matemáticas y la Informática (CWI, Centrum Wiskunde e Informatica), en los Países Bajos. El nombre del lenguaje proviene de la afición de su creador por los humoristas británicos Monty Python. Van Rossum es el principal autor de Python.

En 1991, van Rossum publicó el código de la versión 0.9.0 en alt.sources. En esta etapa del desarrollo ya estaban presentes clases con herencia, manejo de excepciones, funciones y los tipos modulares, como: str, list, dict, entre otros. Además en este lanzamiento inicial aparecía un sistema de módulos adoptado de Modula-3. El modelo de excepciones en Python es parecido al de Modula-3, con la adición de una cláusula else. En el año 1994 se formó comp.lang.python, el foro de discusión principal de Python, marcando un hito en el crecimiento del grupo de usuarios de este lenguaje.

Python alcanzó la versión 1.0 en enero de 1994. Una característica de este lanzamiento fueron las herramientas de la programación funcional: lambda, reduce, filter y map. La última versión liberada proveniente de CWI fue Python 1.2. En 1995, van Rossum continuó su trabajo en Python en la Corporation for National Research Initiatives (CNRI) en Reston, Virginia, donde lanzó varias versiones del software.

Durante su estancia en CNRI, van Rossum lanzó la iniciativa Computer Programming for Everybody (CP4E), con el fin de hacer la programación más accesible a más gente, con un nivel de 'alfabetización' básico en lenguajes de programación.

En el año 2000, el equipo principal de desarrolladores de Python se cambió a BeOpen.com para formar el equipo BeOpen PythonLabs. CNRI pidió que la versión 1.6 fuera pública, continuando su desarrollo hasta que el equipo de desarrollo abandonó CNRI; su programa de lanzamiento y el de la versión 2.0 tenían una significativa cantidad de traslapo. Python 2.0 fue el primer y único lanzamiento de BeOpen.com. Después que Python 2.0 fuera publicado por BeOpen.com, Guido van Rossum y los otros desarrolladores de PythonLabs se unieron en Digital Creations.

Python 2.0 tomó una característica mayor del lenguaje de programación funcional Haskell: listas por comprensión. La sintaxis de Python para esta construcción es muy similar a la de Haskell, salvo por la preferencia de los caracteres de puntuación en Haskell, y la preferencia de Python por palabras claves alfabéticas. Python 2.0 introdujo además un sistema de recolección de basura

capaz de recolectar referencias cíclicas.

Posterior a este doble lanzamiento, y después que van Rossum dejara CNRI para trabajar con desarrolladores de software comercial, quedó claro que la opción de usar Python con software disponible bajo GNU GPL era muy deseable. La licencia usada entonces, la Python License, incluía una cláusula estipulando que la licencia estaba gobernada por el estado de Virginia, por lo que, bajo la óptica de los abogados de Free Software Foundation (FSF), se hacía incompatible con GPL. CNRI y FSF se relacionaron para cambiar la licencia de software libre de Python para hacerla compatible con GPL. En el año 2001, van Rossum fue premiado con FSF Award for the Advancement of Free Software.

Python 1.6.1 es esencialmente el mismo que Python 1.6, con unos pocos arreglos de bugs, y con una nueva licencia compatible con GPL.

Python 2.1 fue un trabajo derivado de Python 1.6.1, así como también de Python 2.0. Su licencia fue renombrada a: Python Software Foundation License. Todo el código, documentación y especificaciones añadidas, desde la fecha del lanzamiento de la versión alfa de Python 2.1, tiene como dueño a Python Software Foundation (PSF).

Una innovación mayor en Python 2.2 fue la unificación de los tipos en Python (tipos escritos en C), y clases (tipos escritos en Python) dentro de una jerarquía. Esa unificación logró un modelo de objetos de Python puro y consistente. También fueron agregados los generadores que fueron inspirados por el lenguaje Icon.

Las adiciones a la biblioteca estándar de Python y las decisiones sintácticas fueron influenciadas fuertemente por Java en algunos casos: el package logging, introducido en la versión 2.3, está basado en log4j; el parser SAX, introducido en 2.0; el package threading, cuya clase Thread expone un subconjunto de la interfaz de la clase homónima en Java.

## 2.2. Modo interactivo

El intérprete de Python estándar incluye un modo interactivo en el cual se escriben las instrucciones en una especie de intérprete de comandos: las expresiones pueden ser introducidas una a una, pudiendo verse el resultado de su evaluación inmediatamente, lo que da la posibilidad de probar porciones de código en el modo interactivo antes de integrarlo como parte de un programa. Esto resulta útil tanto para las personas que se están familiarizando con el lenguaje como para los programadores más avanzados.

Existen otros programas, tales como IDLE, bpython o IPython, que añaden funcionalidades extra al modo interactivo, como el autocompletado de código y el coloreado de la sintaxis del lenguaje.

Ejemplo del modo interactivo:

```
\textgreater\textgreater\textgreater 1 + 1
2
\textgreater\textgreater\textgreater a = range(10)
\textgreater\textgreater\textgreater print(list(a))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
\cite{Wikipedia1WebSite}
```

### 2.3. Usos de Python

- Es una gran multiplataforma.

Python es un lenguaje de programación interpretado, por lo que funciona en cualquier tipo de sistema que integre su interpretador. A parte de esta ventaja, Python nos ofrece dialectos como el ya conocido Jython, que se utiliza para escribir en Java.

- Frameworks de gran utilidad.

Python no sólo es multiplataforma y multiparadigma, sino que también nos servirá para desarrollar cualquier tipo de vía, como por ejemplo web o móvil. Para que esto se lleve a cabo, este lenguaje de programación cuenta con frameworks de gran calibre, los cuales auxilan desde el desarrollo web, hasta el desarrollo de juegos o algoritmos científicos de cálculos avanzados.

- Es libre y nos ofrece código abierto.

Si hablamos de la licencia que posee, ésta es Python Software Foundation License, licencia muy parecida a la de GPL, pero encontrando la excepción de que se pueden distribuir los binarios del lenguaje sin tener que anexar las fuentes.

- Empresas de alto prestigio utilizan Python para programar todo tipo de aplicaciones y servicios.

Python se encuentra en multitud de aplicaciones y servicios que usamos habitualmente. Ostenta una gran lista de usuarios de gran calibre como Google, YouTube o Facebook, los cuales utilizan este lenguaje de programación. Poco a poco Python va ganando territorio y, entre los entendidos, se ha convertido en uno de los lenguajes más solicitados y, sobretodo, más esenciales del momento.

- Gran calidad en su sintaxis.

La sintaxis que nos ofrece este lenguaje de programación es una de sus características más notorias. En Python, un bloque de código interno como puede ser un ‘if’, se crea a través de indentaciones, lo que fuerza al desarrollador a indentar su código fuente garantizando una legibilidad notoria.

Otras de sus funciones son las de reducir el uso de caracteres como ‘=’, ‘{’, ‘}’, entre otros, y de ser capaz de escribir un ‘for’ que testeé una determinada secuencia.

- Python: programación orientada a objetos.

Si hablamos de programación orientada a objetos, podemos decir que nos encontramos ante un paradigma que propone modelar todo en función a clases y a objetos, el cual nos ofrece un uso de conceptos de cohesión, polimorfismo, herencia, abstracción y mucho más.

Este paradigma de programación se utiliza para tratar el rápido aumento en el tamaño y la complejidad de los sistemas de software, y facilitar la modificación de esos grandes y complicados sistemas a lo largo del tiempo.

- Nos ofrece un tipado dinámico fuerte.

Por último, cabe destacar la fácil atribución de una variable que nos ofrece a cualquier tipo de valor, y lo mejor de todo, en cualquier lugar de su código fuente. [2]

## 2.4. Versiones de Python

Guido van Rossum ideó el lenguaje Python a finales de los 80 y comenzó a implementarlo en diciembre de 1989. En febrero de 1991 publicó la primera versión pública, la versión 0.9.0. La versión 1.0 se publicó en enero de 1994, la versión 2.0 se publicó en octubre de 2000 y la versión 3.0 se publicó en diciembre de 2008. El desarrollo de Python lo lleva a cabo un colectivo de programadores que trabaja bajo el paraguas de la fundación Python Software Foundation, pero Guido van Rossum sigue dirigiendo el desarrollo de Python.

Las versiones de Python se identifican por tres números X.Y.Z, en la que:

- X corresponde a las grandes versiones de Python (1, 2 y 3), incompatibles entre sí:
- Los principales cambios introducidos en Python 2 fueron las cadenas Unicode, las comprensiones de listas, las asignaciones aumentadas, los nuevos métodos de cadenas y el recolector de basura para referencias cíclicas.
- Los principales cambios introducidos en Python 3 fueron la separación entre cadenas Unicode y datos binarios, la función print(), cambios en la sintaxis, tipos de datos, comparadores, etc.
- Por el momento, no hay planes de crear una nueva versión Python 4, incompatible con las anteriores.
- Y corresponde a versiones importantes en las que se introducen novedades en el lenguaje pero manteniendo la compatibilidad (salvo excepciones).
- Desde hace unos años, las versiones X.Y se publican aproximadamente cada año y medio y se mantienen durante cinco años, excepto la versión 2.7, que se mantendrá por lo menos durante diez años, hasta 2020.
- Z corresponde a versiones menores que se publican durante el período de mantenimiento, en las que sólo se corrigen errores y fallos de seguridad.
- Normalmente, se publica una última versión X.Y.Z justo antes de que una versión X.Y deje de mantenerse. Algunas empresas comerciales ofrecen el mantenimiento de versiones antiguas una vez acabado el mantenimiento oficial.

La figura 2.1 muestra la fecha de publicación de las versiones principales de Python, en cada una de las tres grandes versiones, Python 1, Python 2 y Python 3. Las versiones indicadas con punto rojo se consideran obsoletas, las versiones indicadas con punto negro siguen publicando actualizaciones, las versiones indicadas con punto blanco corresponden a versiones futuras con fechas ya previstas.

La figura 2.2 muestra la fecha de publicación de las últimas versiones menores de Python. Las versiones indicadas en rojo se consideran obsoletas, las versiones indicadas con punto blanco corresponden a versiones futuras con fechas ya previstas.

Es posible tener instalados en el ordenador varias versiones de Python pero, salvo que sea necesario para la ejecución de programas o paquetes incompatibles, se recomienda instalar siempre la última versión disponible.

- Referencias:
- Principales novedades en Python (documentación oficial):
- Python 2.X: 2.0 - 2.1 - 2.2 - 2.3 - 2.4 - 2.5 - 2.6 - 2.7

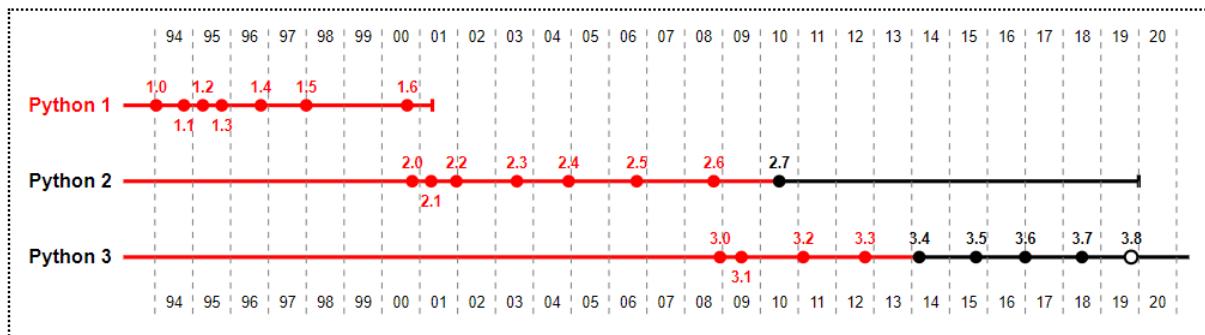


Figura 2.1: Versiones de Python.

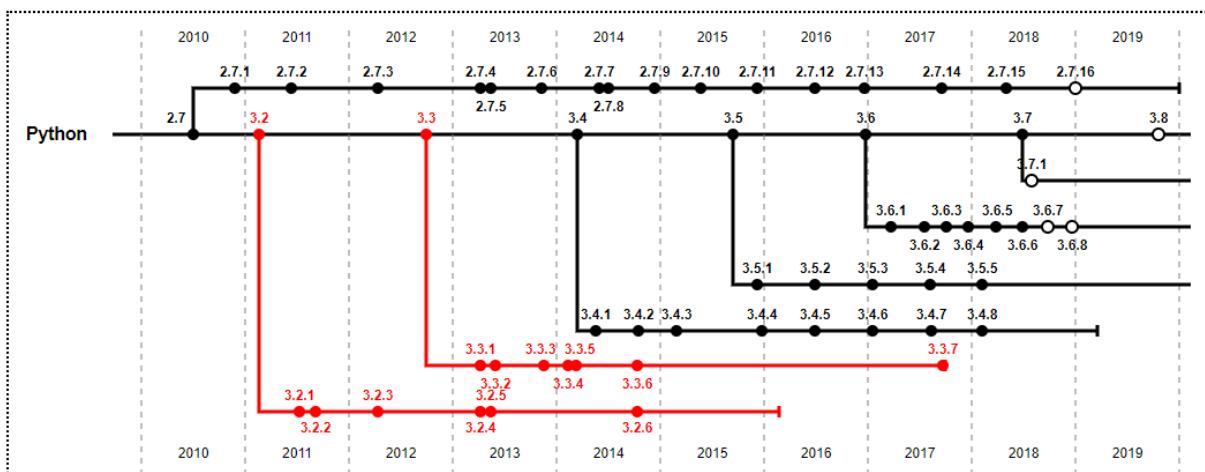


Figura 2.2: Versiones de Python.

- Python 3.X: 3.0 - 3.1 - 3.2 - 3.3 - 3.4 - 3.5 - 3.6 - 3.7
- Planificación de la publicación de cada versión (release schedules):
- Python 2.X: 2.6 - 2.7 - 2.8
- Python 3.X: 3.0 - 3.1 - 3.2 - 3.3 - 3.4 - 3.5 - 3.6 - 3.7 - 3.8
- Sobre los cambios en Python 3 (Nick Coghlan)
- Cool New Features in Python 3.7, de Geir Arne Hjelle (27/06/18)

## 2.5. Transición de Python 2 a Python 3

La transición de Python 2 a Python 3 ha resultado mucho más costosa de lo esperado, debido a que Python 3 introdujo muchos cambios en el lenguaje y obligaba a reescribir prácticamente todos los programas (aunque se han creado herramientas para ayudar en ese proceso).

La intención inicial era haber terminado Python 2 con la versión 2.6, pero en 2010 se tuvo que publicar la versión 2.7, incorporando parte de las novedades de Python 3. Además, el período de mantenimiento de Python 2.7 se tuvo que duplicar de los cinco años habituales a diez, hasta 2020.

Un primer obstáculo en el proceso de transición de Python 2 a Python 3 ha sido la propia disponibilidad de Python 3 en las distribuciones GNU/Linux. Muchas herramientas internas de las distribuciones están escritas en Python y su conversión de Python 2 a Python 3 no era fácil, por lo que las distribuciones no podían pasar simplemente de incluir una versión a otra.

Hasta 2015, Python 2 siguió siendo la versión predeterminada de Python en la mayoría de distribuciones GNU/Linux (aunque se podía instalar Python 3 sin problemas en ellas). Felizmente, esta situación está en vías de solución:

- Fedora hizo la transición a Python 3.4 en Fedora 23 (publicada en noviembre de 2015) [wiki de Fedora]. Eso quiere decir que RedHat Linux 8 (sin fecha de publicación prevista) usará Python 3. Posteriormente, Fedora 24 (publicada en junio de 2016) incluyó Python 3.5 y Fedora 26 (publicada en julio de 2017) incluyó Python 3.6.
- Ubuntu hizo la transición a Python 3.5 en Ubuntu 16.04 (publicada en abril de 2016) [wiki de Ubuntu] e incluye Python 3.6 en Ubuntu 18.04 (publicada en abril de 2018) [wiki de Ubuntu].
- En abril de 2015 Debian empezó a discutir los planes de transición a Python 3. Debian 9 (publicado en junio de 2017) incluye Python 2.7.13 y Python 3.5.3 y Debian 10 (que se espera publicar en 2019) debería pasar completamente a Python 3 [artículo de Linux Week News del 29/04/15].
- OpenStack, una importante plataforma de virtualización, incluyó soporte para Python 3 (concretamente, Python 3.5) en la versión Pike, publicada en agosto de 2017. Aunque las nuevas versiones de las distribuciones ya incluyen Python 3, las distribuciones que incluyen Python 2 seguirán instaladas en servidores durante bastantes años. Ese es el motivo por el que se vayan a seguir publicando actualizaciones de seguridad de Python 2.7 hasta 2020, como mínimo.

Un segundo obstáculo en el proceso de transición de Python 2 a Python 3 (y que ha afectado además a todos los sistemas operativos) ha sido la disponibilidad de las bibliotecas.

Python cuenta con un gran número de bibliotecas, cuyo repositorio oficial es PyPI (Python Package Index), que facilitan la programación de aplicaciones complejas. Cuando se publicó Python 3, la inmensa mayoría de bibliotecas sólo estaban disponibles para Python 2 y, lógicamente, si un programa necesitaba alguna biblioteca que sólo estaba disponible para Python 2, el programa no se podía pasar tampoco a Python 3.

Poco a poco, la mayoría de bibliotecas de Python han ido publicando versiones para Python 3, por lo que este problema también está en vías de solución.

En marzo de 2016, un estudio de empleados de Microsoft señalaba que por aquel entonces algo más del 50 % de las bibliotecas estaban disponibles tanto para Python 2 como para Python 3, un 25 % estaban disponibles sólo para Python 2 y un poco menos del 25 % estaban disponibles sólo para Python 3, pero la tendencia parecía indicar que a mediados de 2016 Python 3 pasaría a ser la versión más popular.

Varias páginas web hacen un seguimiento de la compatibilidad con Python 3 de las bibliotecas más populares:

- Python 3 Wall of Superpowers (parece que desde abril de 2018 esta web ya no actualiza los datos) y Python 3 Readiness muestran cómo la inmensa mayoría de las bibliotecas más populares ya están disponibles en Python 3.

- La distribución GNU/Linux Fedora lleva el seguimiento de los paquetes de Python que forman parte de la distribución: paquetes y evolución.
- Drop Python muestra las bibliotecas más populares han dejado de dar soporte a las versiones obsoletas de Python (Python 2.6, Python 3.2 y Python 3.3). [8]

## Capítulo 3

# La biblioteca OpenCV

OpenCV (Open Source Computer Vision Library) se lanza bajo licencia BSD y, por lo tanto, es gratuito tanto para uso académico como comercial. Tiene interfaces C++, Python y Java y es compatible con Windows, Linux, Mac OS, iOS y Android. OpenCV fue diseñado para la eficiencia computacional y con un fuerte enfoque en aplicaciones en tiempo real. Escrito en C/C++ optimizado, la biblioteca puede aprovechar el procesamiento de múltiples núcleos. Habilitado con OpenCL, puede aprovechar la aceleración de hardware de la plataforma informática heterogénea subyacente.

Adoptado en todo el mundo, OpenCV tiene más de 47 mil personas de usuarios y una cantidad estimada de descargas que supera los 14 millones. El uso va desde el arte interactivo hasta la inspección de minas, la costura de mapas en la web o la robótica avanzada. [10]

OpenCV, es una biblioteca informática de código abierto desarrollado originalmente por la visión de Intel. Es gratuito para uso comercial y la investigación bajo un licencia BSD. La biblioteca es multiplataforma y funciona en Mac OSX, Windows y Linux. Se centra principalmente hacia procesamiento imagen tiempo real como tal si encuentra Intel Integrated Performance Primitives sobre el sistema, utilizará estas rutinas optimizado comerciales a acelerarse.

### 3.1. Diseño del proyecto

Lanzado oficialmente en 1999, el proyecto fue inicialmente OpenCV iniciativa de investigación del Intel para avanzar en la CPU aplicaciones de uso intensivo, que forma parte de una serie de proyectos, entre ellos en tiempo real el trazado de rayos y las paredes de visualización 3D. Los principales contribuyentes al proyecto incluye equipo de Intel la biblioteca de rendimiento. En los primeros días de OpenCV, los objetivos del proyecto se describe como:

1. Avanzar en la investigación, proporcionando la visión no sólo libre, sino también el código optimizado para las infraestructuras básicas de la visión.
2. Difundir el conocimiento de la visión de aportar una infraestructura común que los desarrolladores pueden construir, para que el código sería más fácil lectura e intransferible.
3. Promover la visión basada en aplicaciones comerciales, haciendo portátiles, rendimiento optimizado de código disponible de forma gratuita con una licencia que no requiere ser abierto o gratuitas.



Figura 3.1: Logotipo de OpenCV.

### 3.2. Historia

Desde que apareció su primera versión alfa en el mes de enero de 1999, se ha utilizado en infinidad de aplicaciones. Desde sistemas de seguridad con detección de movimiento, hasta aplicativos de control de procesos donde se requiere reconocimiento de objetos. Esto se debe a que su publicación se da bajo licencia BSD, que permite que sea usada libremente para propósitos comerciales y de investigación con las condiciones en ella expresadas.

OpenCV es multiplataforma, existiendo versiones para GNU/Linux, Mac OSX y Windows. Contiene más de 500 funciones que abarcan una gran gama de áreas en el proceso de visión, como reconocimiento de objetos (reconocimientos facial), calibración de cámaras, visión estéreo y visión robótica.

El proyecto pretende proporcionar un entorno de desarrollo fácil de utilizar y altamente eficiente. Esto se ha logrado, realizando su programación en código C y C++ optimizados, aprovechando además las capacidades que proveen los procesadores multi núcleo. OpenCV puede además utilizar el sistema de primitivas de rendimiento integradas de Intel, un conjunto de rutinas de bajo nivel específicas para procesadores Intel (IPP).

### 3.3. Funciones

- Captura en tiempo real.
- Importación de archivos de vídeo.
- El tratamiento básico de imágenes (brillo, contraste, umbral).
- Detección de objetos (cara, cuerpo).
- Blob detección.

### **3.4. Aplicaciones**

- OpenCV ha sido usado en el sistema de visión del vehículo no tripulado Stanley de la Universidad de Stanford, el ganador en el año 2005 del Gran desafío DARPA.
- OpenCV se usa en sistemas de vigilancia de vídeo.
- OpenCV es la clave en el programa Swistrack, una herramienta de seguimiento distribuida.

### **3.5. Lenguaje de programación**

La biblioteca fue originalmente escrita en C y esta interfaz C hace OpenCV portátil para algunas plataformas específicas, tales como procesadores de señal digital.

OpenCV incluye tanto la interfaz de C tradicionales, así como una nueva interfaz C++, que busca reducir el número de líneas de código necesarias al código de la funcionalidad de la visión, así como reducir los errores comunes de programación, tales como la memoria fugas que pueden surgir al utilizar OpenCV en la mayoría de los nuevos desarrollos y algoritmos OpenCV C.

Se desarrollan en el interfaz C++, es mucho más difícil para las envolturas en otras lenguas de código C++ en lugar de código en C. [20] [21] [25]

## Capítulo 4

# Medición de la radiación solar proyectada

### 4.1. Variables de entorno

Antes de trabajar el código, hay que agregar unas variables de entorno en el sistema.

Para ello, se hace clic en Inicio, situado en la esquina inferior izquierda de la pantalla, y teclear ‘Variables de entorno’ para buscar esta configuración. Hacer clic en ‘Editar las variables de entorno del sistema’, el resultado que aparecerá. Y en la ventana que se abrirá, hacer clic en ‘Variables de entorno’.

Por si acaso, en mi caso particular, he agregado las variables de entorno tanto para mi propio usuario como para el sistema completo. Las variables de entorno que deberán agregarse son:

Variable: Python35-32.

Valor: C:\Users\Pc\AppData\Local\Programs\Python\Python35-32.

Variable: Python36.

Valor: C:\Users\Pc\AppData\Local\Programs\Python\Python36.

(Siendo ‘Pc’ el nombre de mi usuario; aquí se debe introducir el nombre de usuario correspondiente.)

Para agregar una variable de entorno, hacer clic en ‘Nueva...’. Hay dos botones ‘Nueva...’. El que se ubica arriba se agrega en el usuario específico del sistema, y el que se ubica abajo se agrega para todo el sistema en general. En ‘Nombre de la variable’ y ‘Valor de la variable’, introducir los datos mencionados anteriormente, y hacer clic en ‘Aceptar’.

### 4.2. Instalación del software Python

Figura 4.1. Para este trabajo de fin de grado, se necesitaba desarrollar un software usando un programa de ordenador: Python. La última versión disponible a la hora de realizar dicho trabajo era la 3.6.4. Se ha utilizado además la versión de 64 bits, debido a que mi PC donde desarrollé el programa monta esta arquitectura de CPU.

A continuación, se explicará cómo descargar Python 3.6.4.

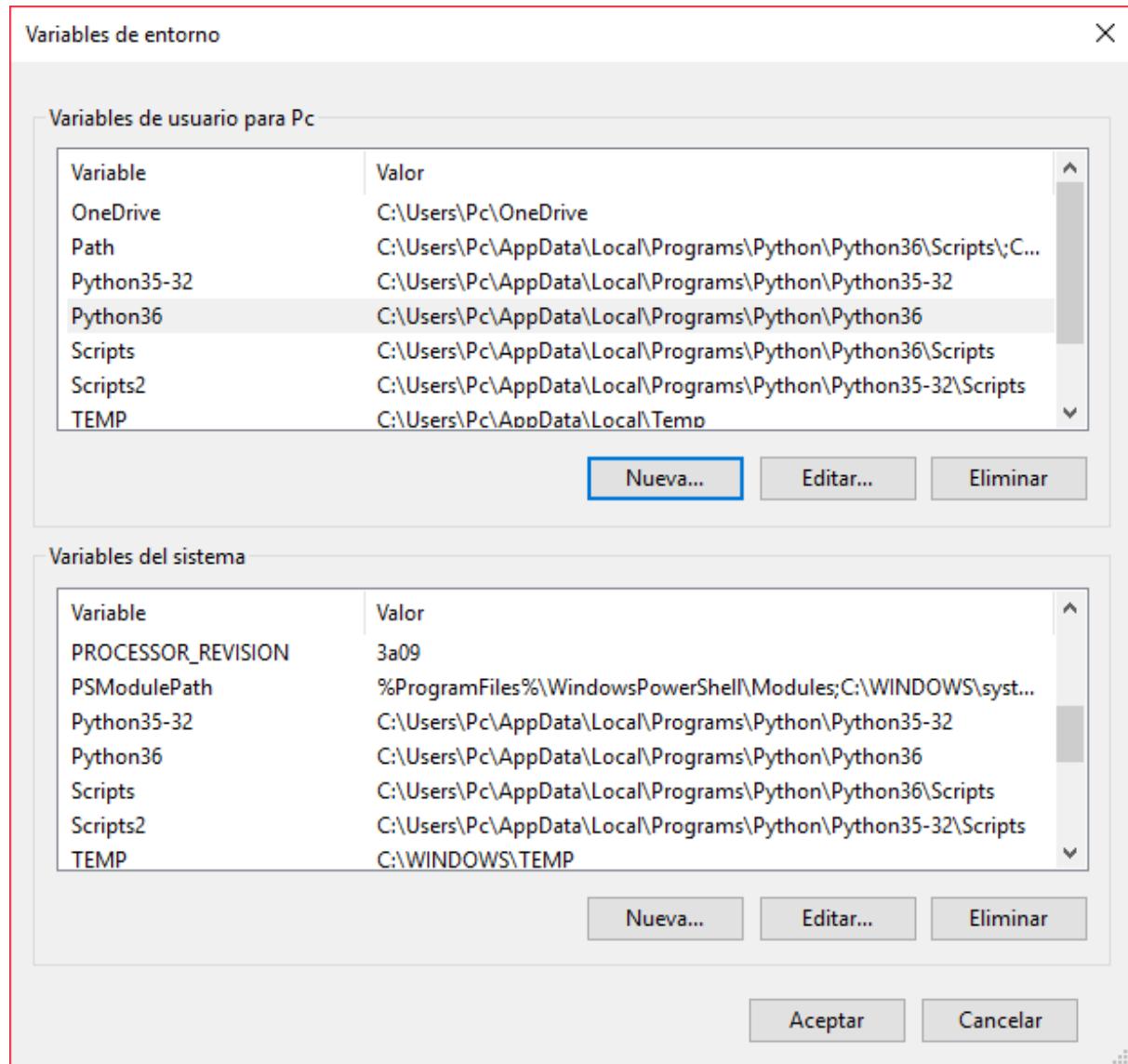


Figura 4.1: Configuración de las variables de entorno en Windows.

Figura 4.2. Abrir un explorador de internet como Google Chrome, y buscar en Google ‘Instalar Python 3.6.4 64 bits’. Pinchar en el resultado ‘Python Release Python 3.6.4 — Python.org’.

Figura 4.3. Una vez se haya accedido a esa página oficial de Python, realizar scroll hacia abajo para acceder a los archivos disponibles de esa versión de Python a descargar en el PC.

Figura 4.4. Pinchar en ‘Windows x86-64 executable installer’ para descargar en el PC el instalador de Python, el cual se encargará de instalar este programa. La descarga se completará en cinco segundos. Pinchar en el archivo ‘EXE’ mostrado en la esquina inferior izquierda para abrir

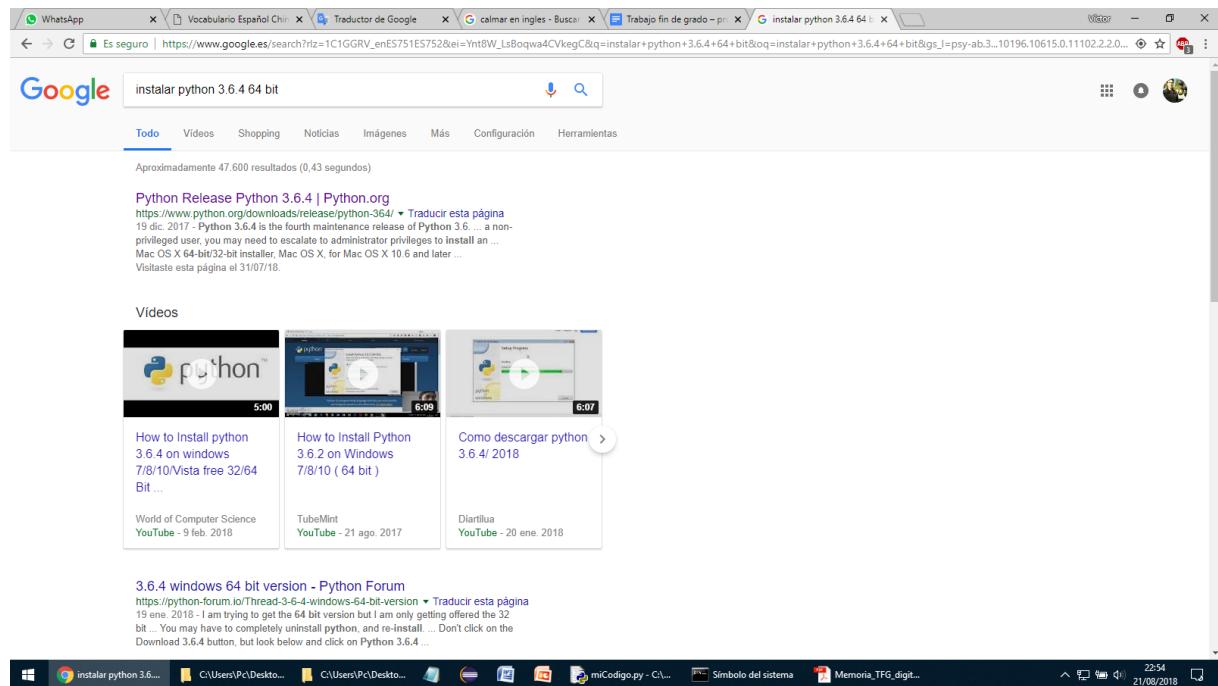


Figura 4.2: Navegador Google Chrome donde se ha buscado la versión 3.6.4 de 64 bits de Python.

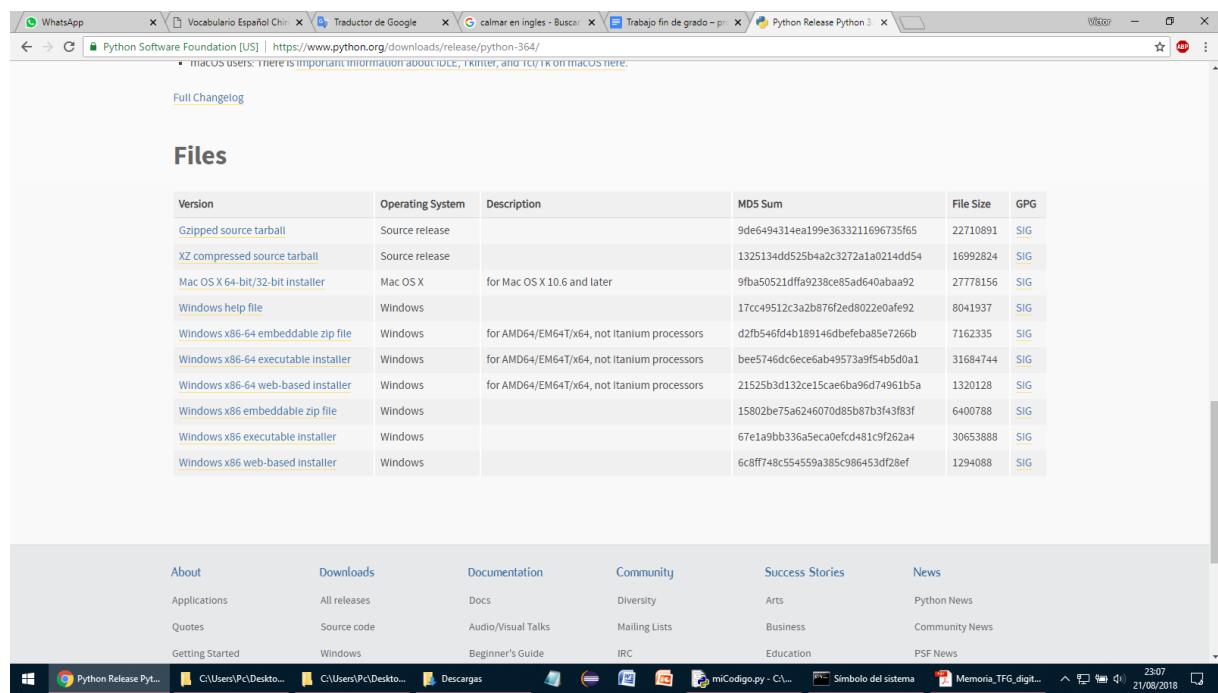


Figura 4.3: Archivos de la versión elegida de Python.

el instalador de Python.

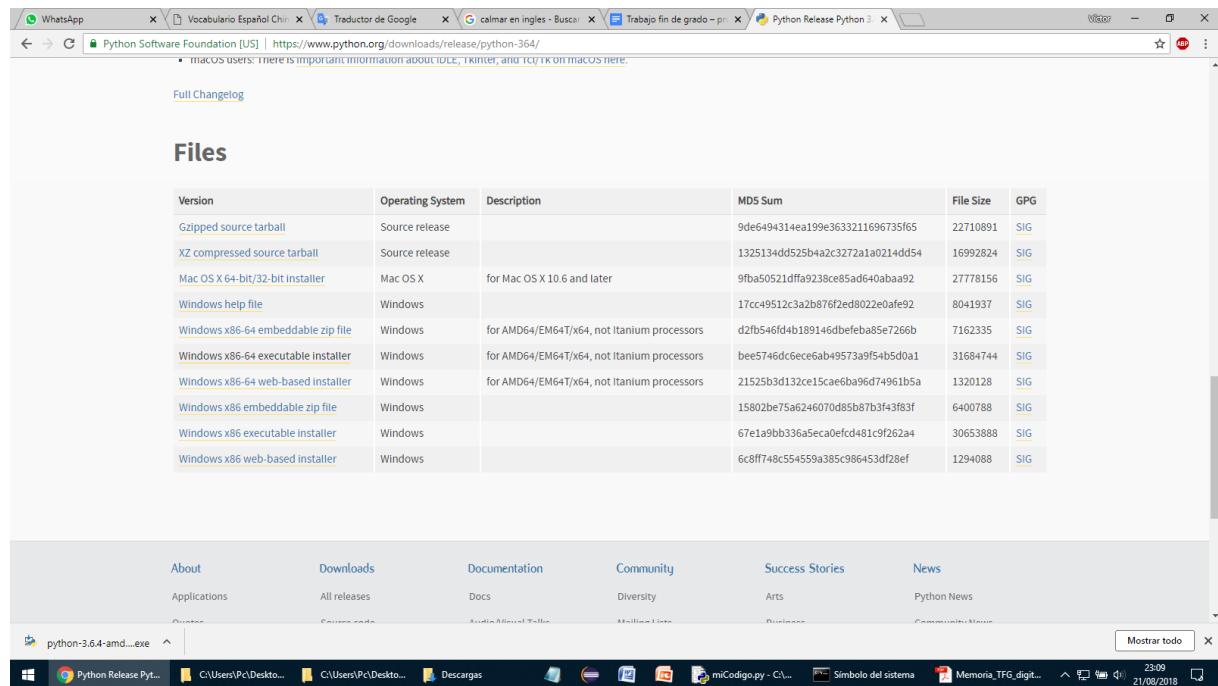


Figura 4.4: Descarga del archivo ejecutable de Python en el sistema.

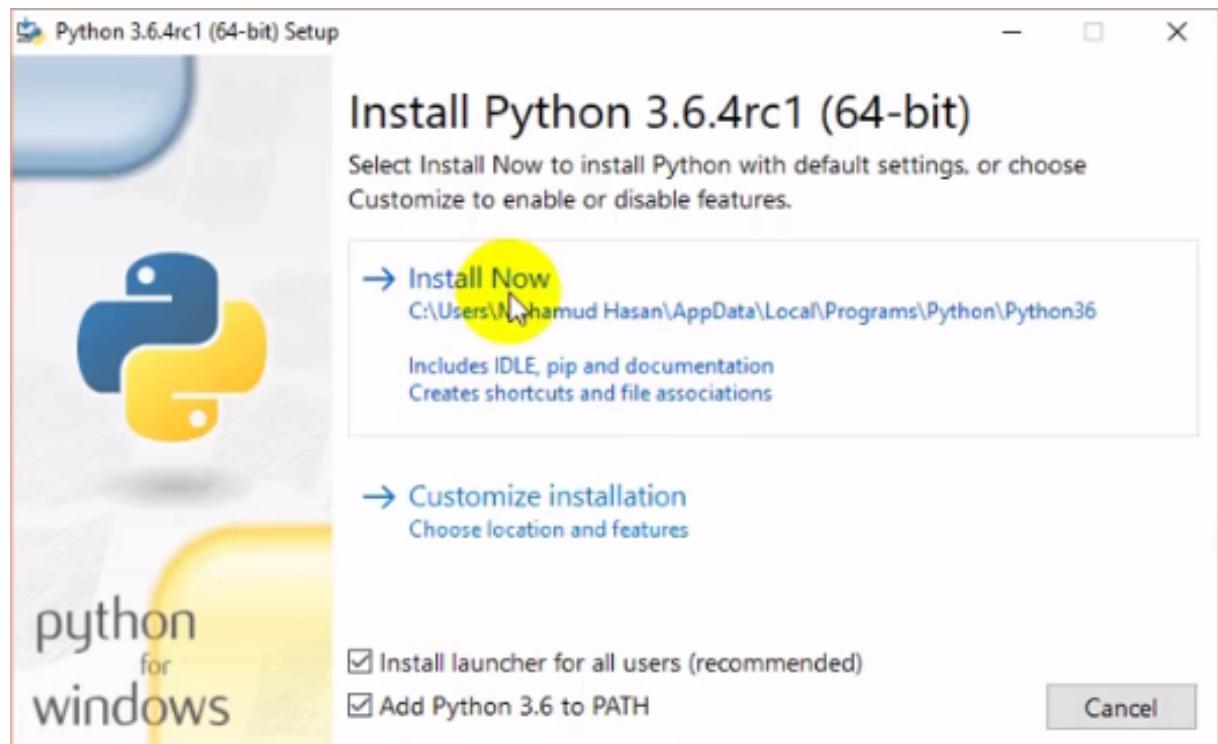


Figura 4.5: Instalador de Python.

Figura 4.5. Aparecerá esta ventana de instalación. A no ser que se desee agregar manualmente las variables de entorno de Python (con el fin de poder ejecutar Python desde la consola de comandos de Windows), hacer clic en ‘Add Python 3.6 to PATH’. El propio instalador llevará a

cabo este procedimiento. A continuación, hacer clic en ‘Install Now’.

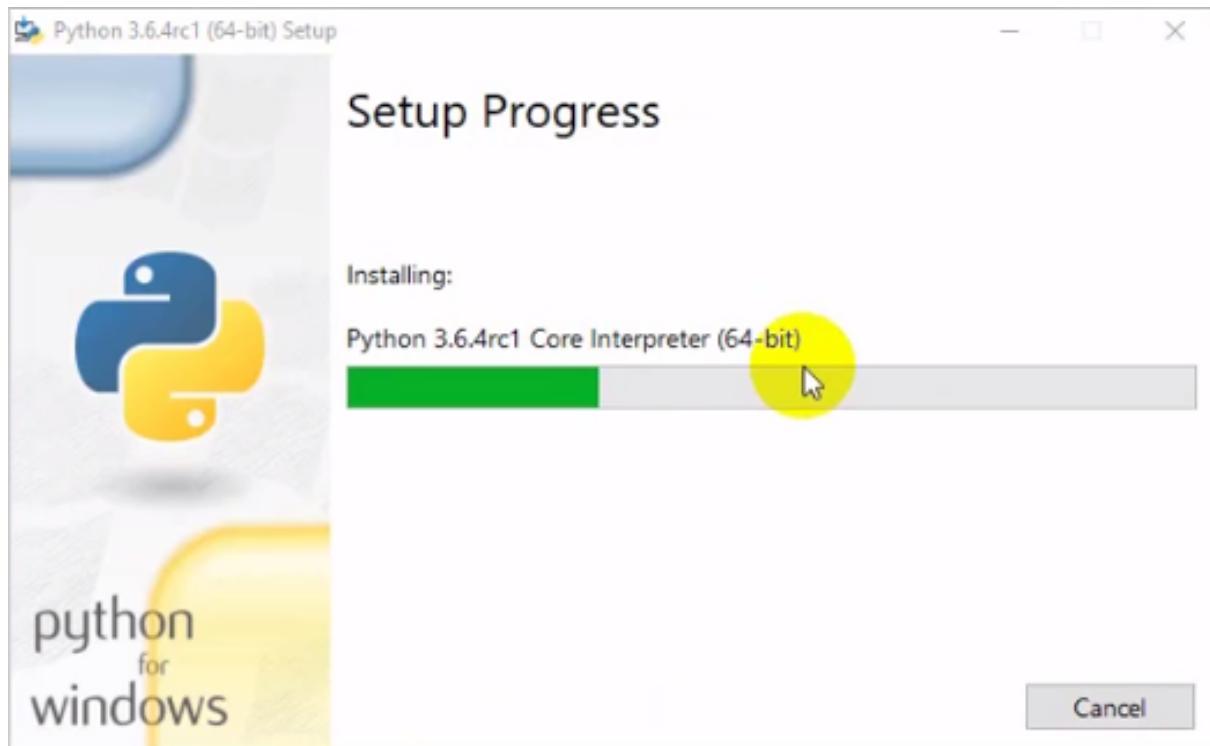


Figura 4.6: Instalación de Python en curso.

Figura 4.6. Se instalará Python en el sistema. Hay que esperar unos minutos para que este procedimiento se complete (cuando la barra verde se llene por completo).

Figura 4.7. Cuando finalice correctamente la instalación, se mostrará en la ventana del instalador ‘Setup was successful’. Hacer clic en ‘Close’ para cerrar el instalador.

Figura 4.8. El icono de Python debería aparecer en el escritorio. Hacer doble clic para abrir el programa.

Figura 4.9. Si no aparece el icono de Python en el escritorio, hacer clic en el logotipo (o bandera) de Windows mostrado en la esquina inferior izquierda de la pantalla, y teclear ‘idle’. Debería ya aparecer en el buscador de programas ‘IDLE (Python 3.6 64-bit)’. Hacer clic ahí para abrirlo.

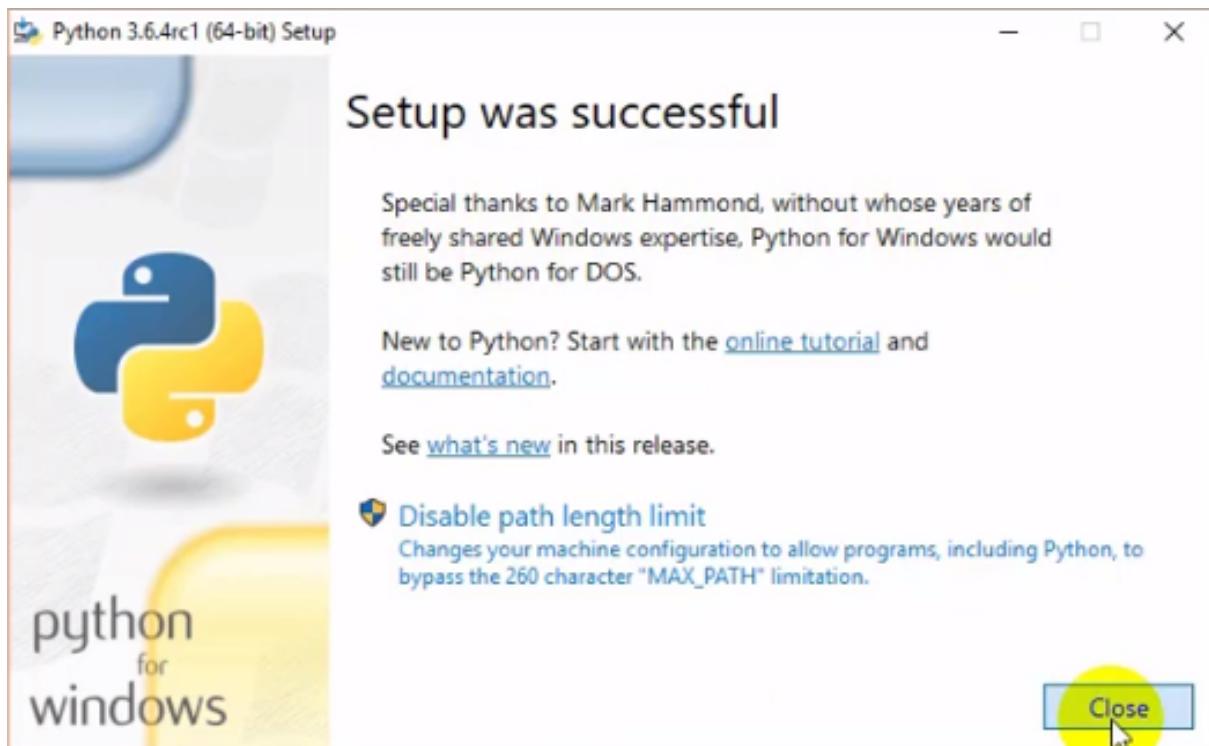


Figura 4.7: Instalación de Python exitosa.

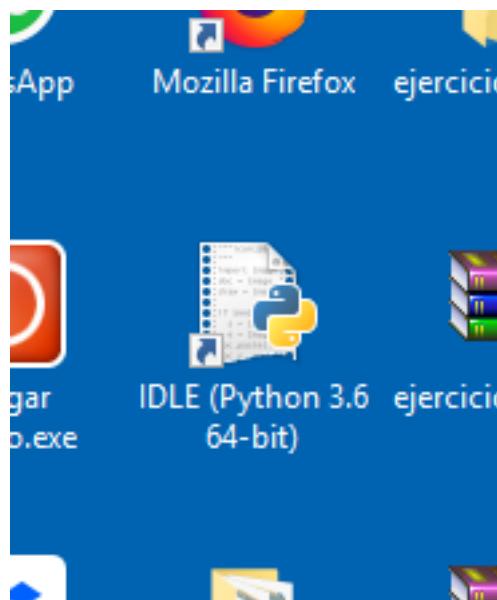


Figura 4.8: Icono en el escritorio para abrir Python.

Figura 4.10. En el caso de querer agregar el ícono del programa al escritorio (si no lo estaba), desde esa misma pantalla del buscador de programas, situar el cursor sobre el programa, hacer clic con el botón derecho del ratón, y seleccionar ‘Abrir ubicación de archivo’. Se mostrará la ubicación exacta del archivo en el sistema, abriéndose la ventana del explorador de archivos de Windows.

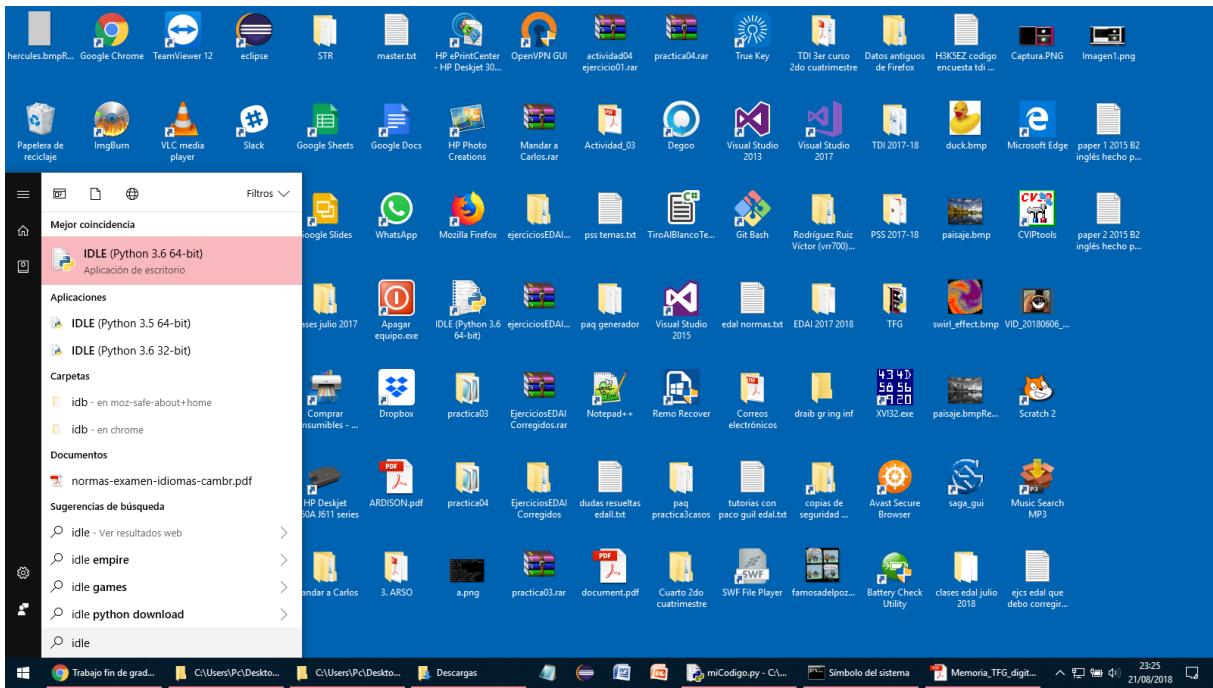


Figura 4.9: Menú Inicio de Windows, buscando Python desde él.

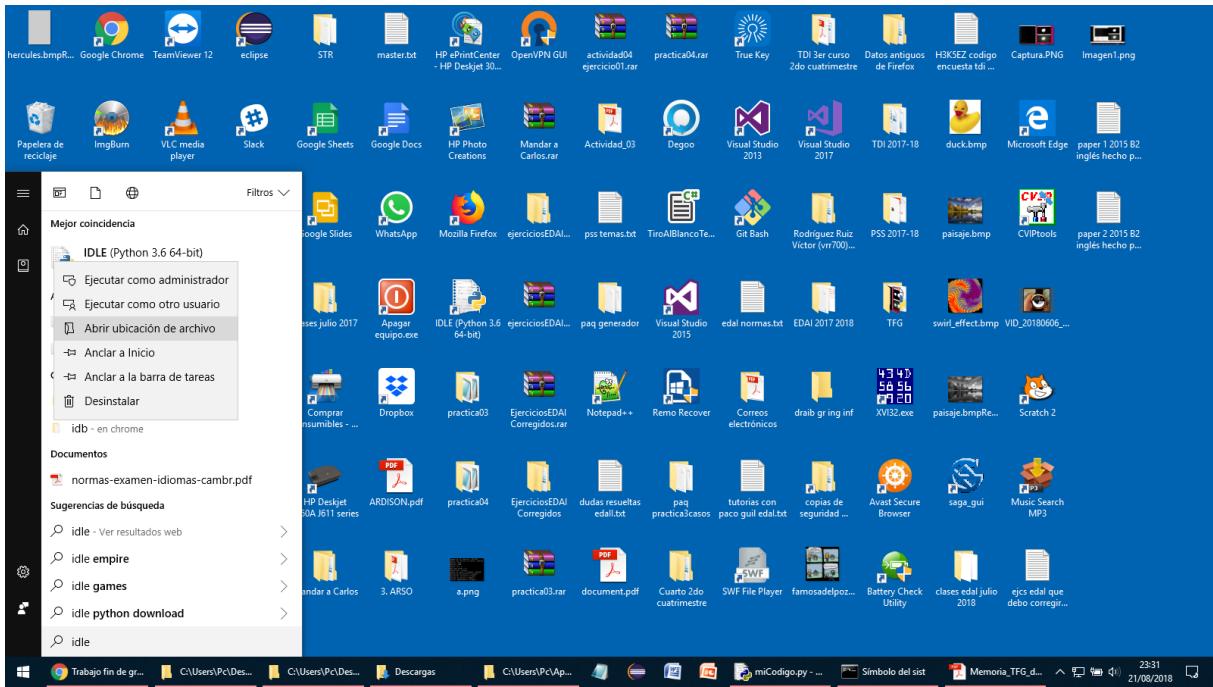


Figura 4.10: Abrir la ubicación del archivo Python en el sistema.

Figura 4.11. Desde esa ventana, y haciendo clic con el botón derecho del ratón sobre el programa (acceso directo) ‘IDLE (Python 3.6 64-bit)’, seleccionar desde el menú contextual ‘Crear acceso directo’. Se creará un nuevo acceso directo a ese programa, en esta misma ventana, con nombre

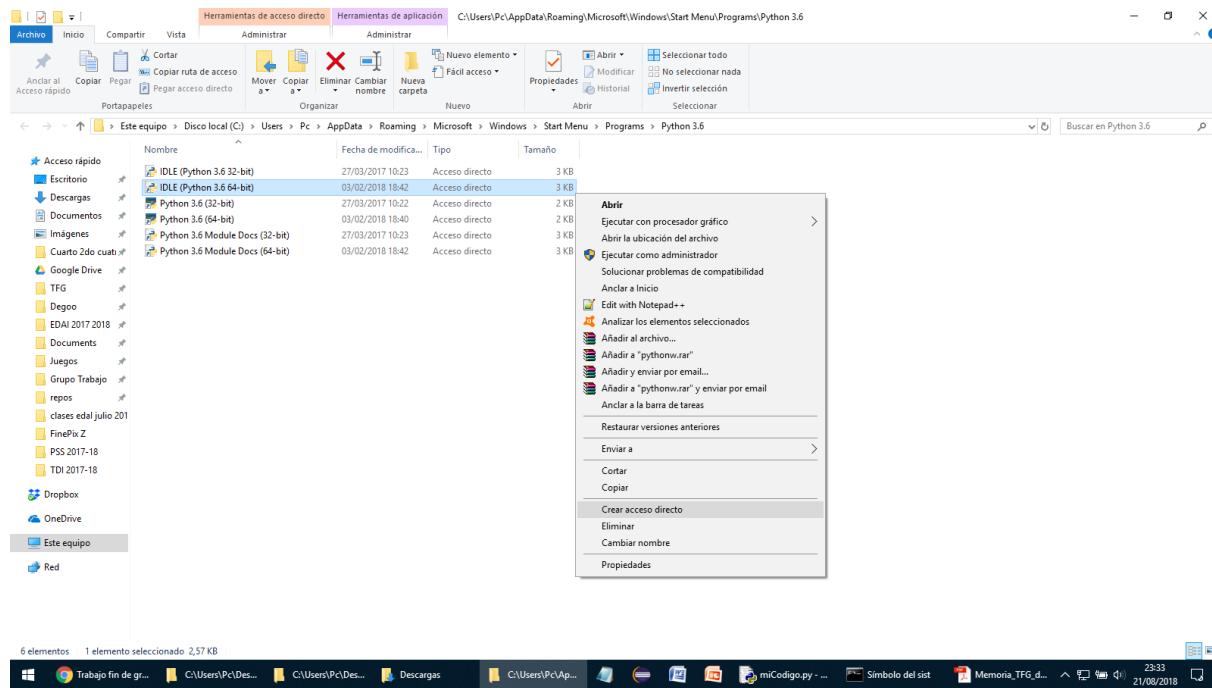


Figura 4.11: Creando un acceso directo de Python.

'IDLE (Python 3.6 64-bit) (2)'.

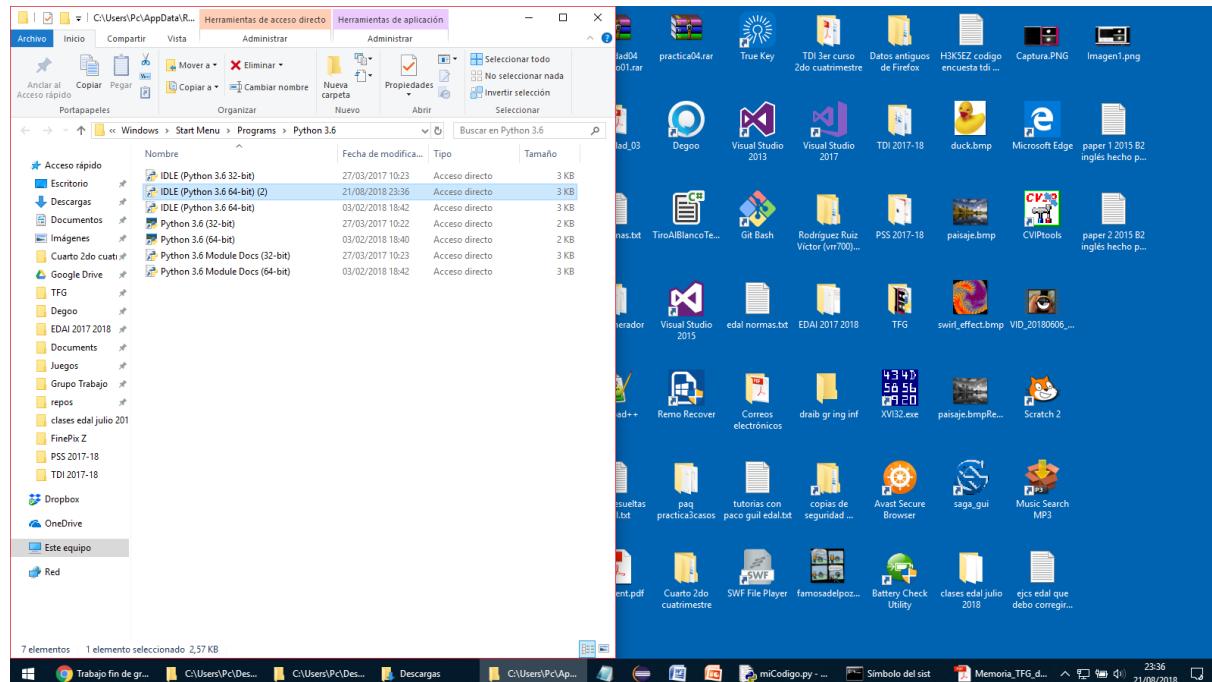


Figura 4.12: Arrastrar el acceso directo de Python creado previamente al escritorio.

Figura 4.12. Reducir el tamaño de la ventana, de forma parecida a la mostrada en esta captura de pantalla. Tras esto, simplemente habría que arrastrar y soltar este acceso directo ‘IDLE (Python 3.6 64-bit) (2)’ desde esa ventana al escritorio. Así, ya se tiene el programa en el escritorio.

### 4.3. Cómo usar el software

Para usar y ejecutar el software en el PC descrito en este informe, especialmente por primera vez, se siguen las siguientes instrucciones:

# Helióstatos

— Cómo abrir y ejecutar el proyecto de caracterización de helióstatos —

Nota: el código de este proyecto se realizó mediante el programa Python e IDLE 3.6.4.

1. Acceder a la página Web donde se ubica el proyecto: <https://github.com/VictorRodri/Heliostatos>.
2. En ella, hacer clic en el botón verde ‘Clone or download’ >‘Download ZIP’.
3. El proyecto se descargará en el disco duro del equipo, generalmente en ‘Documentos’ >‘Descargas’. De lo contrario, especificar en qué ruta exacta del equipo se realizará la descarga.
4. Descomprimir el proyecto descargado previamente, usando un software como WinRAR: <https://www.winrar.es/descargas>.
5. Instalar la última versión de Python desde su página Web oficial: <https://www.python.org/downloads/>.
6. Abrir la terminal de comandos de Windows. Para ello, hacer clic en ‘Inicio’ >‘Ejecutar’. En la ventana que aparecerá, escribir en el cuadro de texto ‘cmd’ y pulsar Enter.
7. La terminal mostrará la ruta o directorio del sistema donde se ubica usted actualmente, como C:\Users\Pc>. Ir navegando hasta encontrar el directorio que contiene el proyecto descomprimido previamente. Para acceder a una carpeta, escribir ‘cd NombreCarpeta’ y pulsar Enter. Para salir del directorio actual, escribir ‘cd ..’.
8. Una vez se haya navegado al directorio que contiene el proyecto, ejecutarlo con el comando estimacion\_potencia.py Videos/varios\_heliostatos.mp4 50 50, siendo respectivamente el nombre del proyecto ‘.py’ con el software ejecutable, el directorio que contiene el vídeo de helióstatos a ser procesado, y el ancho y alto mínimos del helióstato para su detección y análisis.
9. Durante aproximadamente un minuto, se ejecutará el software que consistirá en medir la radiación de energía que proyecta cada helióstato. Concretamente, la energía se calcula como la sumatoria de los cuadrados de cada componente BGR del helióstato. Aparecerán estos resultados en tiempo de ejecución en la consola, para cada fotograma del vídeo de helióstatos.
10. Si desea cancelar la ejecución del software, pulsar ‘Ctrl+C’.

## 4.4. Diagrama de flujo

El funcionamiento del código de helióstatos mediante la representación de un diagrama de flujo es el siguiente:

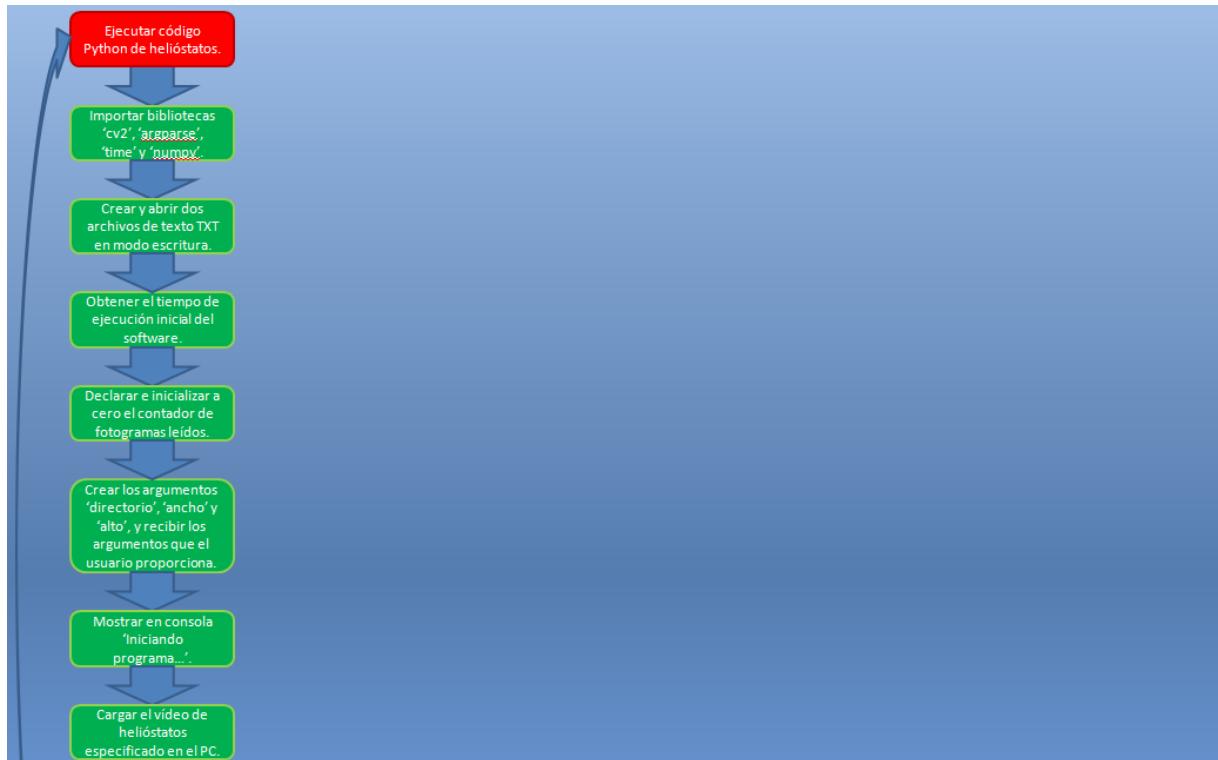


Figura 4.13: Diagrama de flujo, parte 1.

## 4.5. 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:

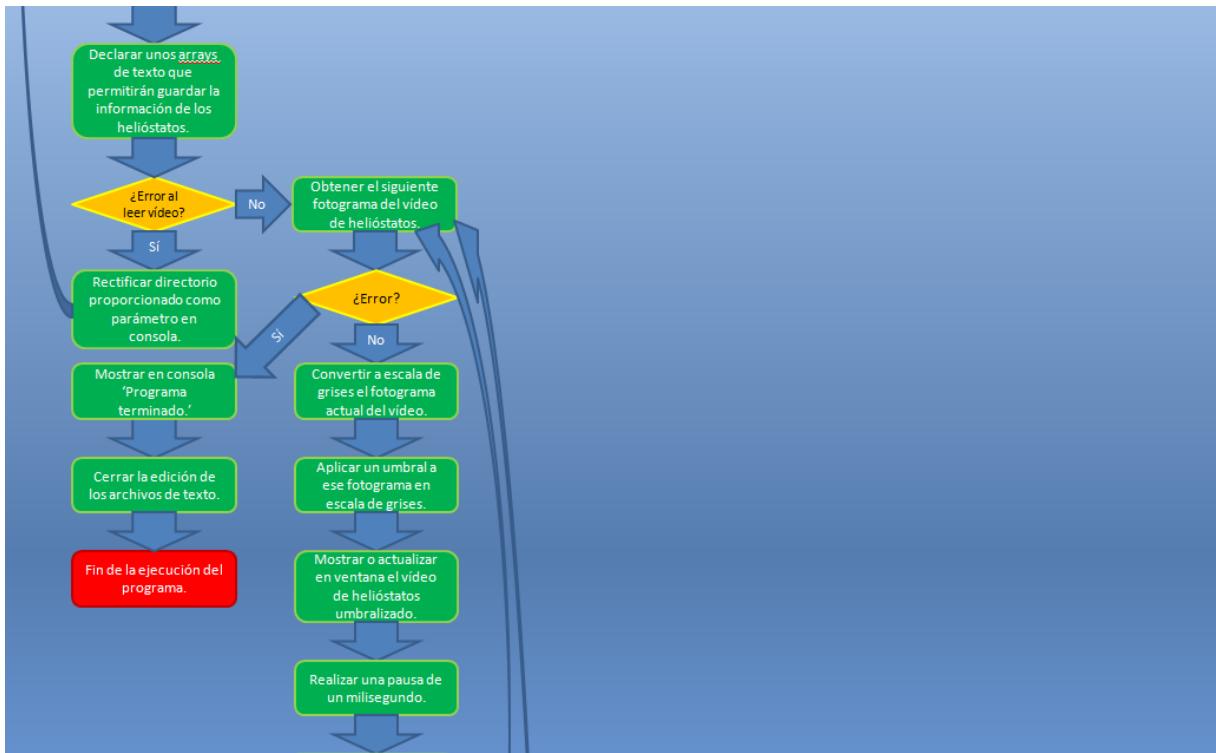


Figura 4.14: Diagrama de flujo, parte 2.

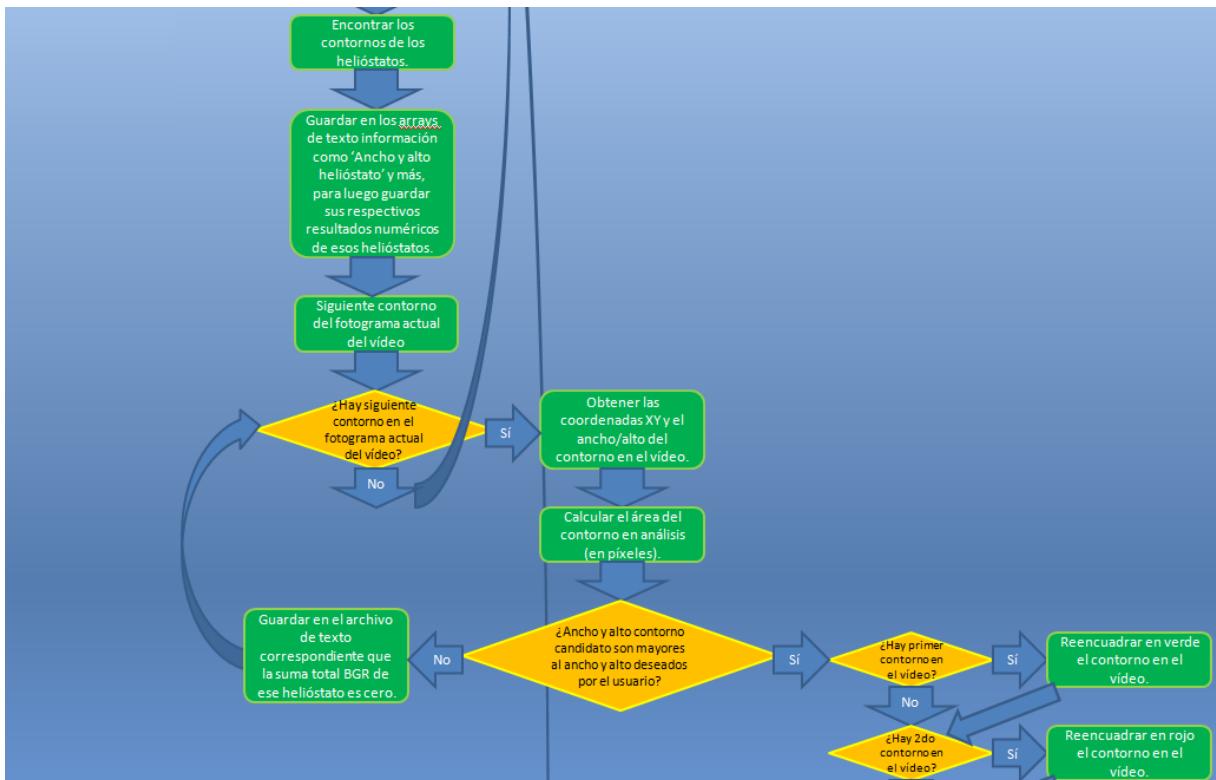


Figura 4.15: Diagrama de flujo, parte 3.

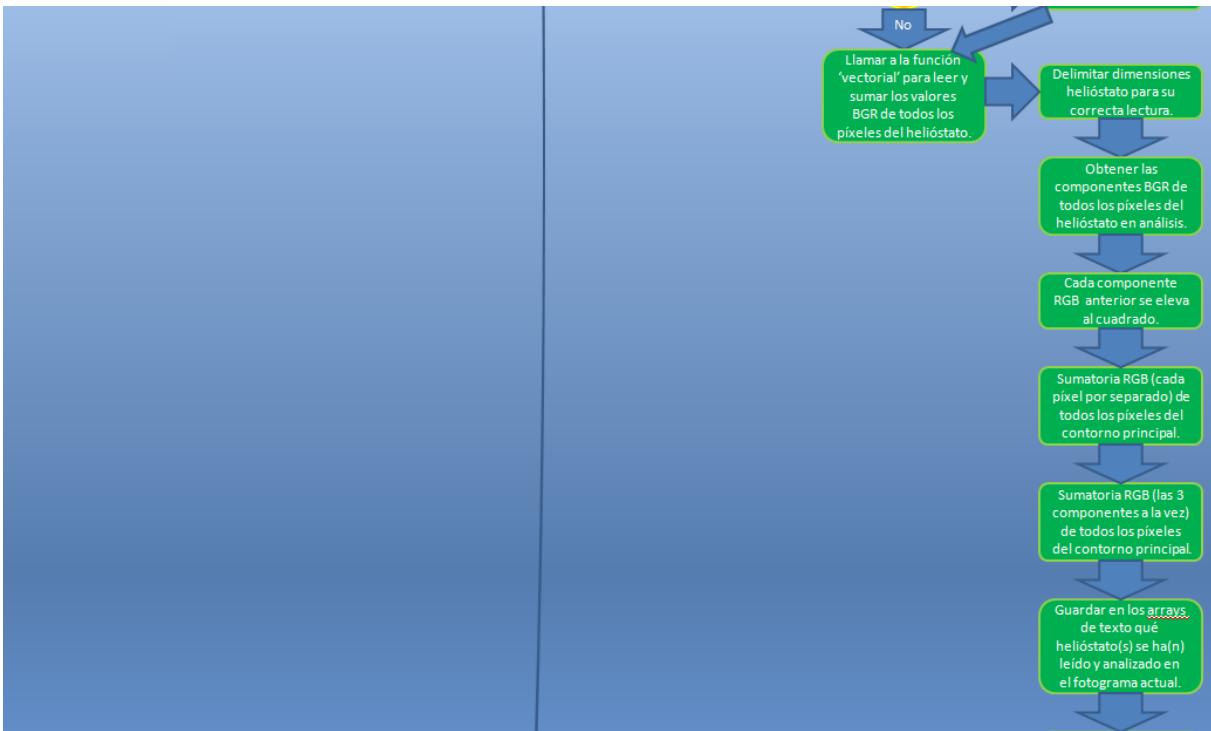


Figura 4.16: Diagrama de flujo, parte 4.

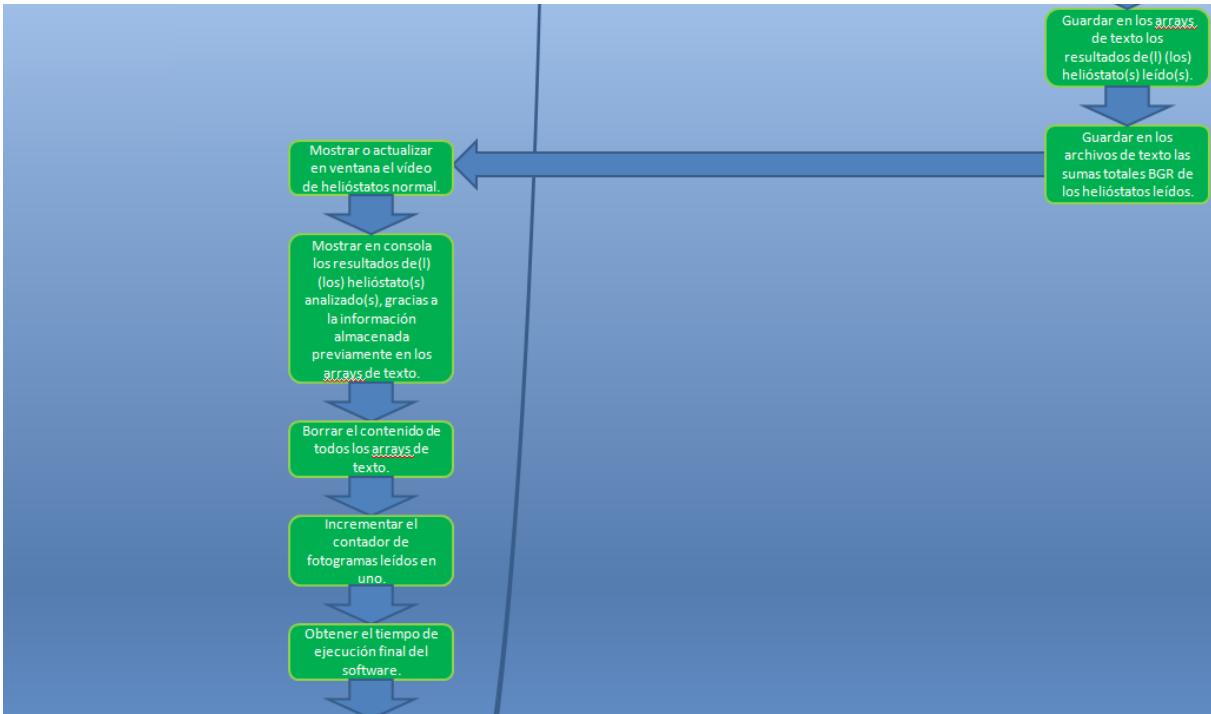


Figura 4.17: Diagrama de flujo, parte 5.

np).

cv2 permite el uso de distintas funcionalidades de Python. En este proyecto, se ha usado para



Figura 4.18: Diagrama de flujo, parte 6.

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’.

```

# Crear y abrir los siguientes archivos de texto en modo
# escritura .
f1 = open("SumasBGRHeliostatosVerdes.txt" , "w")
f2 = open("SumasBGRHeliostatosRojos.txt" , "w")

```

En el directorio del sistema donde se está ejecutando este software, se crearán los ficheros de texto ”SumasBGRHeliostatosVerdes.txt” y ”SumasBGRHeliostatosRojos.txt” y se escribirán informaciones en ellos (significado del parámetro `w`). Si ya existían estos ficheros en el sistema, serán sobreescritos por los nuevos.

```

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('anchoMinimoHeliostato', 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  
# heliosztatos a partir del cual podría estar detectándose un  
# heliosztato.  
parser.add_argument('numeroHeliostatosAnalizar', type=int)  
# Crear el argumento 6: número máximo de heliosztatos a detectar y  
# analizar en cada fotograma del vídeo de heliosztatos.  
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 heliosztatos, y el número máximo de heliosztatos que el programa deberá detectar y analizar, para cada fotograma del vídeo de heliosztatos. 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 fotogramas 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 (o 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 anterior 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 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 heliostatos .
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 recorrer 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 = int(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 (como número entero, usando `int`), 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 pueden 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 video.
m = frame[y+2:y+h-1, x+2:x+w-1]

# Obtener en matrices las componentes BGR de
# todos los píxeles del helióstato.
mB = m[:, :, 2]
mG = m[:, :, 1]
mR = m[:, :, 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

# Introducir en el array 'heliostato' qué
# helióstato(s) se ha(n) leído y analizado en el
# fotograma actual del vídeo (reencuadre verde o
# rojo).
if (i == 0):
    heliostato.append("Verde")
else:
    heliostato.append("Rojo")

# 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("-----")

# Dependiendo de qué helióstato se esté
# analizando (reencuadre verde o rojo), los
# resultados de la estimación de potencia para
# cada fotograma del vídeo se guardarán en un
# fichero o en otro.
if (i == 0):
    f1.write(str(sumaBGR)+"\n")
else:
    f2.write(str(sumaBGR)+"\n")

# 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:

$$E = \text{sumatoria R al cuadrado} + \text{sumatoria G al cuadrado} + \text{sumatoria B al cuadrado} \quad (4.1)$$

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:

$$E_{\text{total}} = Er + Eg + Eb \quad (4.2)$$

Es decir, obtener la sumatoria total (4.2) de los valores de las tres componentes RGB entre sí de todos los píxeles al cuadrado del contorno entero.

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.

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.

Donde se realiza un `heliostato.append`, se guardará en el array de texto si el helióstato analizado tiene un recuadro verde, o rojo, o ambos. De esta forma, cuando se muestren por consola los resultados de los helióstatos analizados en el fotograma actual, se sabrá a qué información le corresponde cada helióstato.

En el final de este fragmento de código, donde hay un `if-else` y unos métodos `write`, se irán guardando en los dos ficheros creados al inicio de la ejecución de este programa todos los resultados de las estimaciones de potencia (sumatoria BGR final) de cada helióstato analizado, para cada fotograma del vídeo de helióstatos. Dependiendo de si el helióstato tiene un reencuadre verde o rojo, la información será guardada en un fichero o en otro.

Respecto a las unidades de medición de la estimación de potencia (sumatoria total BGR) del helióstato, con el algoritmo lo que se intenta estimar es una potencia, que se mide en W (vatio) en el Sistema Internacional de Medidas. Pero como el algoritmo aplicado no está preparado para retornar una medida calibrada, se podría decir que el algoritmo retorna como resultado de su cálculo una variable no normalizada (al no estar calibrado) que se intenta aproximar a la potencia total reflejada por el helióstato. Hay que tener en cuenta que no están aún cuantificados los errores, lo que se deja para futuros trabajos que incluirían una campaña experimental en una planta termosolar real. Diciendo ésto, queda justificado que no se pongan unidades a la variable estimada, aunque sí se puede decir que está correlacionada con la potencia reflejada por el helióstato. La correlación, que aún no se conoce, es la que se podrá obtener en un futuro con los datos de la campaña experimental. Una corrección podría ser: "Sistema Internacional de unidades".

```

else :
# Si el ancho y alto proporcionados por el usuario
# en consola no son mayores al ancho y alto del
# helióstato actual (en análisis), aparte de no
# analizarlo, marcar su estimación de potencia como
# cero y guardarlo en un fichero o en otro,
# dependiendo de si se trata del helióstato con
# reencuadre verde o rojo.
if (i == 0):
    f1 . write ( str (0)+ "\n" )
else :
    f2 . write ( str (0)+ "\n" )

```

Si el ancho y alto proporcionados por el usuario en consola no son mayores al ancho y alto del helióstato actual (en análisis), aparte de no analizarlo (al menos para este fotograma del vídeo de helióstatos), marcar su estimación de potencia como cero y guardarla en un fichero o en otro (los dos ficheros que se crearon al comienzo de la ejecución de este programa), dependiendo de si se trata del helióstato con reencuadre verde o rojo. Es importante esto, porque si se desea generar una gráfica con estos ficheros (usando 'GnuPlot' por ejemplo), la gráfica también mostrará valores nulos (cero).

```
# 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("")
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.
print("")
print("")

```

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.

```
# Tras finalizar la edición de ambos ficheros, cerrarlos.
f1.close()
f2.close()
```

Como ya se alcanzó el final del programa, los ficheros de texto que fueron abiertos al comienzo de la ejecución de este programa y editados durante su ejecución, podrán (y deberán) ser cerrados totalmente, ya que no se editarán más.

## 4.6. Cronograma de tareas

En la figura 4.19 se muestra un cronograma de tareas con las tareas realizadas en este trabajo de fin de grado (TFG) y sus respectivas duraciones.

## 4.7. Descripción de los requerimientos/requisitos

En la elaboración de este TFG, se ha creado además un programa de ordenador en lenguaje de programación Python. Para ello, se requerían de los siguientes elementos:

Ordenador personal con conectividad Wi-Fi.

Agregar unas variables de entorno en el sistema, explicado previamente en este informe.

Tener instalados los siguientes programas en el sistema:

- ‘IDLE (Python 3.6 64-bit)’, concretamente la versión ‘3.6.4’.
- ‘Git Bash’, versión 2.7.7 para sistemas de 64 bits.
- ‘Texmaker 5.0.2.’, para realizar este informe o memoria.

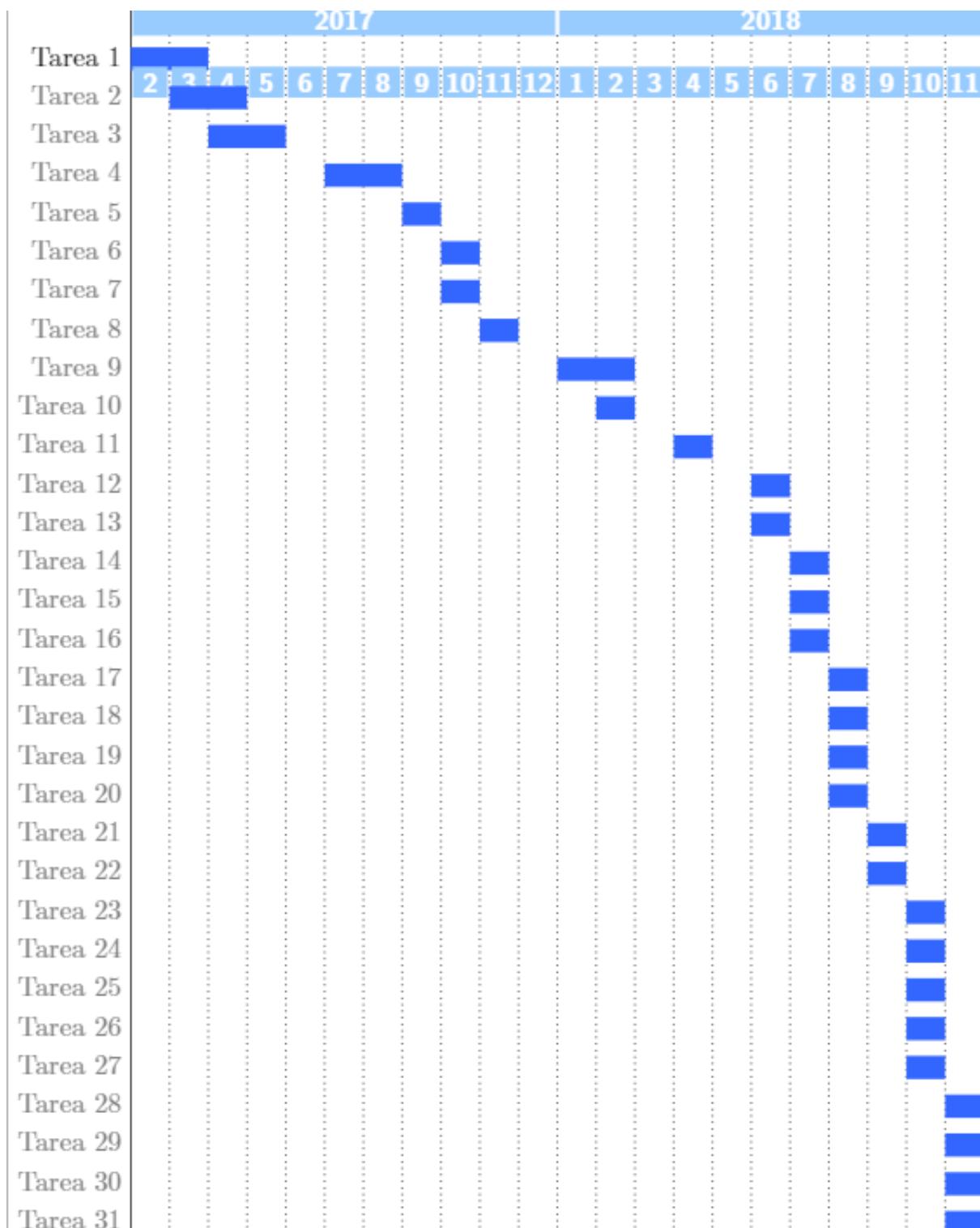


Figura 4.19: Cronograma de tareas.

## 4.8. Entradas y salidas

El software requiere de un vídeo cualquiera en formato MP4, de cualquier resolución, y guardado en una ubicación o ruta cualquiera en el disco duro del PC. El vídeo preferentemente será de helióstatos, con el fin de sacar el máximo partido al código ya que este está adaptado para procesar helióstatos.

En el código de ese software, se han utilizado las siguientes variables de entrada:

- **start\_time**. Indica el tiempo de ejecución nada más iniciar la ejecución del programa.
- **frame\_counter**. Cuenta los fotogramas analizados del vídeo.
- **end\_time**. Indica el tiempo de ejecución tras haber procesado un fotograma del vídeo (o tiempo final), y lo mismo para los siguientes, en cuyo caso el tiempo final nunca se reinicia a cero, sino que es acumulado.
- **fps**. Con la fórmula `frame_counter / float(end_time - start_time)`, es decir, usando las variables anteriores, se calcula, para cada fotograma analizado del vídeo, sus fotogramas por segundo.
- **parser**. Permite definir unos parámetros para que cuando se ejecute el programa por consola, se proporcione como parámetros o entradas (y en el siguiente orden) la ruta o directorio del vídeo a leer en el PC, el ancho y el alto mínimos del helióstato para su detección y análisis.
- **args**. Devuelve o carga el parámetro deseado y que fueron especificados por el usuario (explicado previamente): directorio, ancho o alto.
- **camara**. Contenido del vídeo de helióstatos que se irá leyendo, fotograma a fotograma, por el software.
- **grabbed**. Indica si se ha logrado cargar el siguiente fotograma del vídeo de helióstatos, queriendo además decir que dicho vídeo aún no se han analizado todos sus fotogramas y que debe proseguir la ejecución del programa.
- **frame**. Es el contenido del fotograma concreto del vídeo de helióstatos.
- **img**. Es el vídeo de helióstatos convertido a escala de grises.
- **thresh**. Aplicar un umbral para cada fotograma del vídeo de helióstatos, con el fin de determinar en cada píxel, si este es relativamente brillante o no lo es.
- **i**. Para el fotograma del vídeo actual, indica el número de helióstato en análisis: 0 (helióstato 1) ó 1 (helióstato 2).
- **contours**. Todos los contornos o helióstatos detectados, para cada fotograma del vídeo de helióstatos.
- **x, y, w, h**. Referido al contorno en análisis: **xy** son las coordenadas de un punto, **w** es el ancho, y **h** es la altura.
- **area**. Valor del área del contorno o helióstato en análisis.
- **m**. Vector BGR en tres dimensiones que representa dicha información del helióstato en análisis.
- **mB, mG, mR**. Lo mismo que la variable anterior **m**, pero esta vez separadas en sus distintas componentes BGR.

- **mB2, mG2, mR2.** Resultados de elevar al cuadrado los valores de las componentes BGR del helióstato.
- **sumB, sumG y sumR.** Indica la sumatoria acumulativa de todos los píxeles de valor R al cuadrado del helióstato en análisis, y respectivamente para G y B.
- **sumaBGR.** Indica la sumatoria de todos los píxeles de valores R al cuadrado más G al cuadrado más B al cuadrado del helióstato en análisis.
- **heliostato, anchoAlto, areaTotal, sumaBGparcial, sumaBGRtotal.** Variables en forma de arrays que almacenan (y después muestran en consola) los textos, datos y resultados de los helióstatos analizados. Cada array almacena su correspondiente información.

Las siguientes variables de entrada son completamente personalizables por el usuario, ya que estas son insertables desde la ventana de comandos de Windows al ejecutar el programa:

- **args.directorioVideoHeliostatosCargar.** Parámetro que el programa solicita al usuario para que lo introduzca por consola y que permite indicar el directorio o ruta donde se ubica el vídeo de helióstatos a procesar.
- **args.umbralVideoHeliostatos.** Parámetro que el programa solicita al usuario para que lo introduzca por consola y que permite definir el umbral del vídeo de helióstatos para que, si los píxeles (del helióstato) en análisis de ese vídeo superan ese umbral, sean analizados, y viceversa.
- **args.numeroHeliostatosAnalizar.** Parámetro que el programa solicita al usuario para que lo introduzca por consola y que permite definir el número máximo de helióstatos que serán analizados (si cumplen los requisitos) en cada fotograma del vídeo de helióstatos, ignorando así posibles falsos contornos/helióstatos que suelen ser más pequeños que los helióstatos reales.
- **args.anchoMinimoHeliostato, args.altoMinimoHeliostato.** Parámetros que el programa solicita al usuario para que los introduzca por consola y que permiten definir el ancho/alto mínimos que debe tener el helióstato para ser analizado.

Las siguientes variables no son usadas para este proyecto y se explicarán muy brevemente. No obstante, era necesario declararlas para evitar errores de compilación en el software.

- **ret.** Umbralización de Otsu en el vídeo de helióstatos.
- **im2.** Imagen fuente, procedente del fotograma actual del vídeo de helióstatos.
- **hierarchy.** Método de aproximación del contorno.

Tras el uso de las variables de entrada anteriores, el software mostrará por pantalla una consola de información y resultados y dos ventanas del vídeo de helióstatos. Ambos elementos se van actualizando en tiempo de ejecución. Los resultados mostrados en consola de los helióstatos son finales, no parciales. En la consola aparece la siguiente información, para cada fotograma del vídeo de helióstatos:

Analizando el helióstato verde y/o rojo.

Ancho y alto **del** helióstato en píxeles.

Área **del** helióstato en píxeles.

Sumatorias acumulativas de los valores BGR al cuadrado (cada componente por separado) de

Suma total (o unificación) de esas tres componentes BGR entre sí.

Fotogramas por segundo **del** vídeo.

Ejemplos gráficos (figuras 4.20 y 4.21):

```
[['Verde']]
[['Ancho y alto WH del helióstato en píxeles: ', 178, 152, ''],
 ['Área del helióstato en píxeles: ', 20838, ''],
 ['Sumatorias BGR al cuadrado de todos sus píxeles: ', 2211695, 1607043, 2479993, ''],
 ['Suma total BGR al cuadrado helióstato completo: ', 6298731, ''],
 
FPS: 10.091275469218912

[['Rojo']]
[['Ancho y alto WH del helióstato en píxeles: ', 178, 152, ''],
 ['Área del helióstato en píxeles: ', 20794, ''],
 ['Sumatorias BGR al cuadrado de todos sus píxeles: ', 2213187, 1596344, 2479648, ''],
 ['Suma total BGR al cuadrado helióstato completo: ', 6289179, ''],
 
FPS: 10.09667121914513
```

Figura 4.20: Resultados por consola al analizar el helióstato verde y rojo, aparecidos en distintos fotogramas del vídeo.

```
[['Verde', 'Rojo']]
[['Ancho y alto WH del helióstato en píxeles: ', 104, 88, ''],
 ['Área del helióstato en píxeles: ', 6811, ''],
 ['Sumatorias BGR al cuadrado de todos sus píxeles: ', 839252, 785348, 890931, ''],
 ['Suma total BGR al cuadrado helióstato completo: ', 2515531, ''],
 
FPS: 52.801887688030014
```

Figura 4.21: Resultados por consola al analizar el helióstato verde y rojo, aparecidos en el mismo fotograma del vídeo.

En las ventanas del vídeo de helióstatos, en una de ellas aparece el vídeo umbralizado, con el fin de detectar lo que son contornos (helióstatos) e ignorar los que no lo sean, así como los falsos contornos. Y en la otra ventana, aparece el vídeo original. En esta última, cuando se detecten helióstatos de un tamaño (ancho y alto) determinados (deseados por el usuario en los parámetros que él mismo proporcionó al ejecutar el programa), se reencuadrará con un cuadrado o rectángulo verde o rojo y se analizará por el software. El rectángulo en general será de color verde si solo se muestra un helióstato (o dos helióstatos rozándose entre sí, tratados como si fueran uno solo) en el fotograma actual del vídeo, pero si en dicho fotograma se muestran dos helióstatos (y separados entre sí), el rectángulo para el helióstato secundario será rojo. Esto permitirá diferenciar bien los helióstatos, y saber qué información mostrada en consola le corresponde a cada helióstato.

Ejemplo gráfico (figura 4.22):



Figura 4.22: El software detecta, analiza y reencuadra en el vídeo los helióstatos detectados, y en distintos colores para su diferenciación.

#### 4.9. Análisis de los recursos consumidos

A continuación, se mostrarán y explicarán los resultados del uso de la CPU y del consumo de la memoria RAM en el PC, en el administrador de tareas de Windows 10, durante la ejecución

del software en distintos períodos de tiempo. También se medirán los fotogramas por segundo (FPS) actuales del vídeo.

#### 4.9.1. Cargando el vídeo 'variosHeliostatos.mp4'

Se inicia la ejecución del software desde la terminal de Windows, estando en el directorio que contiene dicho software a ejecutar, y usando el comando `estimacion_potencia.py Videos/varios_heliostatos.mp4`

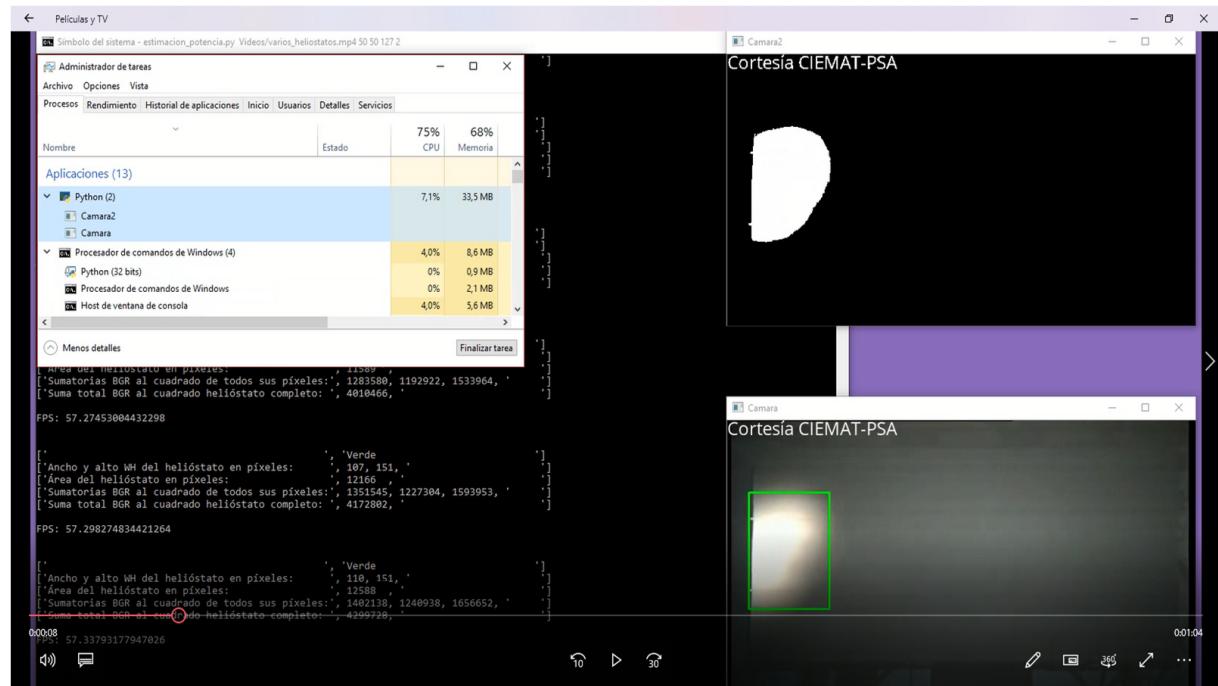


Figura 4.23: Instante de tiempo 1 de la ejecución del software para el vídeo 'varios\_heliostatos.mp4'.

Figura 4.23. Al iniciar la ejecución del software, cuando está entrando el primer helióstato desde el lado izquierdo en el vídeo y aún no se muestra en su totalidad, Python consume un 7,1 % de CPU y 33,5 MB de memoria RAM, mientras que la terminal de Windows consume un 4 % de CPU y 8,6 MB de memoria RAM. Los FPS son de 57,27.

Figura 4.24. Cuando el primer helióstato ya ha llegado al centro del vídeo, Python consume un 6,5 % de CPU y 33,8 MB de memoria RAM, mientras que la terminal de Windows consume un 7 % de CPU y 8,6 MB de memoria RAM. Los FPS son de 60,61.

Figura 4.25. Cuando está entrando desde la izquierda del vídeo el segundo helióstato, Python consume un 5,2 % de CPU y 34,5 MB de memoria RAM, mientras que la terminal de Windows consume un 6,1 % de CPU y 8,6 MB de memoria RAM. Los FPS son de 61,53.

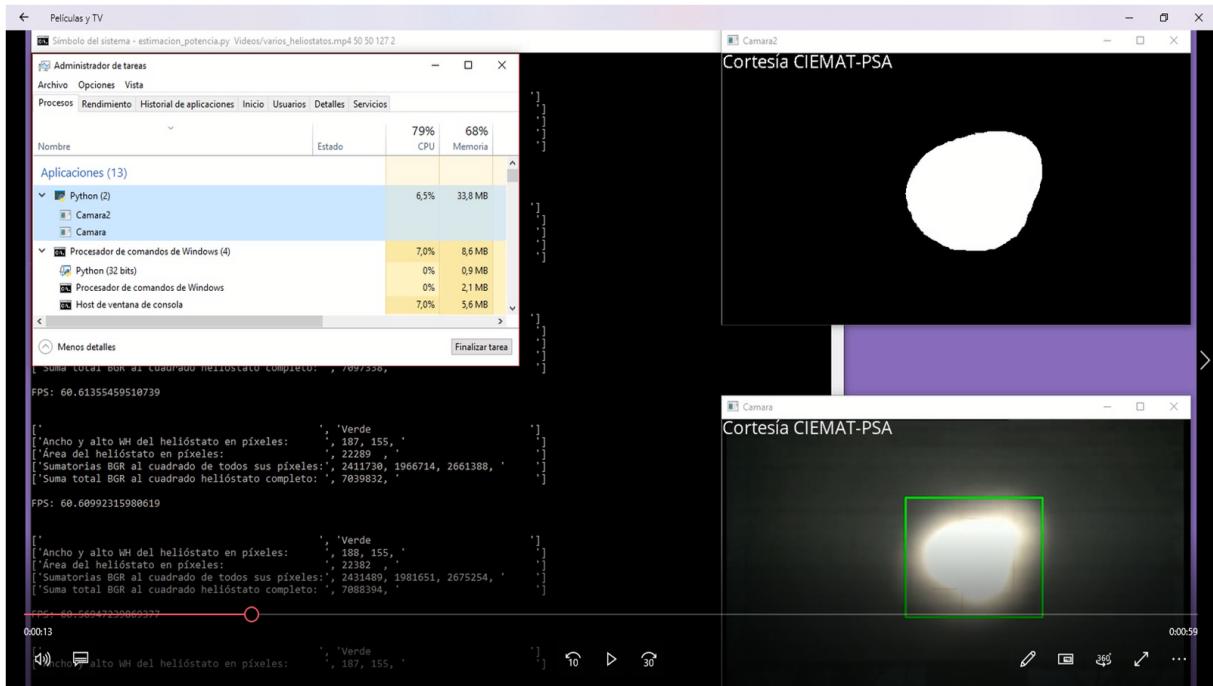


Figura 4.24: Instante de tiempo 2 de la ejecución del software para el vídeo 'varios\_heliostatos.mp4'.

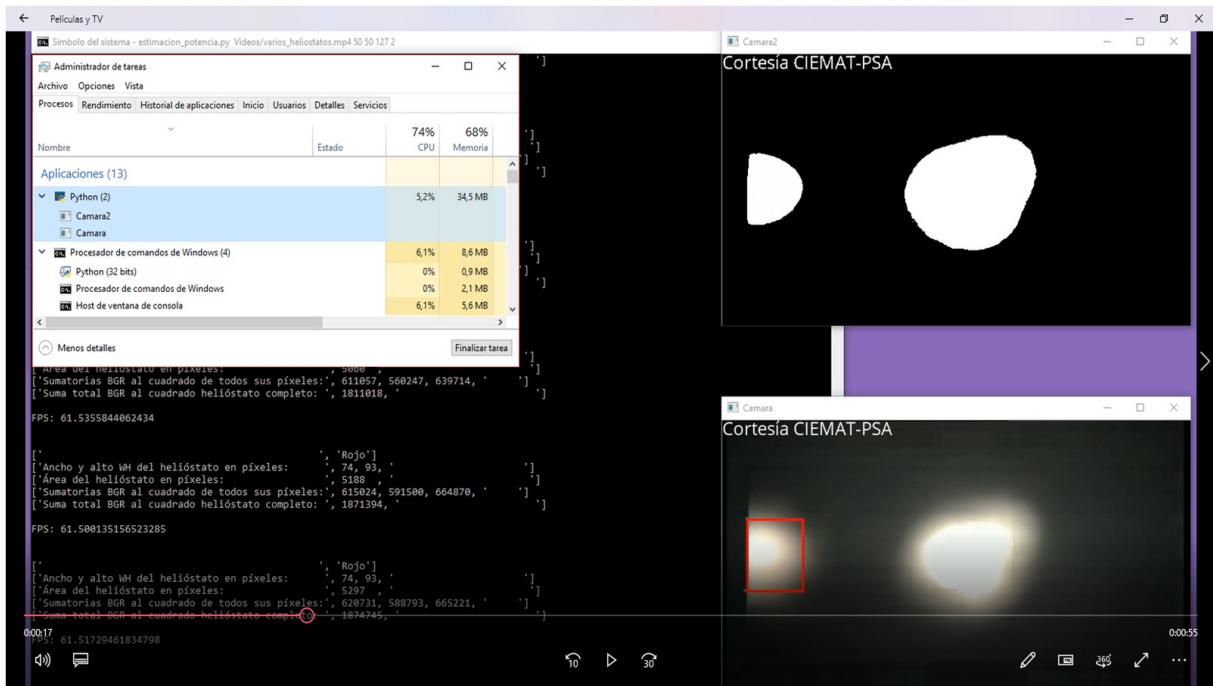


Figura 4.25: Instante de tiempo 3 de la ejecución del software para el vídeo 'varios\_heliostatos.mp4'.

Figura 4.26. Cuando el segundo helióstato prácticamente se ha fusionado con el helióstato permanecido en el centro del video, Python consume un 7,3 % de CPU y 34,5 MB de memoria

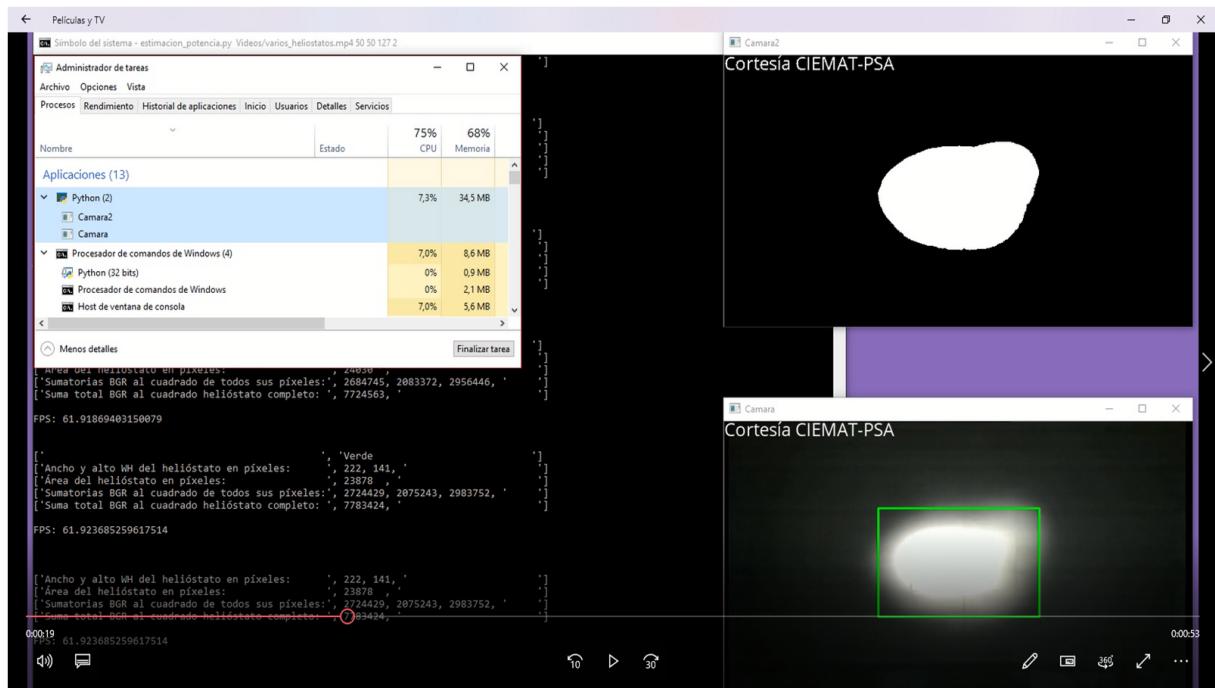


Figura 4.26: Instante de tiempo 4 de la ejecución del software para el vídeo 'varios\_heliostatos.mp4'.

RAM, mientras que la terminal de Windows consume un 7% de CPU y 8,6 MB de memoria RAM. Los FPS son de 61,91.

Figura 4.27. Cuando está entrando desde la izquierda del vídeo el tercer heliostato, Python consume un 3,7% de CPU y 34,5 MB de memoria RAM, mientras que la terminal de Windows consume un 6,9% de CPU y 8,6 MB de memoria RAM. Los FPS son de 62,38.

Figura 4.28. Cuando el tercer heliostato prácticamente se ha fusionado con el heliostato permanecido en el centro del vídeo, Python consume un 6,7% de CPU y 34,5 MB de memoria RAM, mientras que la terminal de Windows consume un 7,4% de CPU y 8,6 MB de memoria RAM. Los FPS son de 62,48.

Figura 4.29. Cuando está entrando desde la izquierda del vídeo el cuarto y último heliostato, Python consume un 6,2% de CPU y 34,5 MB de memoria RAM, mientras que la terminal de Windows consume un 6,2% de CPU y 8,6 MB de memoria RAM. Los FPS son de 62,57.

Figura 4.30. Cuando el cuarto heliostato comienza a iniciar la fusión con el heliostato perman-

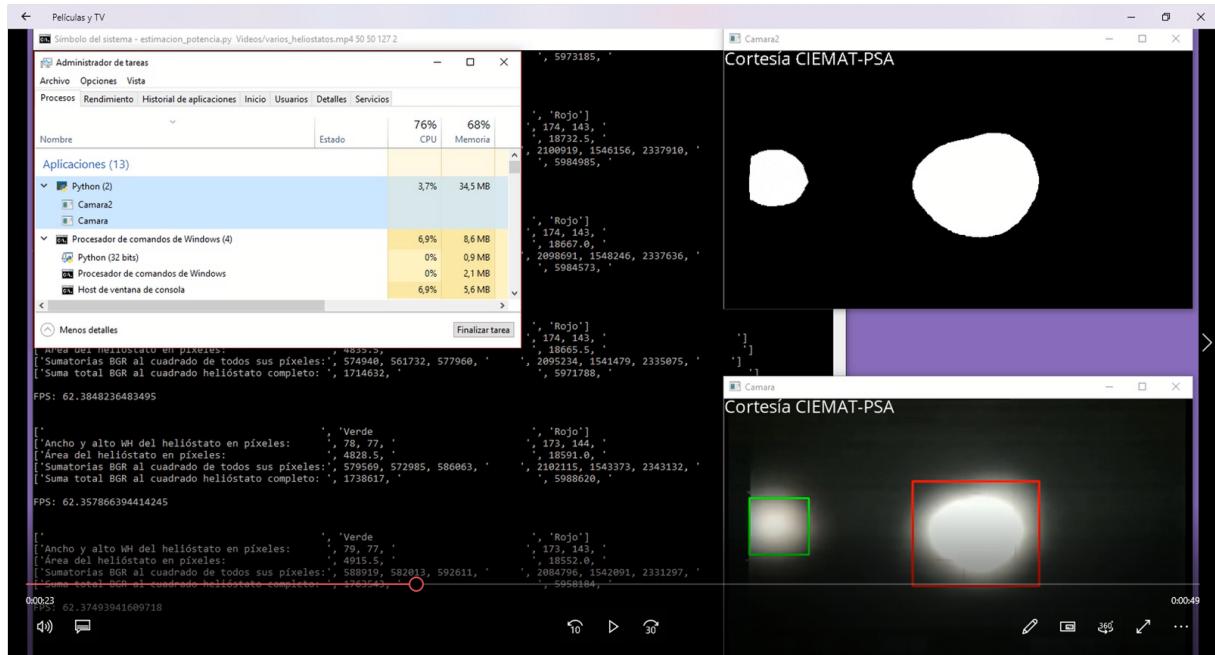


Figura 4.27: Instante de tiempo 5 de la ejecución del software para el vídeo 'varios\_heliostatos.mp4'.

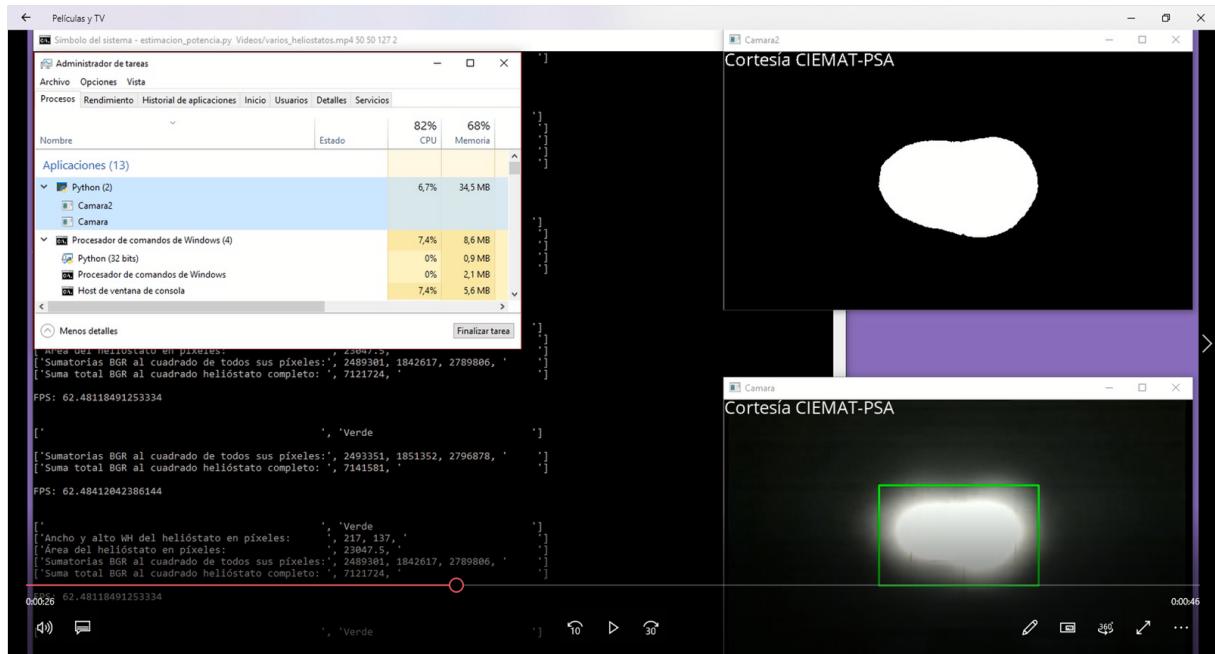


Figura 4.28: Instante de tiempo 6 de la ejecución del software para el vídeo 'varios\_heliostatos.mp4'.

necido en el centro del vídeo, Python consume un 5,1% de CPU y 34,5 MB de memoria RAM, mientras que la terminal de Windows consume un 6,9% de CPU y 8,6 MB de memoria RAM. Los FPS son de 62,49.

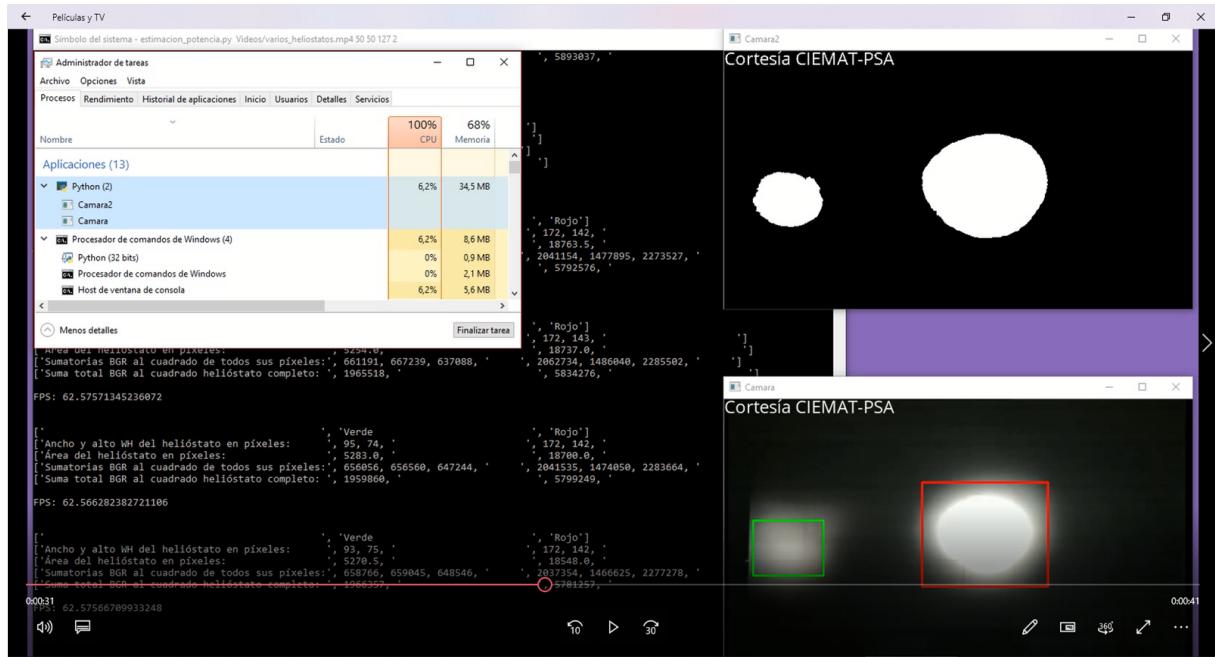


Figura 4.29: Instante de tiempo 7 de la ejecución del software para el vídeo 'varios\_heliostatos.mp4'.

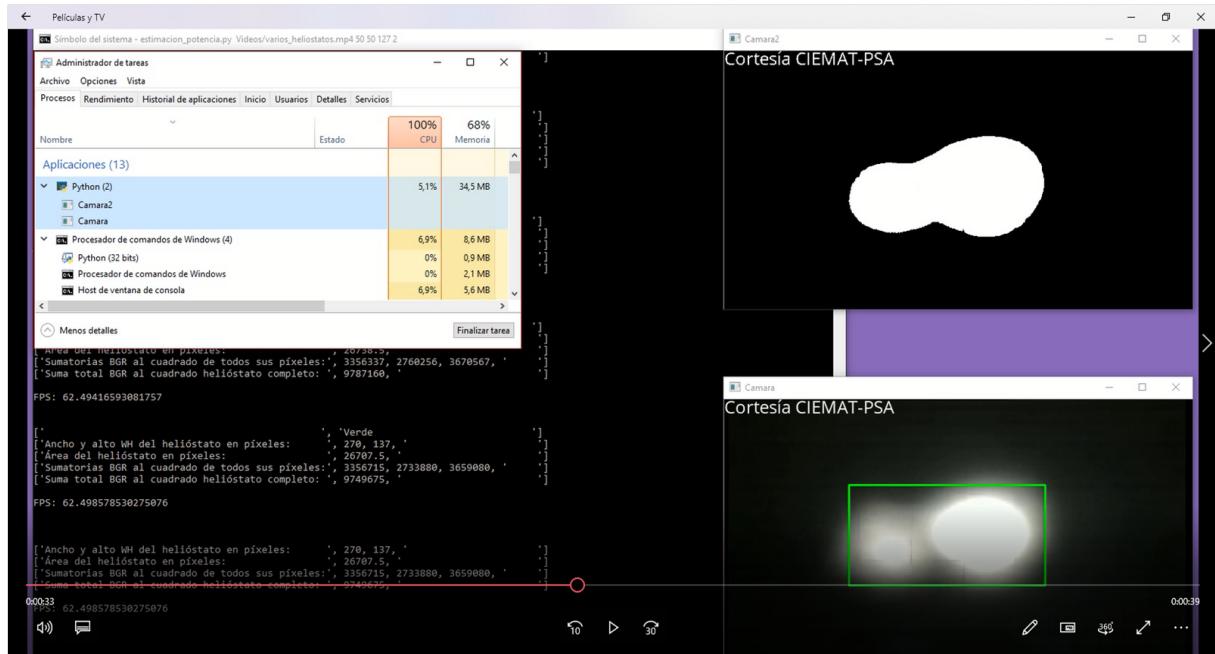


Figura 4.30: Instante de tiempo 8 de la ejecución del software para el vídeo 'varios\_heliostatos.mp4'.

Figura 4.31. Cuando uno de los heliostatos fusionados con el heliostato del centro del vídeo trata de salir del mismo (todavía no se ha completado este procedimiento). Python consume un 4,2 % de CPU y 34,5 MB de memoria RAM, mientras que la terminal de Windows consume un 6,5 %

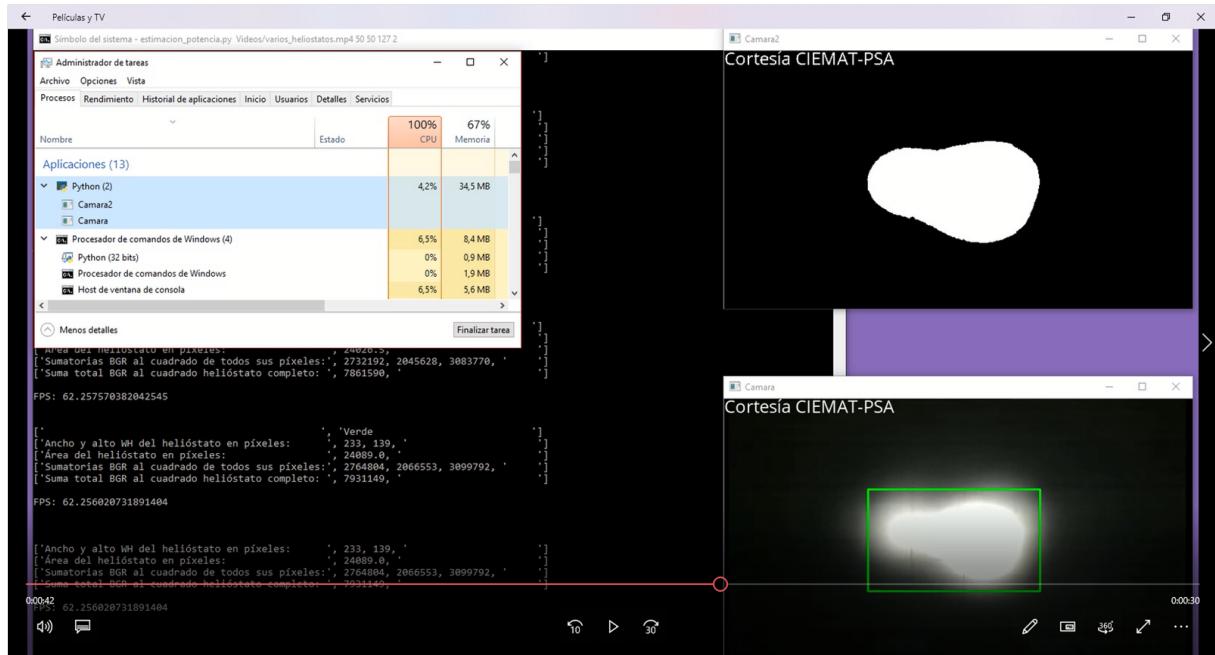


Figura 4.31: Instante de tiempo 9 de la ejecución del software para el vídeo 'varios\_heliostatos.mp4'.

de CPU y 8,4 MB de memoria RAM. Los FPS son de 62,25.

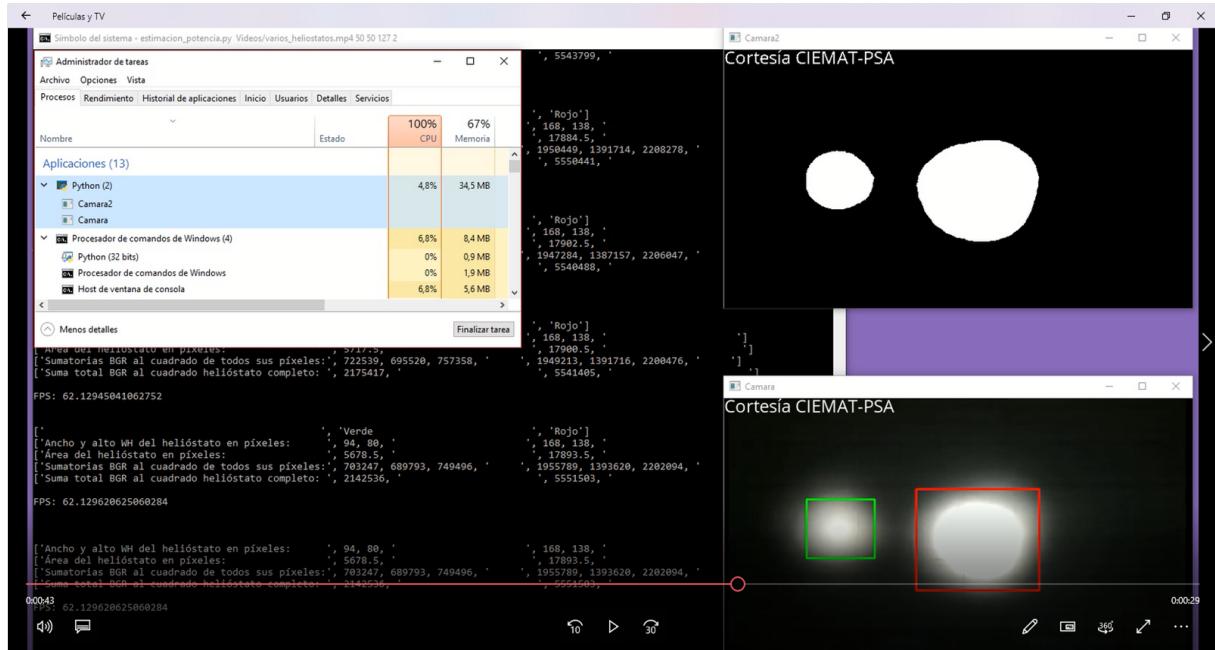


Figura 4.32: Instante de tiempo 10 de la ejecución del software para el vídeo 'varios\_heliostatos.mp4'.

Figura 4.32. Cuando el helióstato ya se ha separado del helióstato central del vídeo, Python consume un 4,8 % de CPU y 34,5 MB de memoria RAM, mientras que la terminal de Windows consume un 6,8 % de CPU y 8,4 MB de memoria RAM. Los FPS son de 62,12.

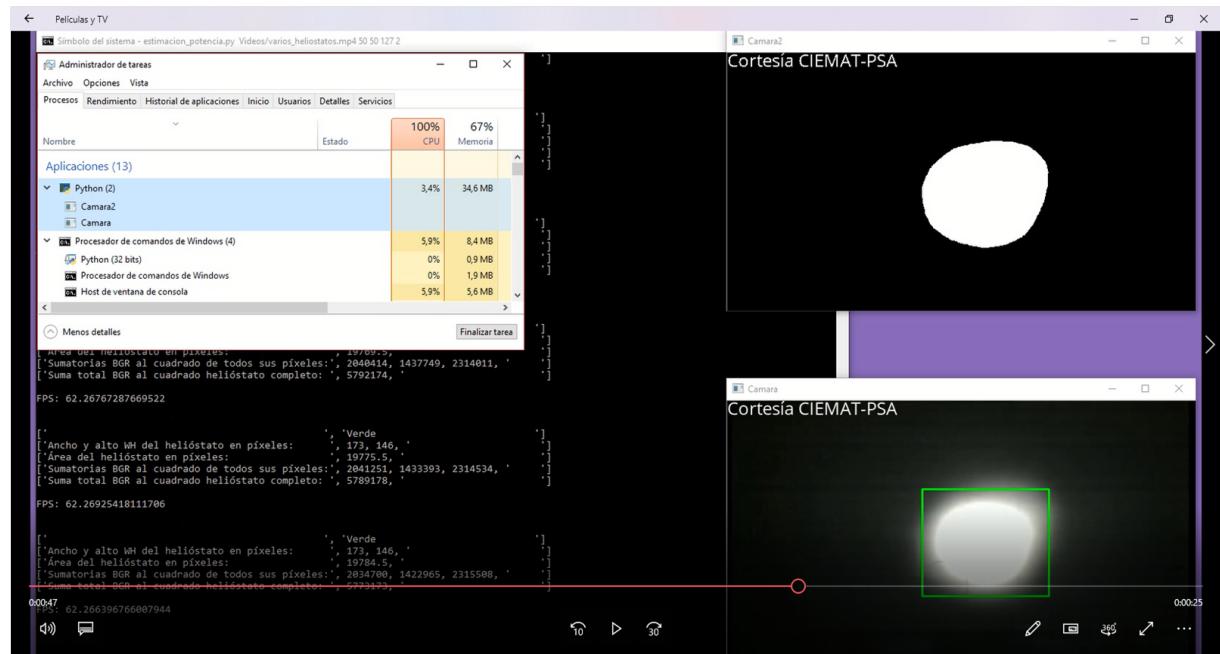


Figura 4.33: Instante de tiempo 11 de la ejecución del software para el vídeo 'varios\_heliostatos.mp4'.

Figura 4.33. Cuando únicamente permanece el helióstato central del vídeo (el otro helióstato ya se ha ido a la izquierda del vídeo), Python consume un 3,4 % de CPU y 34,6 MB de memoria RAM, mientras que la terminal de Windows consume un 5,9 % de CPU y 8,4 MB de memoria RAM. Los FPS son de 62,26.

Figura 4.34. Cuando otro de los helióstatos fusionados con el helióstato del centro del vídeo trata de salir del mismo (todavía no se ha completado este procedimiento). Python consume un 7,4 % de CPU y 34,6 MB de memoria RAM, mientras que la terminal de Windows consume un 4,6 % de CPU y 8,4 MB de memoria RAM. Los FPS son de 62,32.

Figura 4.35. Cuando los dos helióstatos están a punto de separarse entre sí. Python consume un 7,4 % de CPU y 34,6 MB de memoria RAM, mientras que la terminal de Windows consume un 4,6 % de CPU y 8,4 MB de memoria RAM. Los FPS son de 62,33.

Figura 4.36. Cuando ambos helióstatos ya se han separado entre sí, Python consume un 7,2 %

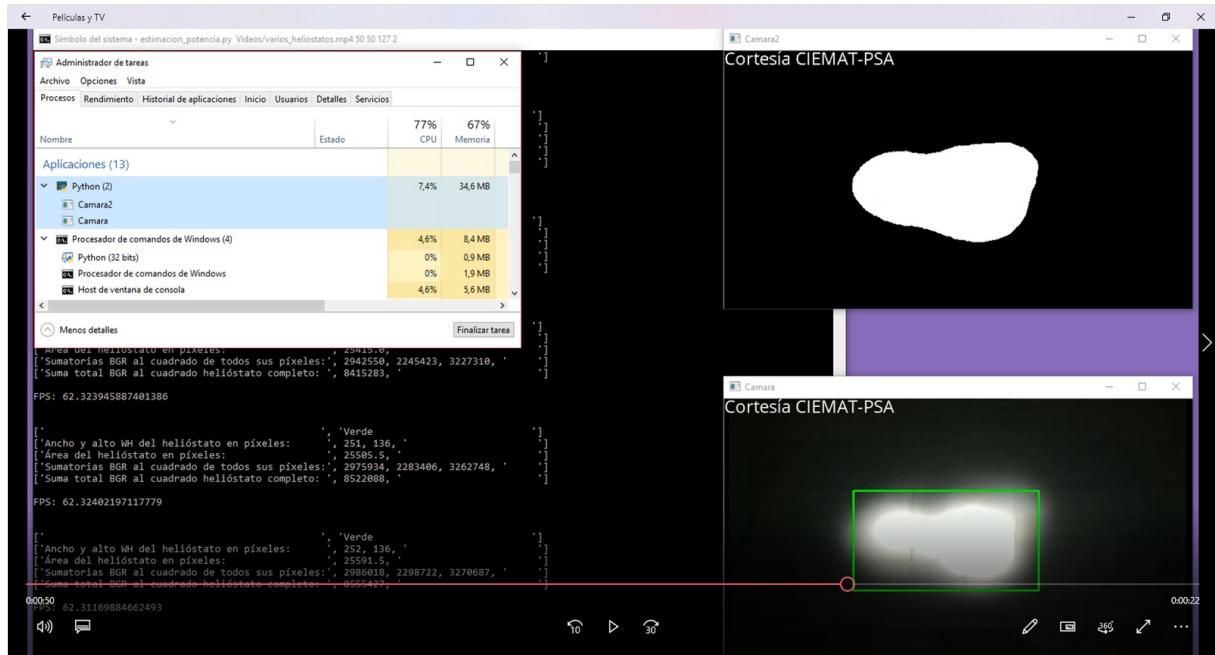


Figura 4.34: Instante de tiempo 12 de la ejecución del software para el vídeo 'varios\_heliostatos.mp4'.

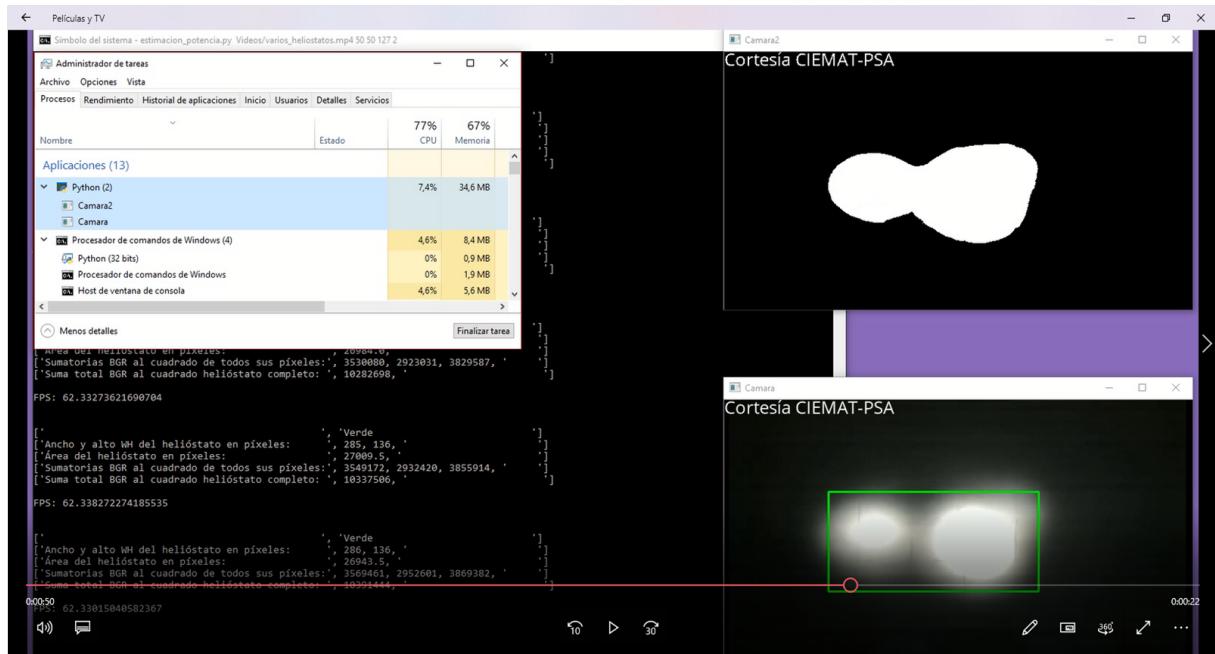


Figura 4.35: Instante de tiempo 13 de la ejecución del software para el vídeo 'varios\_heliostatos.mp4'.

de CPU y 34,6 MB de memoria RAM, mientras que la terminal de Windows consume un 4,9 % de CPU y 8,4 MB de memoria RAM. Los FPS son de 62,35.

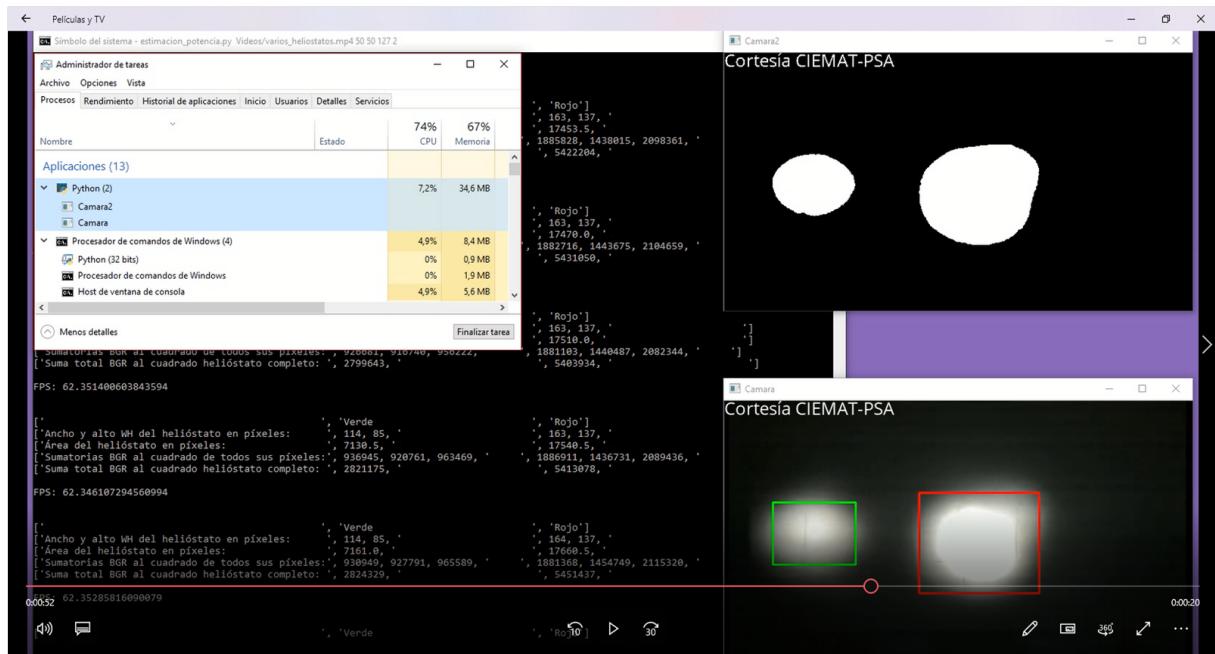


Figura 4.36: Instante de tiempo 14 de la ejecución del software para el vídeo 'varios\_heliostatos.mp4'.

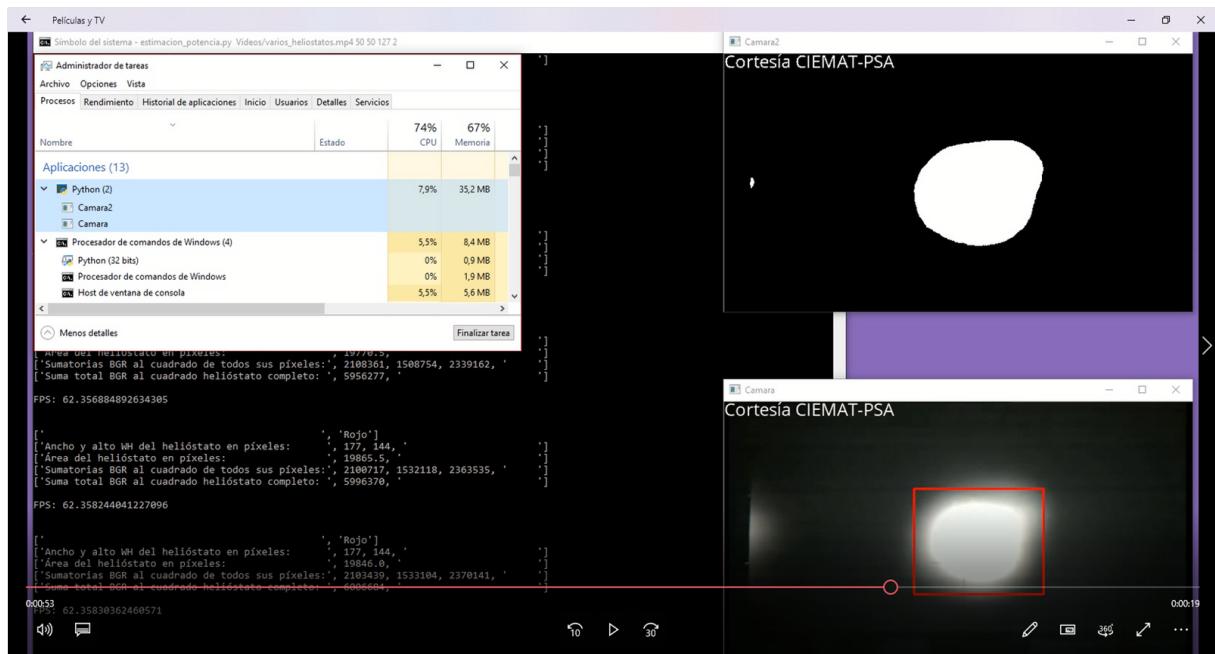


Figura 4.37: Instante de tiempo 15 de la ejecución del software para el vídeo 'varios\_heliostatos.mp4'.

Figura 4.37. Cuando el helióstato de la izquierda ya casi ni se ve porque este se ha separado bastante del helióstato del centro del vídeo, Python consume un 7,9 % de CPU y 35,2 MB de memoria RAM, mientras que la terminal de Windows consume un 5,5 % de CPU y 8,4 MB de memoria RAM. Los FPS son de 62,35.

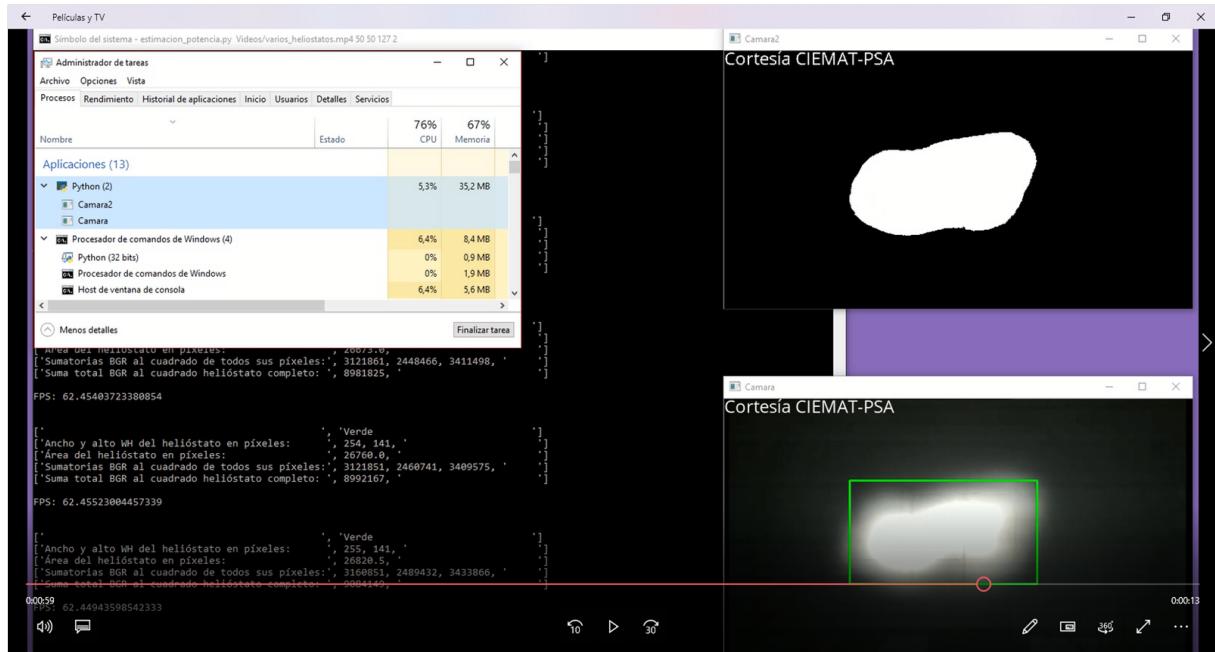


Figura 4.38: Instante de tiempo 16 de la ejecución del software para el vídeo 'varios\_heliostatos.mp4'.

Figura 4.38. Cuando otro (el tercero ya) de los helióstatos fusionados con el helióstato del centro del vídeo trata de salir del mismo (todavía no se ha completado este procedimiento). Python consume un 5,3 % de CPU y 35,2 MB de memoria RAM, mientras que la terminal de Windows consume un 6,4 % de CPU y 8,4 MB de memoria RAM. Los FPS son de 62,45.

Figura 4.39. Cuando los dos helióstatos están a punto de separarse entre sí. Python consume un 7,1 % de CPU y 35,2 MB de memoria RAM, mientras que la terminal de Windows consume un 6 % de CPU y 8,4 MB de memoria RAM. Los FPS son de 62,47.

Figura 4.40. Cuando el helióstato de la izquierda ya casi ni se ve porque este se ha separado bastante del helióstato del centro del vídeo, Python consume un 7,5 % de CPU y 35,2 MB de memoria RAM, mientras que la terminal de Windows consume un 5,9 % de CPU y 8,4 MB de memoria RAM. Los FPS son de 62,52.

Figura 4.41. Cuando el helióstato del centro del vídeo, el último de todos, empieza a trasladarse hacia la izquierda con el fin de salirse del vídeo, Python consume un 6,9 % de CPU y 35,2 MB de memoria RAM, mientras que la terminal de Windows consume un 7,1 % de CPU y 8,4 MB de memoria RAM. Los FPS son de 62,49.

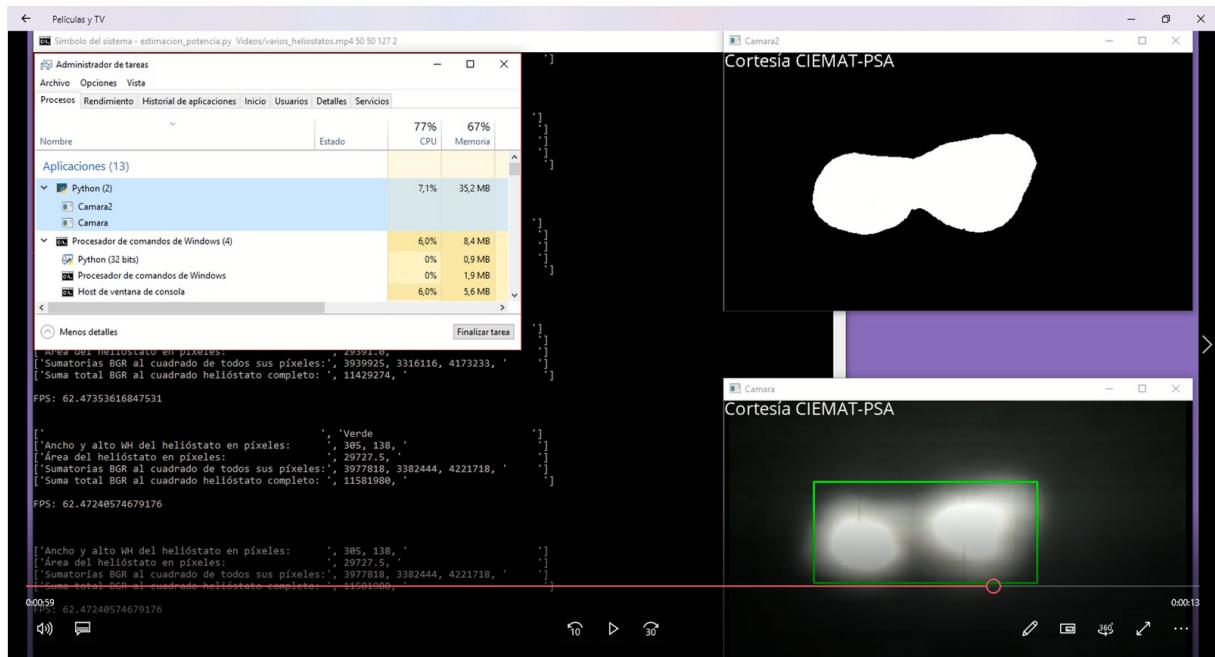


Figura 4.39: Instante de tiempo 17 de la ejecución del software para el vídeo 'varios\_heliostatos.mp4'.

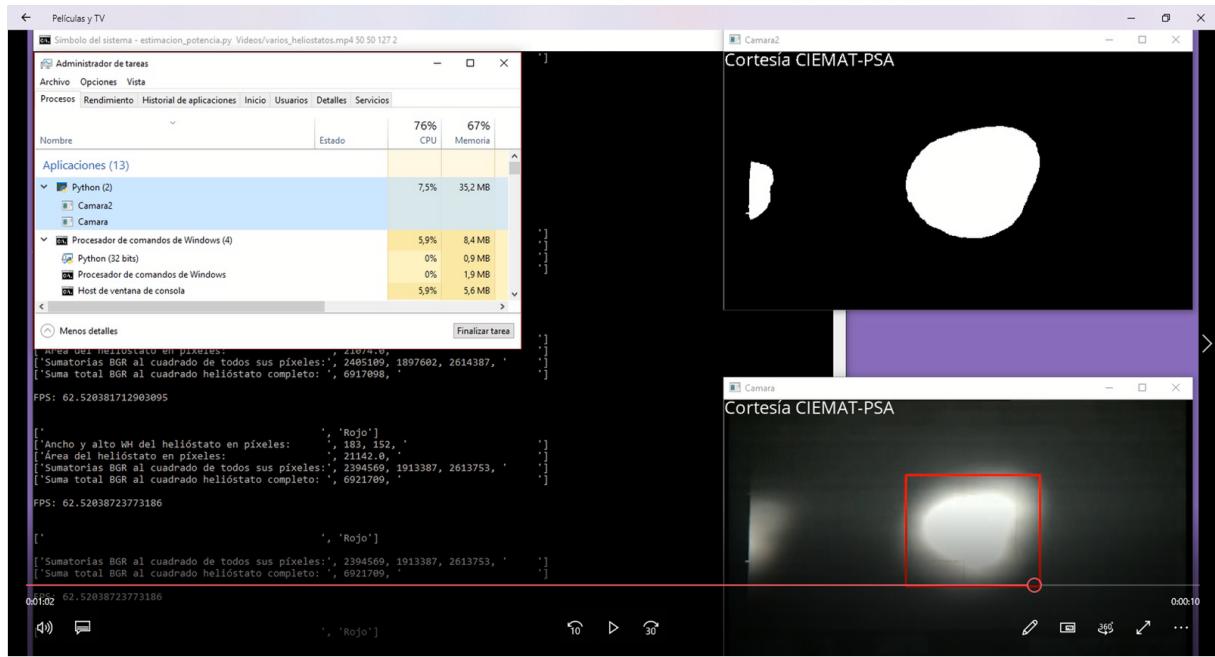


Figura 4.40: Instante de tiempo 18 de la ejecución del software para el vídeo 'varios\_heliostatos.mp4'.

Figura 4.42. Cuando dicho heliostato está a punto de desaparecer del vídeo (aún se muestra parcialmente a la izquierda del mismo), Python consume un 8,4 % de CPU y 35,2 MB de memoria RAM, mientras que la terminal de Windows consume un 6,3 % de CPU y 8,4 MB de memoria

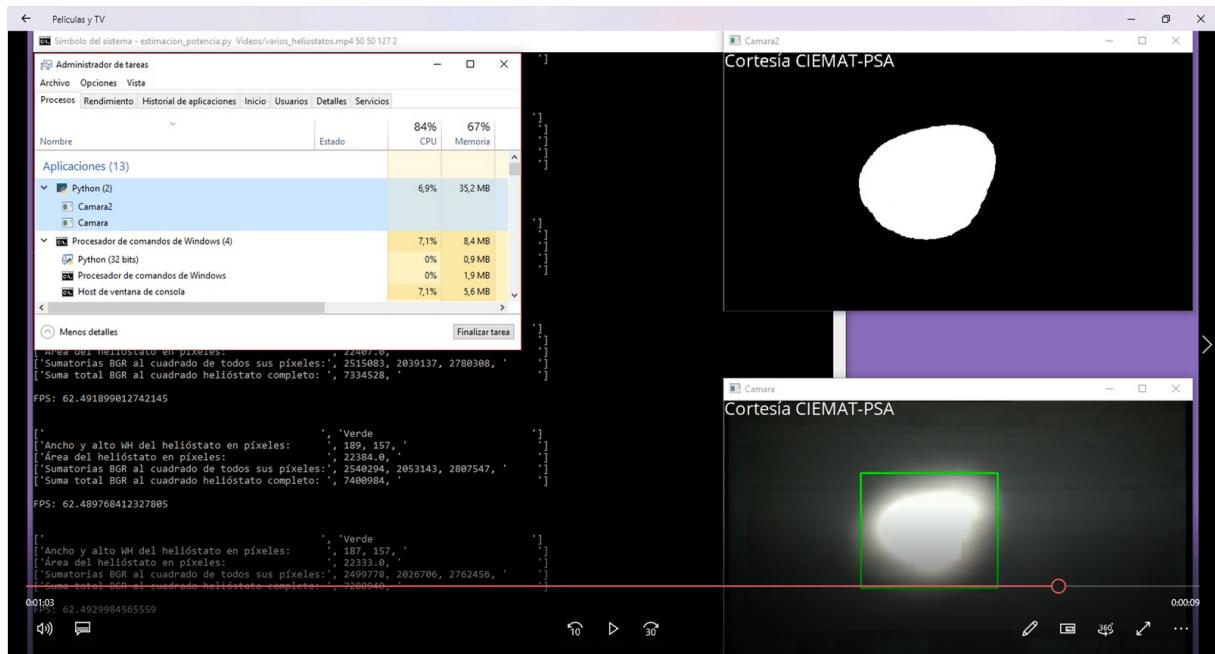


Figura 4.41: Instante de tiempo 19 de la ejecución del software para el vídeo 'varios\_heliostatos.mp4'.

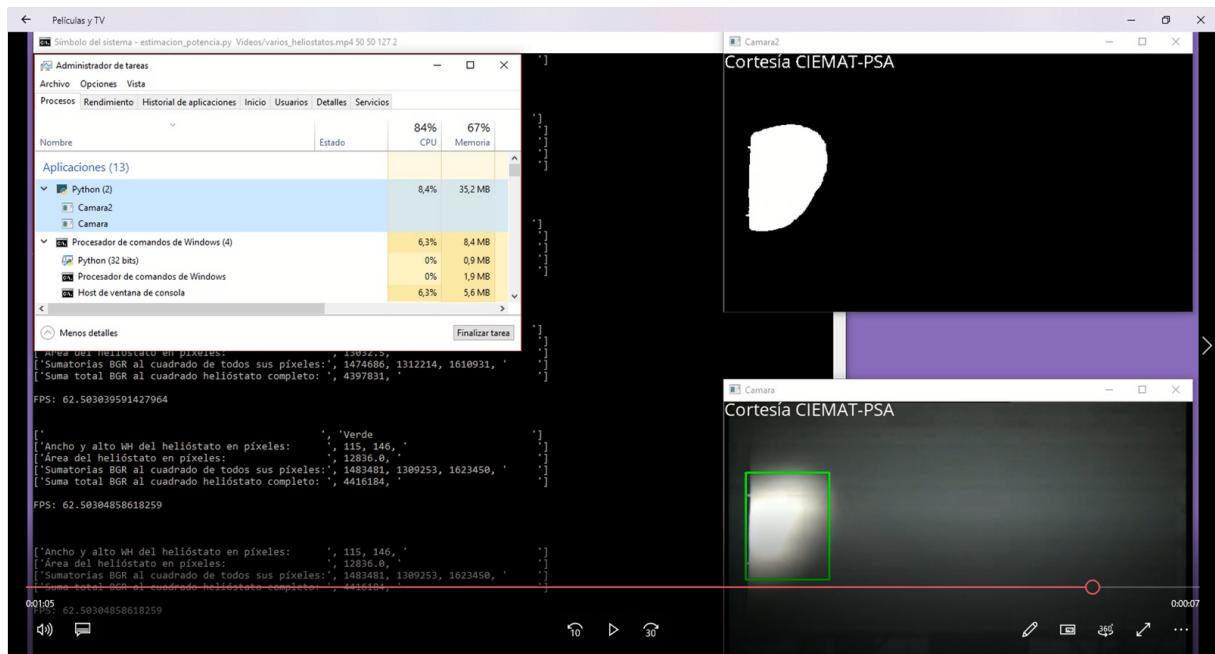


Figura 4.42: Instante de tiempo 20 de la ejecución del software para el vídeo 'varios\_heliostatos.mp4'.

RAM. Los FPS son de 62,50.

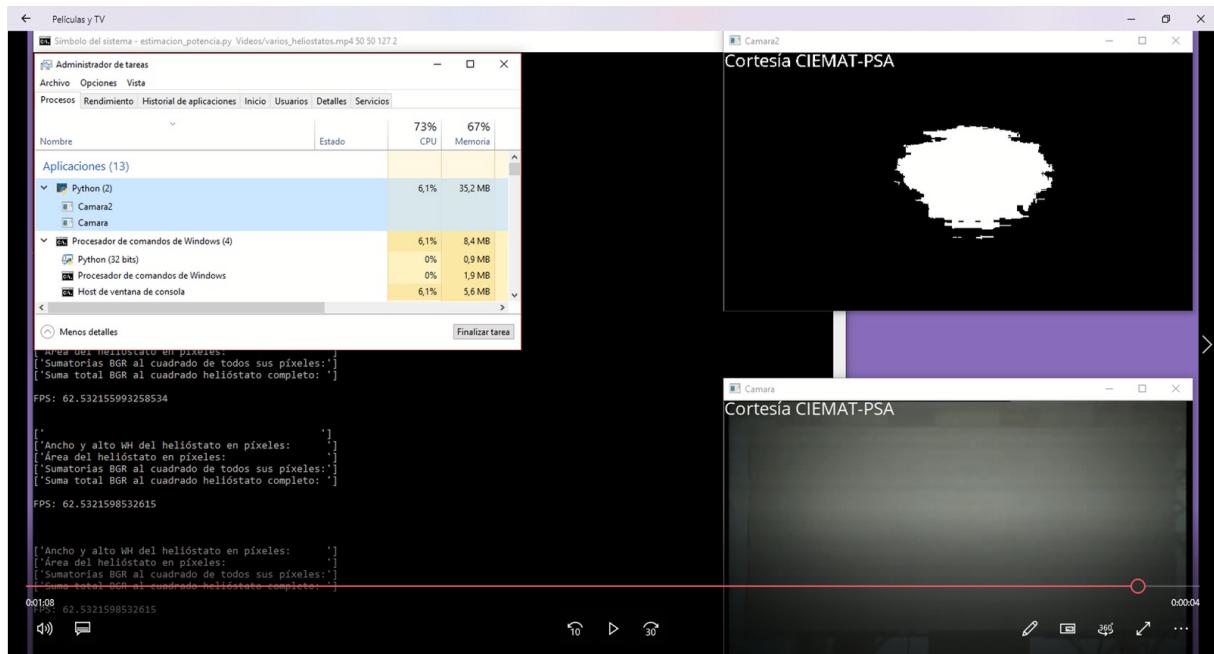


Figura 4.43: Instante de tiempo 21 de la ejecución del software para el vídeo 'varios\_heliostatos.mp4'.

Figura 4.43. Cuando finalmente no permanecen helióstatos en el vídeo (final del mismo, ya se han ido todos), Python consume un 6,1 % de CPU y 35,2 MB de memoria RAM, mientras que la terminal de Windows consume un 6,1 % de CPU y 8,4 MB de memoria RAM. Los FPS son de 62,53.

#### 4.9.2. Cargando el vídeo 'heliostato.MOV'

Se inicia la ejecución del software desde la terminal de Windows, estando en el directorio que contiene dicho software a ejecutar, y usando esta vez el comando `estimacion_potencia.py Videos/heliostato.MOV`.

Figura 4.44. Nada más iniciar la ejecución del software, cuando todavía no han pasado por el vídeo los helióstatos, ni siquiera el primero, Python consume un 45,6 % de CPU y 98,5 MB de memoria RAM, mientras que la terminal de Windows consume un 0,6 % de CPU y 8,6 MB de memoria RAM. Los FPS son de 20,18.

Figura 4.45. Cuando está entrando el primer helióstato en el vídeo (todavía no ha entrado completamente, sino parcialmente), Python consume un 42,1 % de CPU y 101,5 MB de memoria RAM, mientras que la terminal de Windows consume un 1,5 % de CPU y 8,6 MB de memoria RAM. Los FPS son de 20,55.

Figura 4.46. Cuando el primer helióstato ya ha llegado al centro del vídeo, Python consume un

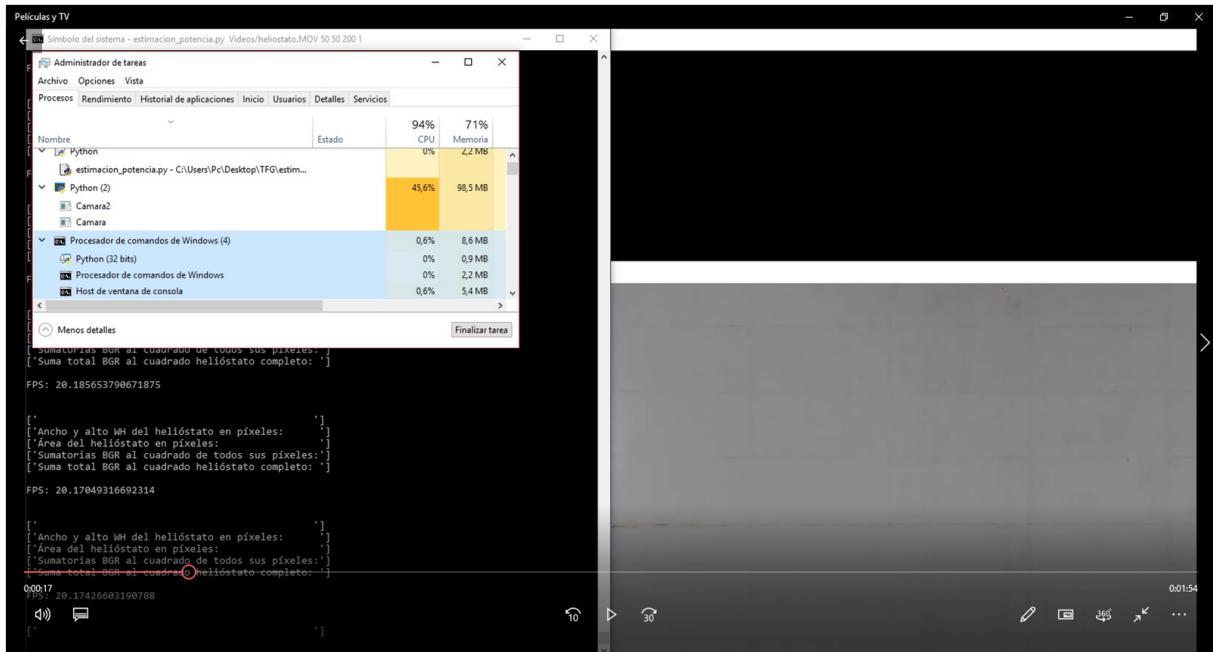


Figura 4.44: Instante de tiempo 1 de la ejecución del software para el vídeo 'heliostato.MOV'.

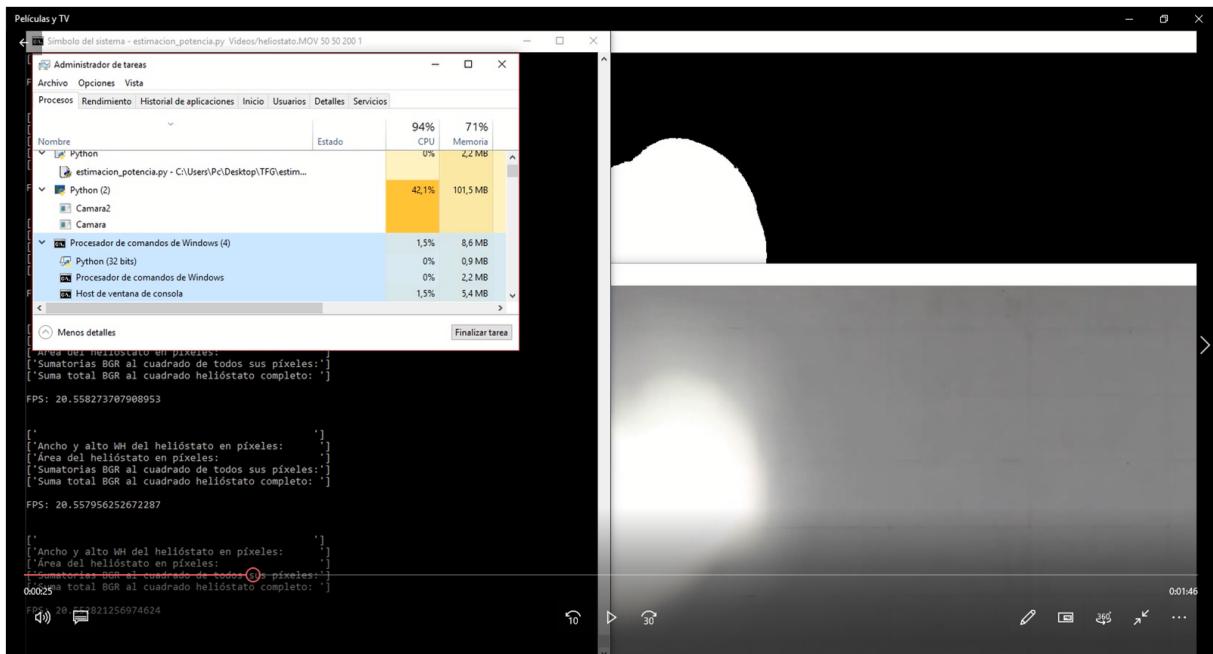


Figura 4.45: Instante de tiempo 2 de la ejecución del software para el vídeo 'heliostato.MOV'.

41,8 % de CPU y 101,0 MB de memoria RAM, mientras que la terminal de Windows consume un 1,8 % de CPU y 8,4 MB de memoria RAM. Los FPS son de 20,62.

Figura 4.47. Cuando el primer helióstato tiende a irse del centro del vídeo, Python consume un

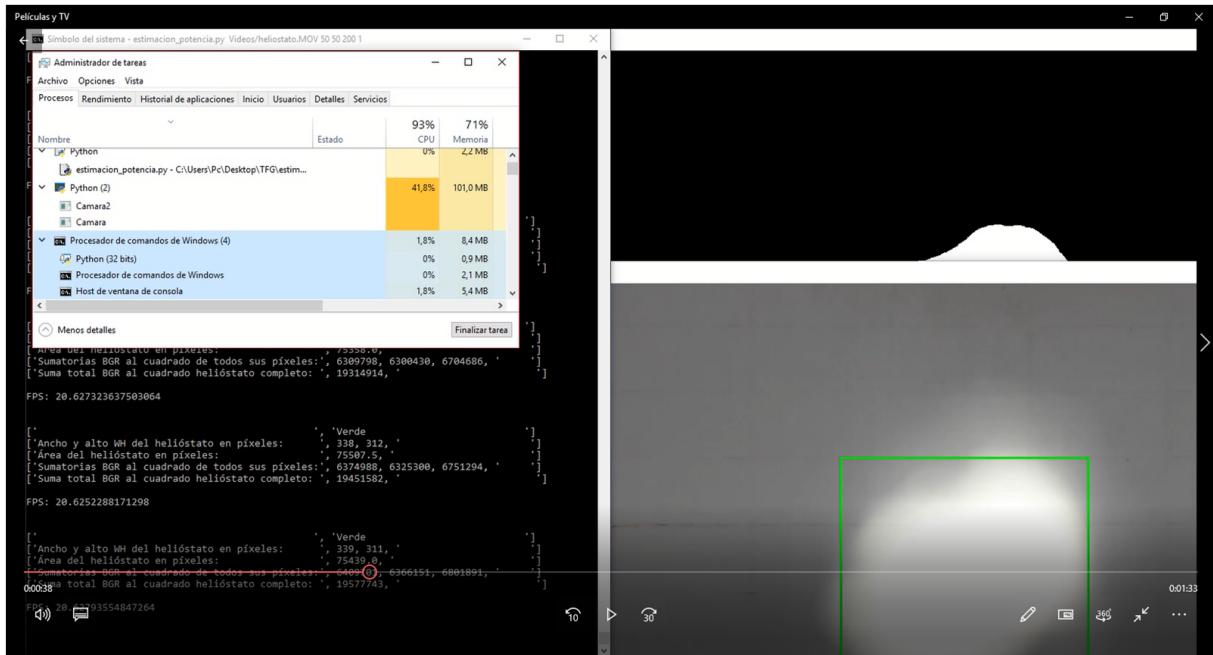


Figura 4.46: Instante de tiempo 3 de la ejecución del software para el vídeo 'heliostato.MOV'.

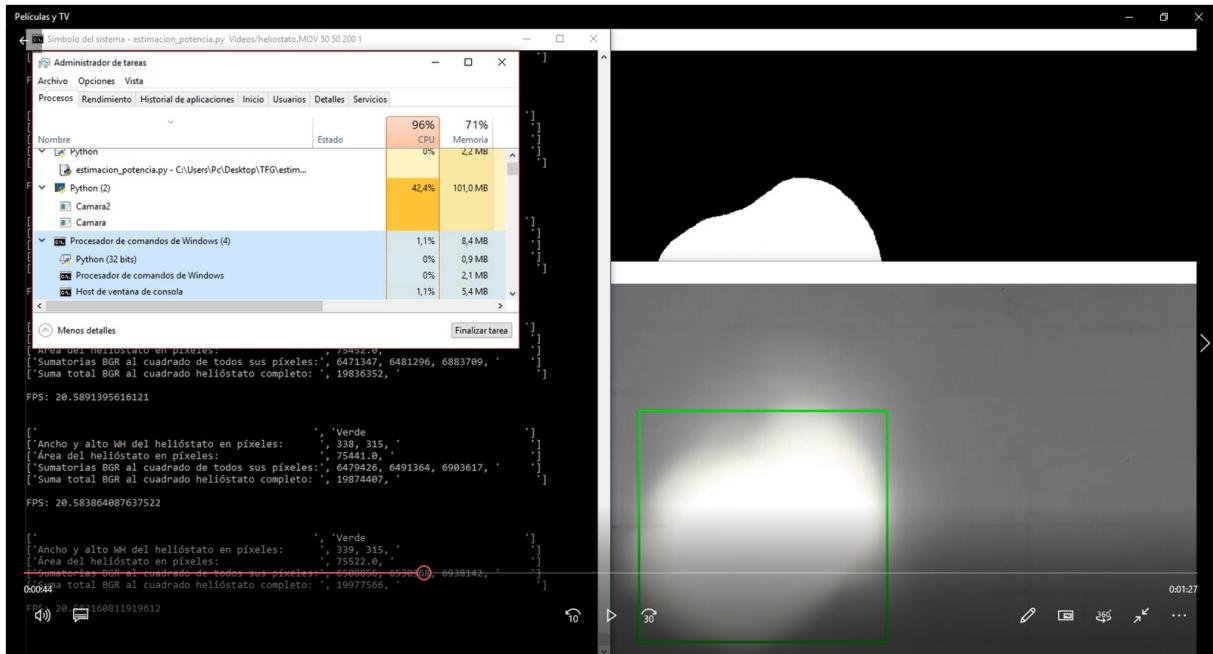


Figura 4.47: Instante de tiempo 4 de la ejecución del software para el vídeo 'heliostato.MOV'.

42,4 % de CPU y 101 MB de memoria RAM, mientras que la terminal de Windows consume un 1,1 % de CPU y 8,4 MB de memoria RAM. Los FPS son de 20,58.

Figura 4.48. Cuando está llegando al centro del vídeo un segundo heliostato desde la izquierda,

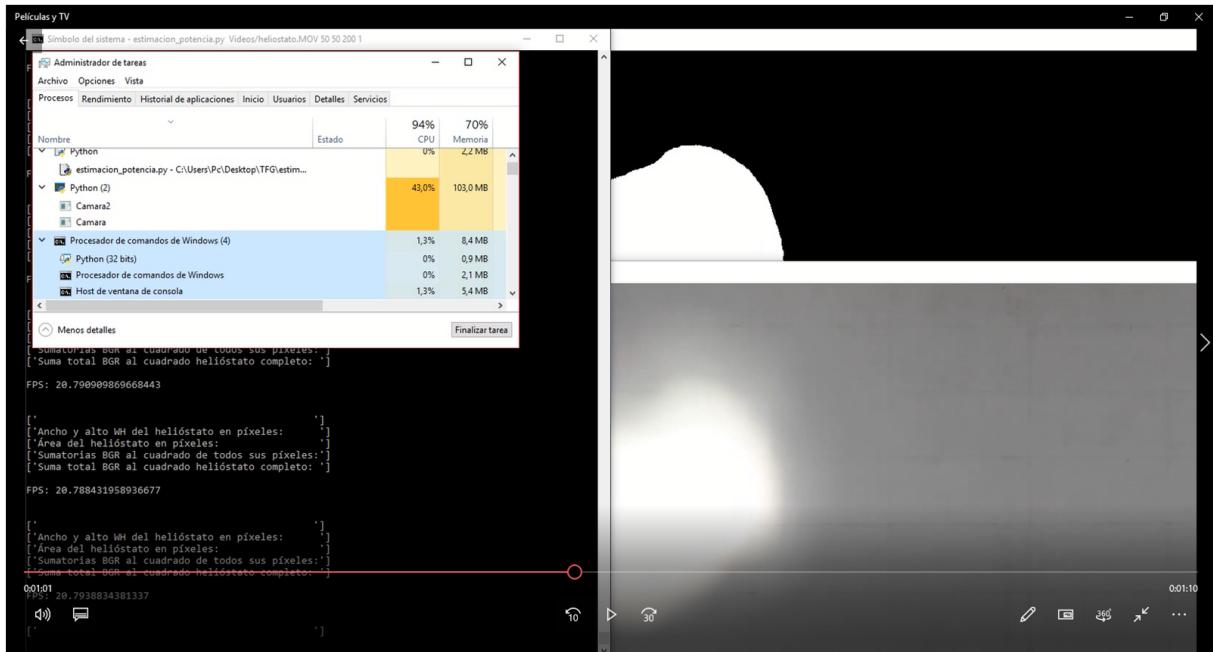


Figura 4.48: Instante de tiempo 5 de la ejecución del software para el vídeo 'heliostato.MOV'.

Python consume un 43 % de CPU y 103 MB de memoria RAM, mientras que la terminal de Windows consume un 1,3 % de CPU y 8,4 MB de memoria RAM. Los FPS son de 20,79.

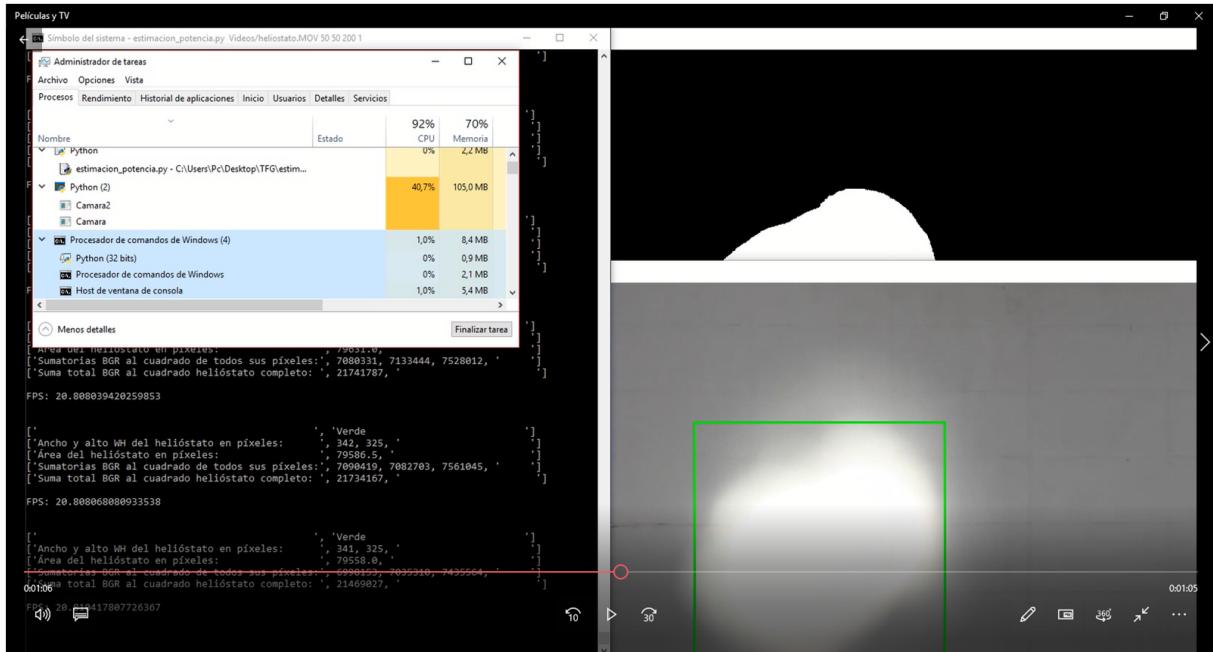


Figura 4.49: Instante de tiempo 6 de la ejecución del software para el vídeo 'heliostato.MOV'.

Figura 4.49. Cuando el segundo heliostato se ha aproximado un poco más al centro del vídeo

(respecto al instante de tiempo anterior), Python consume un 40,7% de CPU y 105 MB de memoria RAM, mientras que la terminal de Windows consume un 1% de CPU y 8,4 MB de memoria RAM. Los FPS son de 20,8.

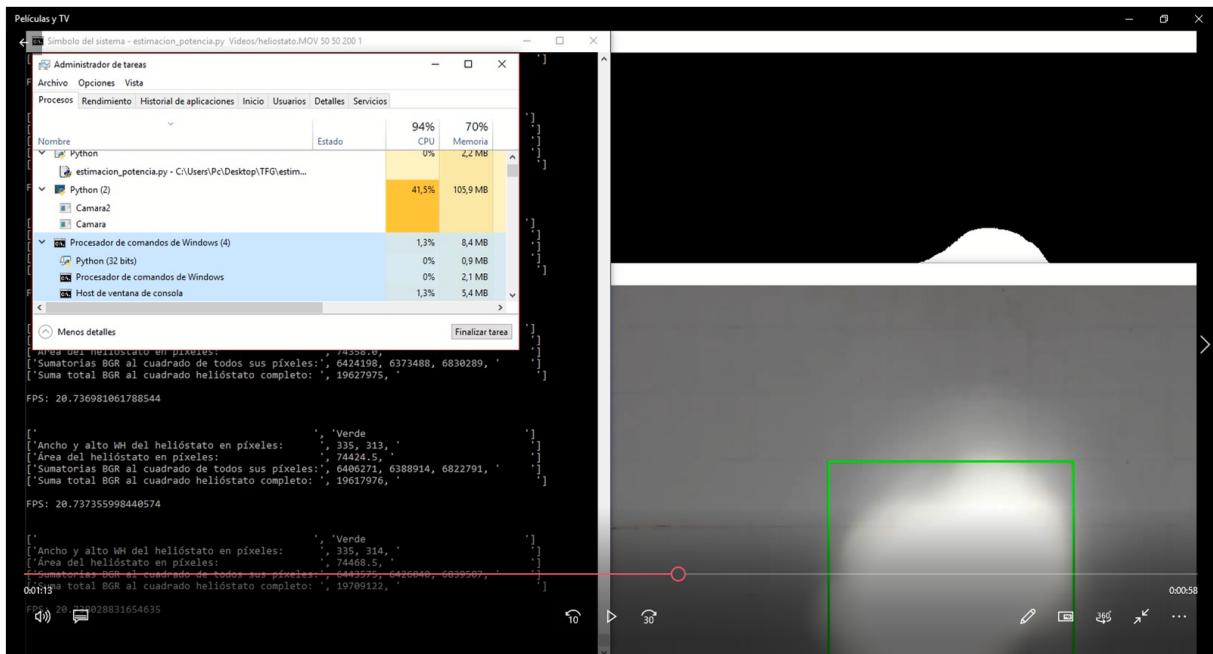


Figura 4.50: Instante de tiempo 7 de la ejecución del software para el vídeo 'heliostato.MOV'.

Figura 4.50. Cuando el segundo helióstato ya ha llegado al centro del vídeo, Python consume un 41,5 % de CPU y 105,9 MB de memoria RAM, mientras que la terminal de Windows consume un 1,3 % de CPU y 8,4 MB de memoria RAM. Los FPS son de 20,73.

Figura 4.51. Cuando el segundo helióstato tiende a irse del centro del vídeo, Python consume un 36,2 % de CPU y 103,1 MB de memoria RAM, mientras que la terminal de Windows consume un 0,9 % de CPU y 8,4 MB de memoria RAM. Los FPS son de 20,58.

Figura 4.52. Cuando el segundo helióstato ya ha abandonado el vídeo y aún no ha llegado el tercero, Python consume un 40,3 % de CPU y 102,0 MB de memoria RAM, mientras que la terminal de Windows consume un 0,8 % de CPU y 8,4 MB de memoria RAM. Los FPS son de 20,59.

Figura 4.53. Cuando el tercer helióstato está entrando en el vídeo (todavía no ha entrado completamente, sino parcialmente), Python consume un 44,3 % de CPU y 105,3 MB de memoria RAM, mientras que la terminal de Windows consume un 0 % de CPU y 8,4 MB de memoria RAM. Los FPS son de 20,65.

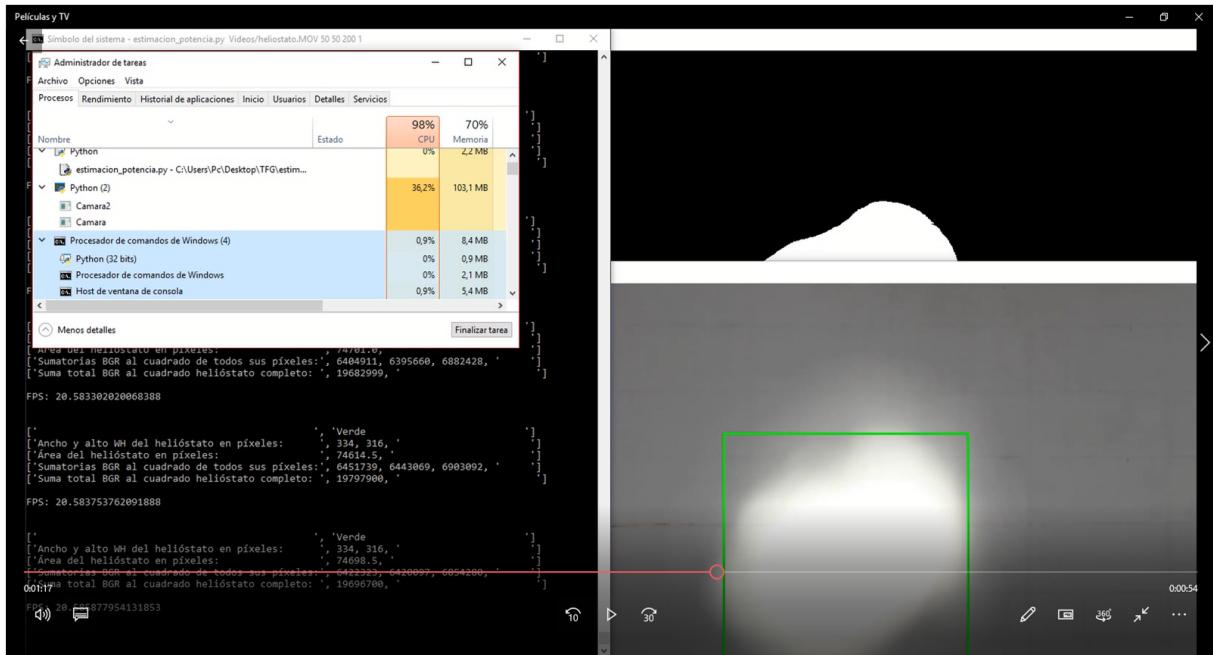


Figura 4.51: Instante de tiempo 8 de la ejecución del software para el vídeo 'heliostato.MOV'.

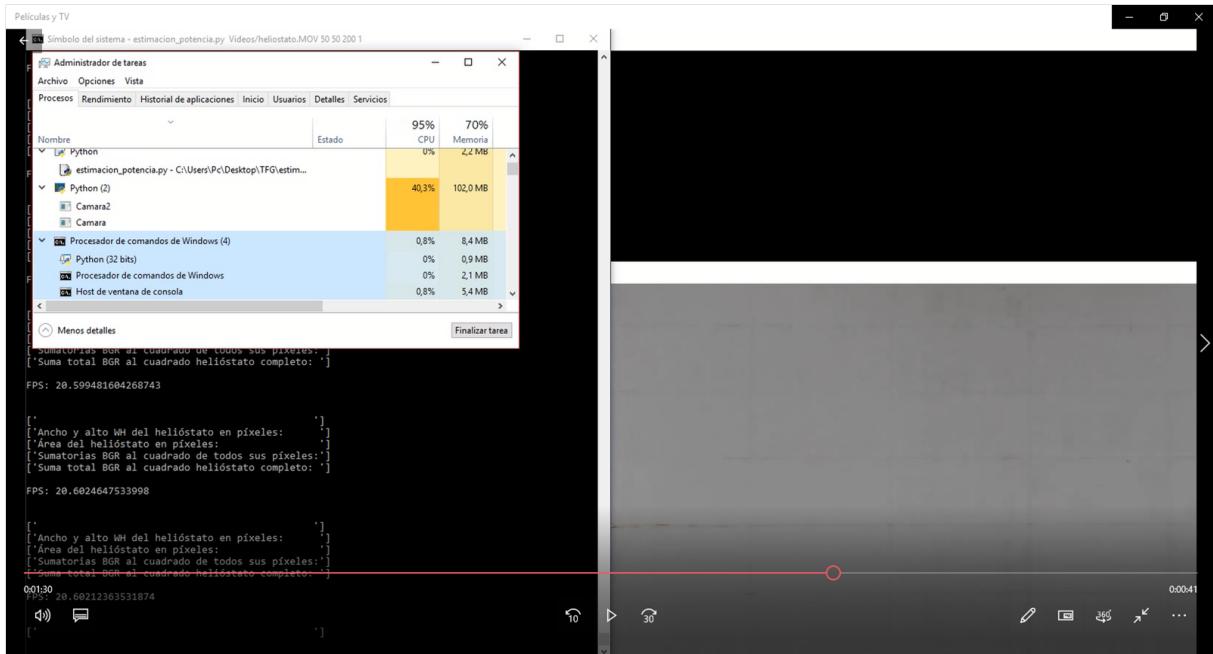


Figura 4.52: Instante de tiempo 9 de la ejecución del software para el vídeo 'heliostato.MOV'.

Figura 4.54. Cuando el tercer helióstato está a punto de alcanzar el centro del vídeo, Python consume un 42,9 % de CPU y 101,1 MB de memoria RAM, mientras que la terminal de Windows consume un 1,3 % de CPU y 8,4 MB de memoria RAM. Los FPS son de 20,67.

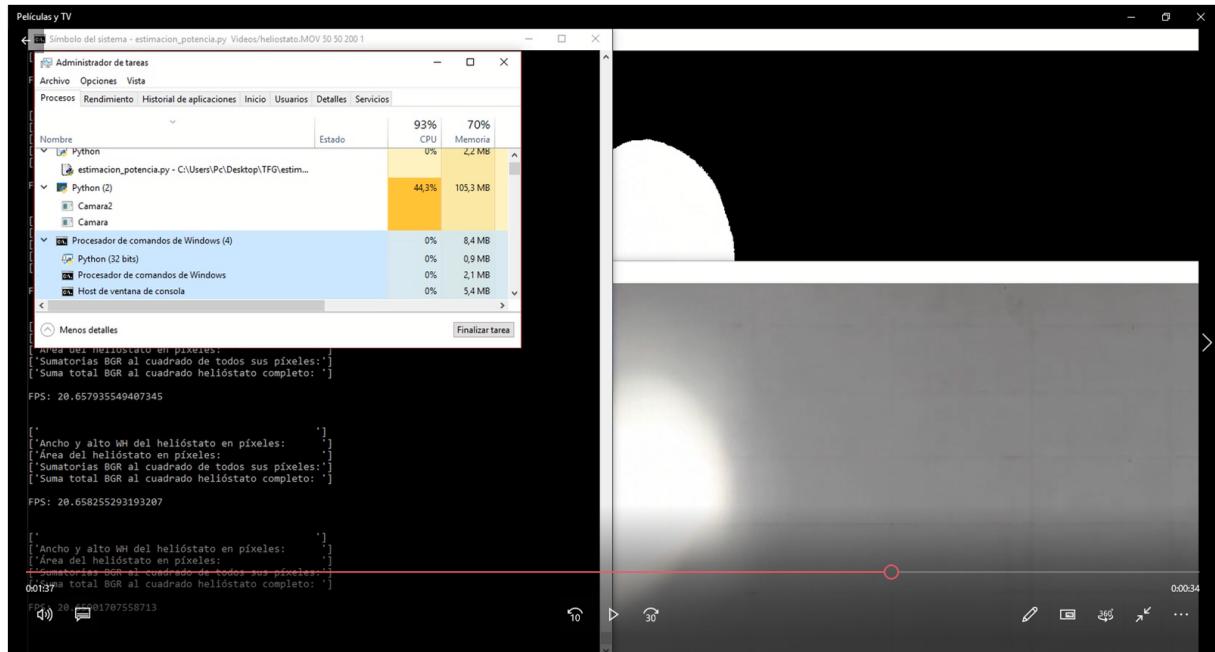


Figura 4.53: Instante de tiempo 10 de la ejecución del software para el vídeo 'heliostato.MOV'.

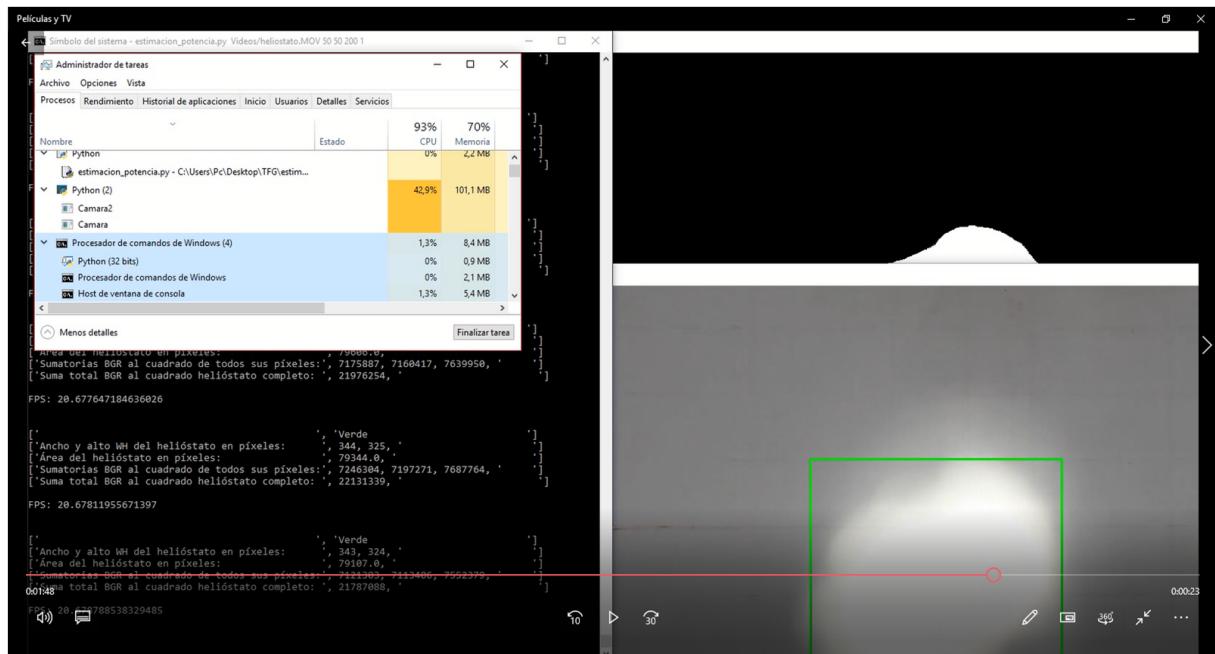


Figura 4.54: Instante de tiempo 11 de la ejecución del software para el vídeo 'heliostato.MOV'.

Figura 4.55. Cuando el tercer helióstato se está alejando del centro del vídeo, Python consume un 39,2 % de CPU y 103,1 MB de memoria RAM, mientras que la terminal de Windows consume un 1,5 % de CPU y 8,4 MB de memoria RAM. Los FPS son de 20,64.

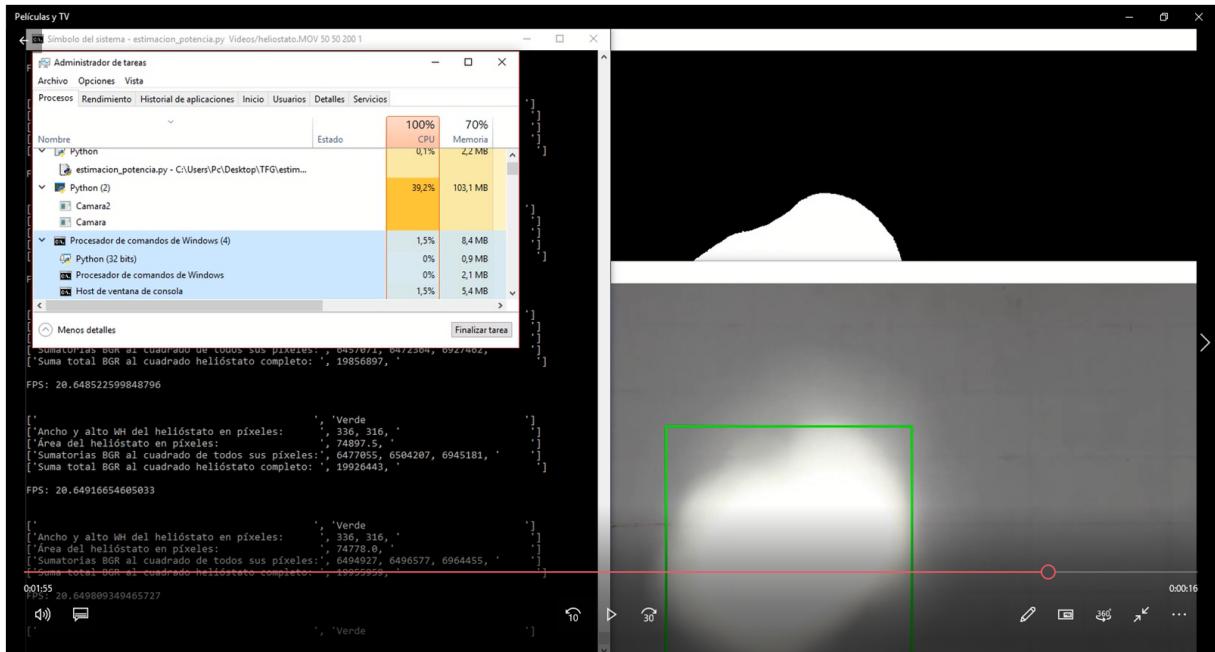


Figura 4.55: Instante de tiempo 12 de la ejecución del software para el vídeo 'heliostato.MOV'.

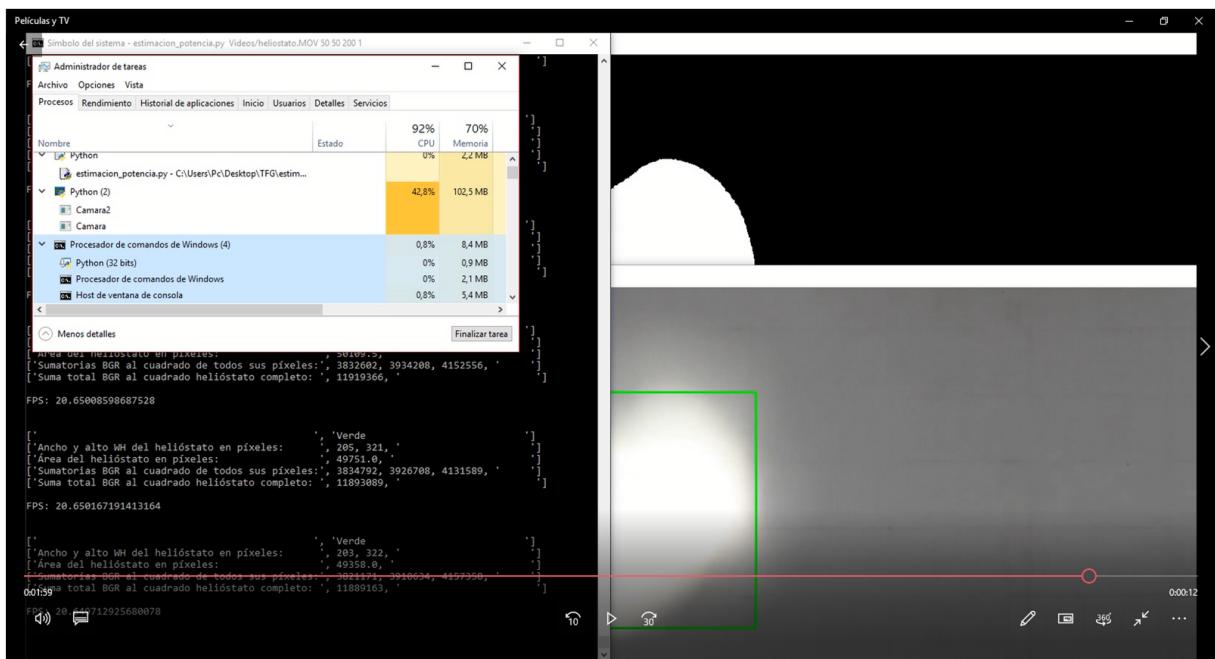


Figura 4.56: Instante de tiempo 13 de la ejecución del software para el vídeo 'heliostato.MOV'.

Figura 4.56. Cuando el tercer helióstato está a punto de abandonar el vídeo, Python consume un 42,8% de CPU y 102,5 MB de memoria RAM, mientras que la terminal de Windows consume un 0,8% de CPU y 8,4 MB de memoria RAM. Los FPS son de 20,65.

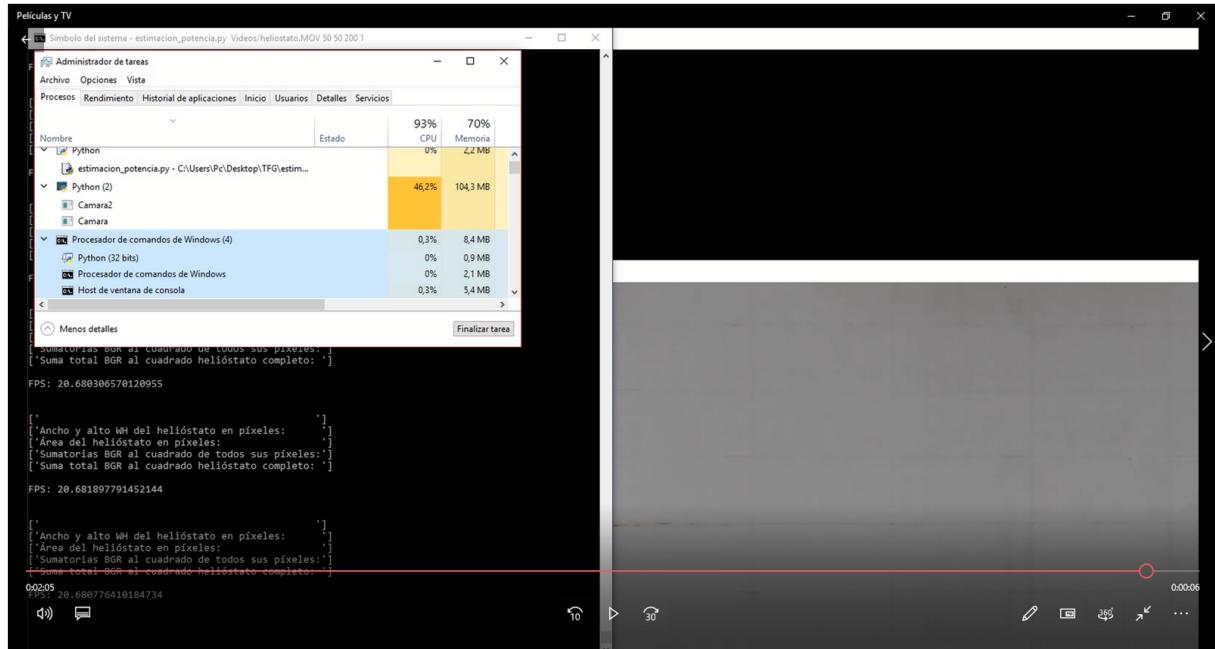


Figura 4.57: Instante de tiempo 14 de la ejecución del software para el vídeo 'heliostato.MOV'.

Figura 4.57. Cuando no hay ningún helióstato en el vídeo (todos ya han pasado por él, y no quedan más), Python consume un 46,2 % de CPU y 104,3 MB de memoria RAM, mientras que la terminal de Windows consume un 0,3 % de CPU y 8,4 MB de memoria RAM. Los FPS son de 20,68.

Las figuras 4.58 y 4.59 muestran dos histogramas a modo resumen de los resultados comentados en este apartado.

## 4.10. Plataformas que son capaces de ejecutar el software

El software está escrito en lenguaje de programación Python, y una de las características de Python es que es un lenguaje multiplataforma. Esto significa que, aunque originalmente Python se diseñó para Unix, cualquier sistema es compatible con el lenguaje siempre y cuando exista un intérprete programado para él. Además, hay diversas versiones de Python en muchos sistemas informáticos distintos.

## 4.11. Ejemplos de ejecuciones y comentarios sobre sus resultados

### 4.11.1. Cargando el vídeo 'variosHeliostatos.mp4'

Al iniciar la ejecución del software desde la terminal de Windows, estando en el directorio que contiene dicho software a ejecutar, y usando el comando `estimacion_potencia.py Videos/varios_heliostatos` ocurre lo siguiente:

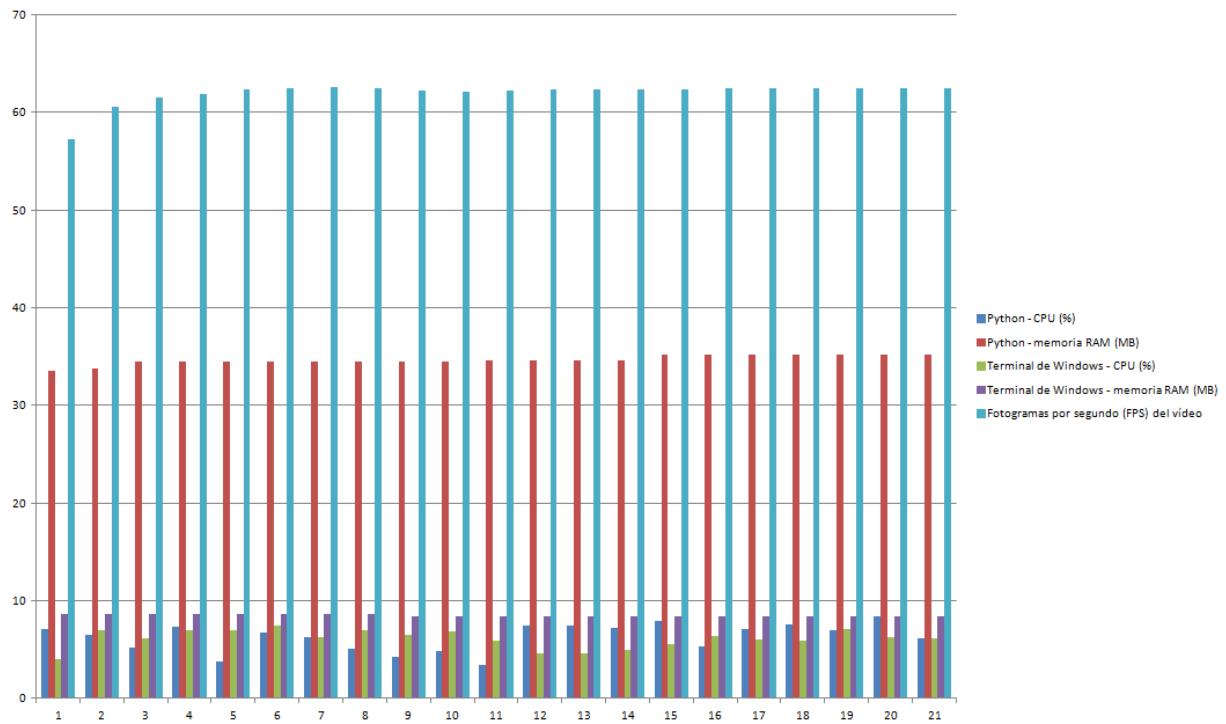


Figura 4.58: Histograma para el caso/comando 'estimacion\_potencia.py Videos/varios\_heliostatos.mp4 50 50 127 2'.

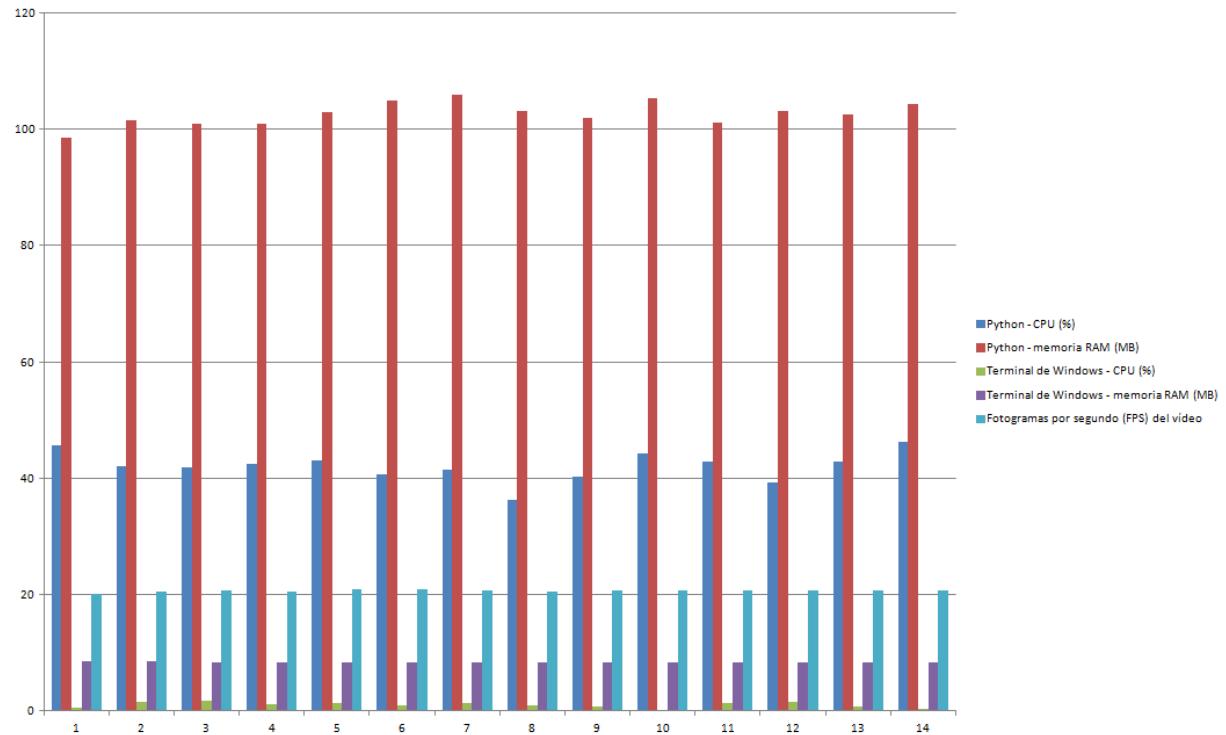


Figura 4.59: Histograma para el caso/comando 'estimacion\_potencia.py Videos/heliostato.MOV 50 50 200 1'.

Cuando un helióstato comienza a entrar en el vídeo de helióstatos, el programa todavía no lo reencuadra en dicho vídeo ni lo analiza. Esto se debe a que, en este instante de tiempo, el helióstato mostrado en el vídeo aún no es de ancho y alto 50 píxeles o más, tal y como solicitó el usuario por parámetros en la consola. De forma análoga ocurre cuando un helióstato está a punto de salir del vídeo: en ese instante apenas se muestra el helióstato en el vídeo, y el ancho o el alto mostrados no son mayores de 50 píxeles, siendo ignorado por el programa. También debe cumplirse el valor de umbral proporcionado por parámetro para el análisis del helióstato, 127. De lo contrario, si el helióstato se muestra en su totalidad en el vídeo, o por lo menos su ancho y alto en un instante determinado son mayores de 50 píxeles (y por supuesto el umbral del vídeo es al menos de 127), el programa lo reencuadra en el vídeo y lo analiza.

Por parámetro en la consola, se ha indicado también que se desea leer hasta un máximo de dos heliosztatos por fotograma del vídeo, porque en este vídeo pueden mostrarse hasta dos heliosztatos en un mismo fotograma. Si en vez de '2' se introduce '1', cuando aparezcan dos heliosztatos en un mismo fotograma, simplemente se reencuadrará en el vídeo y procesará el heliosztato situado a la izquierda, y no el de la derecha.

En el análisis de un heliosztato, el programa calculará su ancho y alto, su área total (todo estos valores se miden en píxeles), la sumatoria acumulativa de los valores de cada componente BGR al cuadrado (cada componente por separado) de todo el heliosztato, y esto mismo pero sumando las tres componentes resultantes entre sí. Los resultados de estos datos son mostrados por consola, para cada heliosztato analizado. En la consola, dichos datos son representados textualmente así (respectivamente):

Ancho y alto WH **del** heliosztato en píxeles .

Área **del** heliosztato en píxeles .

Sumatorias BGR al cuadrado de todos sus píxeles .

Suma total BGR al cuadrado heliosztato completo .

En el análisis de un heliosztato, además, el programa reencuadrará en el vídeo y en tiempo de ejecución los heliosztatos que se están analizando actualmente. Si en el fotograma actual de dicho vídeo simplemente aparece un heliosztato, se reencuadrará en color verde. De lo contrario, si aparecen dos heliosztatos al mismo tiempo en un mismo fotograma, el heliosztato de la izquierda se reencuadrará en verde y el de la derecha en rojo. En cualquiera de los casos, y tal y como se comentó antes, el programa analizará el o los heliosztatos y mostrará sus resultados en la consola. Tener en cuenta que cuando se analizan dos heliosztatos en un mismo fotograma, el programa mostrará sus respectivos resultados en la consola en dos columnas. La primera para el heliosztato reencuadrado en verde, y la segunda para el heliosztato reencuadrado en rojo. Finalmente, indicar que cuando dos heliosztatos, en un mismo fotograma, están fusionados (parcialmente o totalmente) o rozándose, serán tratados y analizados como si fueran uno solo.

La tasa de fotogramas por segundo (FPS) es alta, de aproximadamente 60 FPS, porque se está procesando un vídeo de dimensiones reducidas, y gracias también al uso de matrices vectorizadas (biblioteca 'NumPy' de Python) para la carga, procesamiento y guardado de todos los datos de todos los píxeles de cada heliosztato.

Todo este procedimiento se repite para todos los demás fotogramas del vídeo de heliosztatos, hasta que se hayan leído todos sus fotogramas, finalizando así la ejecución del programa.

El contenido de este vídeo son de cuatro heliosztatos que entran poco a poco en el vídeo desde el lado izquierdo, y se van ubicando en el centro. Los demás (y siguientes) heliosztatos que también

llegan desde la izquierda se solapan con el helióstato o helióstatos permanecidos en el centro de dicho vídeo. Cuando los cuatro helióstatos permanezcan solapados en el centro del vídeo, se irán separando, uno por uno, los helióstatos hacia la izquierda, saliéndose cada uno del vídeo, hasta que no queden ninguno.

#### 4.11.2. Cargando el vídeo 'heliostato.MOV'

Al iniciar la ejecución del software desde la terminal de Windows, estando en el directorio que contiene dicho software a ejecutar, y usando esta vez el comando `estimacion_potencia.py Videos/heliostato.MOV` ocurre lo siguiente:

Para que el helióstato sea en esta ocasión reencuadrado en el vídeo y analizado, el nuevo umbral escogido es de 200, porque el nuevo vídeo tiene una tonalidad de color más clara. En el anterior caso, bastaba con un umbral de 127 porque aquel vídeo era más oscuro. El ancho y alto escogidos del helióstato, al igual que en el caso anterior, es de 50 por 50 píxeles.

Como en este vídeo aparece como máximo un único helióstato por fotograma, se indica por parámetro en la consola que se desea analizar esta vez 1 helióstato por fotograma (en vez de 2, como sucedía en el anterior caso). Si se escribe un número mayor a 1, provocaría en el programa un error de fuera de rango.

Al ser un vídeo con unas dimensiones muy elevadas, a comparación del anterior vídeo usado en aquel caso, la tasa de fotogramas por segundo (FPS) se reduce considerablemente, y oscila en 20 FPS. Esto es debido a que el programa le llevará más tiempo analizar los helióstatos y sus píxeles, ya que dichos helióstatos tendrán unas dimensiones más elevadas, y por tanto, más píxeles a analizar dentro del helióstato.

Como en el anterior caso, el programa reencuadra el helióstato en el vídeo y lo analiza, siempre y cuando se cumplan los requisitos deseados por el usuario por parámetros en la consola, es decir, de qué tamaño y umbral debe ser el helióstato. Y también se mostrarán los resultados por consola de los helióstatos analizados por el programa. Aparece la misma información de los resultados de los helióstatos que en el anterior caso.

Todo este procedimiento se repite para todos los demás fotogramas del vídeo de helióstatos, hasta que se hayan leído todos sus fotogramas, finalizando así la ejecución del programa.

El contenido de este vídeo es de tres helióstatos que entran poco a poco en el vídeo (desde la izquierda), permanecen en el centro durante unos segundos, y se salen poco a poco (hacia la izquierda) del vídeo. Los tres helióstatos hacen esto, de uno en uno. No aparecen dos helióstatos en un mismo fotograma del vídeo como en el anterior caso, así que por tanto no habrán fusiones ni rozamientos de dos helióstatos distintos.

### 4.12. Validación cualitativa de la función de estimación de potencia

La gráfica de la figura 4.60 muestra el tamaño del helióstato reencuadrado en verde en diferentes partes del vídeo, siendo el eje de abscisas X el tiempo del vídeo en segundos y el eje de ordenadas Y el tamaño del helióstato en píxeles, para el caso `estimacion_potencia.py Videos/varios_heliostatos.mp4`.

Al comienzo del vídeo, el helióstato va entrando poco a poco a él, y por eso, la gráfica crece gradualmente en esa región. Luego, el helióstato permanece unos segundos en el centro del vídeo, donde la gráfica adquiere un resultado prácticamente constante. Cuando entra otro helióstato en

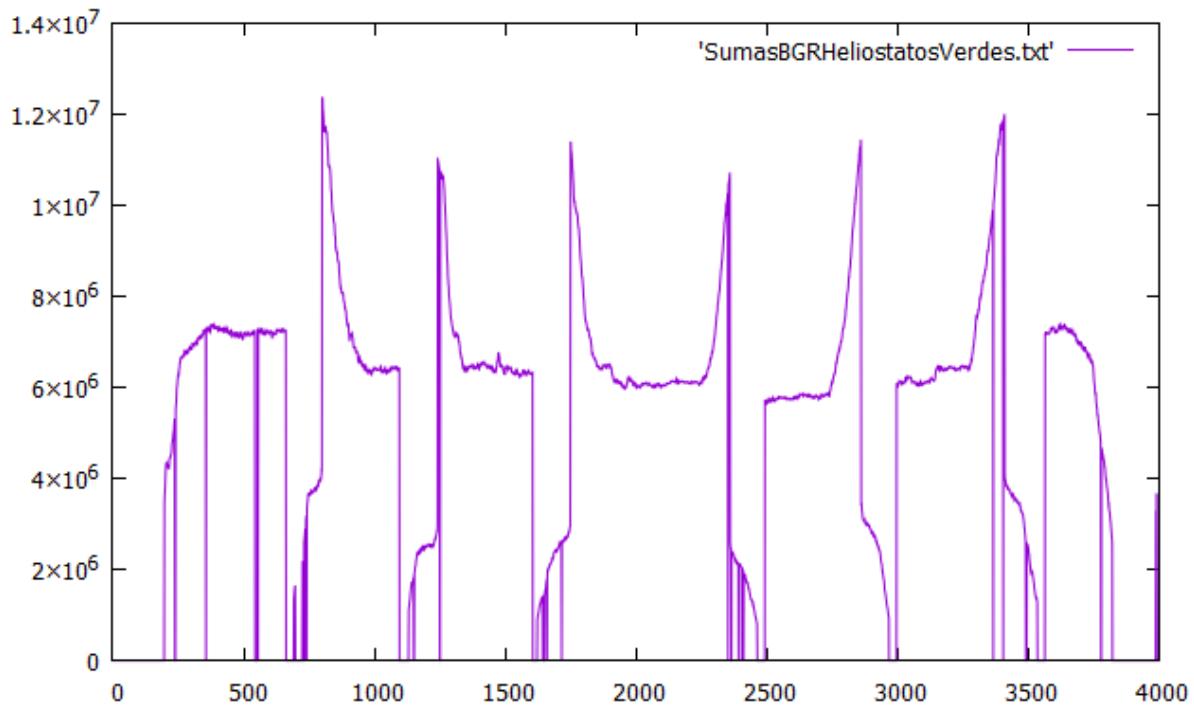


Figura 4.60: Estimación de potencia de los helióstatos verdes en las distintas partes del vídeo 'varios\_heliostatos.mp4'.

el vídeo, y por tanto, se estarán mostrando dos helióstatos a la vez en un mismo fotograma, el helióstato del centro cambia su recuadro de verde a rojo, y es por eso que el valor de la gráfica de repente cae en picado y hacia valores nulos, pero el helióstato nuevo que entra desde la izquierda es reencuadrado en verde, y por tanto la gráfica vuelve a crecer gradualmente. Cuando los dos helióstatos mostrados en el mismo fotograma se rozan entre sí, se obtendría un único helióstato muy grande, por lo que la gráfica ahora crece de golpe, pero después volverá a decrecer poco a poco hasta obtener un valor medio. Sería el caso cuando los dos helióstatos que se estaban rozando, ahora se van fusionando. Este procedimiento continúa hasta que todos los helióstatos hayan entrado y fusionado en el vídeo.

En la mitad del vídeo, se muestra un helióstato en el centro del mismo, combinación de todos los helióstatos (cuatro) que se han ido fusionando previamente. La gráfica adquiere un valor de suma BGR medio. Luego, el primer helióstato se irá separando poco a poco de los demás, pero como aún esos dos helióstatos se están rozando (desfusionándose entre sí), es considerado como un único helióstato grande reencuadrado en verde, y la gráfica vuelve a crecer hacia altos valores de suma BGR. Cuando los dos helióstatos se han separado totalmente, el de la izquierda, reencuadrado en verde, se irá poco a poco hacia la izquierda del vídeo, y el del medio (este aún posee realmente tres helióstatos fusionados entre sí), reencuadrado en rojo, seguirá permaneciendo en esa ubicación. El valor de la gráfica, midiendo el reencuadre verde, irá bajando poco a poco, hasta adquirir un valor nulo, que sería el caso de cuando el helióstato de la izquierda se ha ido totalmente del vídeo. Cuando solo permanece un helióstato en el centro del vídeo, este será reencuadrado en verde (en vez de rojo como estaba antes), y por tanto, la gráfica volverá a adquirir en un único instante de tiempo un valor de suma BGR medio. Este procedimiento continúa hasta que todos los helióstatos se hayan separado (desfusionados entre

sí) y abandonado totalmente el vídeo.

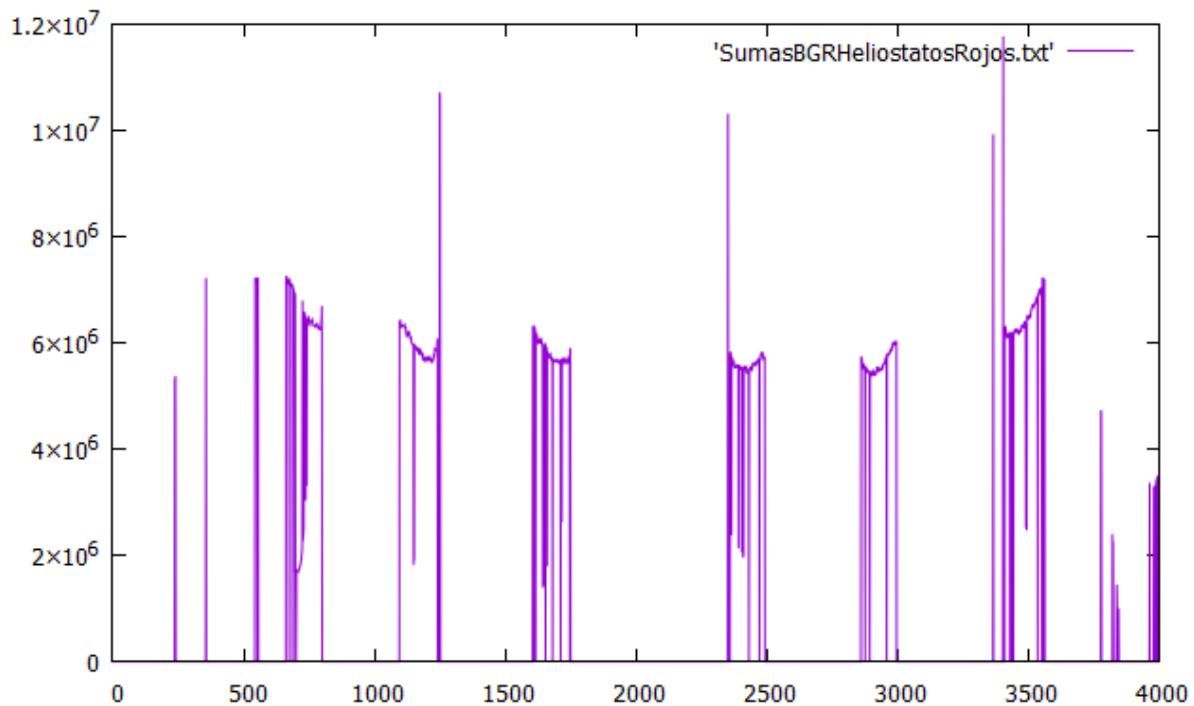


Figura 4.61: Estimación de potencia de los heliostatos rojos en las distintas partes del vídeo 'varios\_heliostatos.mp4'.

La gráfica de la figura 4.61 muestra el tamaño del helióstato reencuadrado en rojo en diferentes partes del vídeo, siendo el eje de abscisas X el tiempo del vídeo en segundos y el eje de ordenadas Y el tamaño del helióstato en píxeles, para el caso `estimacion_potencia.py Videos/varios_heliostatos.mp4`

La gráfica, cuando el helióstato con el reencuadre en rojo es visible en el vídeo, muestra un valor de suma BGR constante e intermedio, porque dicho helióstato solo aparece cuando en un mismo fotograma se muestran dos heliostatos separados entre sí: uno a la izquierda (reencuadre verde), y otro en el centro (reencuadre rojo). Y cuando el helióstato con el reencuadre en rojo aparece, suele tener un valor de suma BGR constante.

En los demás casos, cuando el helióstato con el reencuadre en rojo no aparece en el vídeo, el valor de suma BGR en la gráfica es nulo.

Debido probablemente al ruido del vídeo, se aprecia en ciertas partes de la gráfica que el valor del helióstato reencuadrado en rojo crece enormemente, cosa que no tiene sentido porque su valor de suma BGR, como se comentó antes, es mediano y prácticamente constante, cuando dicho helióstato se muestra en el vídeo.

La gráfica de la figura 4.62 muestra el tamaño del helióstato reencuadrado en verde en diferentes partes del vídeo, siendo el eje de abscisas X el tiempo del vídeo en segundos y el eje de ordenadas Y el tamaño del helióstato en píxeles, para el caso `estimacion_potencia.py Videos/heliostato.MOV 50 50`

Cuando el helióstato entra poco a poco en el vídeo, pero aún no ha entrado totalmente en él, no es reencuadrado aún, y por tanto no se obtienen valores en la gráfica. Pero a partir del instante en el que entra completamente en el vídeo, pero aún no ha llegado al centro, sí es reencuadrado,

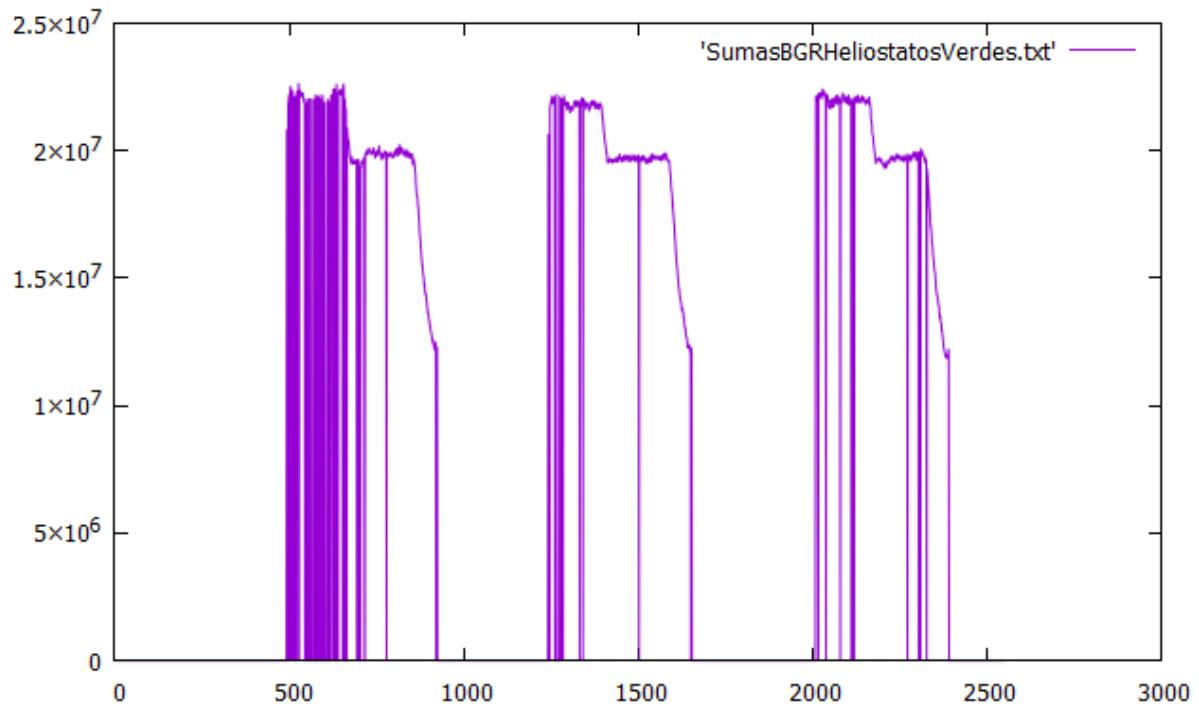


Figura 4.62: Estimación de potencia de los helióstatos verdes en las distintas partes del vídeo 'heliostato.MOV'.

y por tanto, su valor de área (o resultado de la gráfica) es grande. Después, cuando el helióstato alcanza el centro del vídeo y se sitúa ahí unos instantes, su valor de área se reduce ligeramente. Finalmente, cuando el helióstato se va poco a poco hacia la izquierda para salirse del vídeo, y a partir del momento en el cual el helióstato toca el extremo izquierdo de dicho vídeo, su valor de suma BGR se reduce poco a poco, hasta llegar a cero (el helióstato se fue totalmente del vídeo). Este procedimiento sigue igual para los siguientes dos helióstatos.

## Capítulo 5

# Conclusiones y posibles líneas de trabajo futuro

El poder tener una estimación (lo más realista posible) de la potencia reflejada, así como identificar su contorno y centroide de un helióstato en tiempo real permitirá realizar ciertas tareas de control y operación en plantas termosolares de forma sistemática desde un computador. Por ejemplo: recalibración automática de desviaciones (offsets) de cada helióstato de forma periódica; detección de daños en la estructura si la estimación de potencia cambia de forma brusca (pérdida de facetas/espejos); estimación de la potencia contribuida por cada helióstato de forma que se pueda controlar con mayor precisión la potencia por el grupo de helióstatos en el que está operando en cada momento. La contribución más importante es que estas tareas las asumiría el computador de forma automática durante la operación de la planta, aunque con la debida supervisión del operador de planta.

Por otro lado, el cálculo de la "variable que estima la potencia" se realiza tiempo real con las herramientas actuales de prototipado del algoritmo (como Python), por lo que se puede decir que el algoritmo estaría preparado para ser aplicado en una planta termosolar, en este momento, para realizar los ensayos de calibración orientados a obtener la correlación.

Finalmente, dar agradecimientos al centro PSA-CIEMAT por haber puesto a disposición la imágenes utilizadas en este TFG, y al centro mixto CIESOL por los espacios utilizados en las frecuentes reuniones con los tutores en la UAL.

# Bibliografía

- [1] Valenticampderros A. PS10 y PS20 [Mensaje en un blog], 2012.
- [2] BeJob. 7 razones para programar en Python, 2016.
- [3] Claudio. Centrales solares: generar energía eléctrica con el Sol, 2014.
- [4] EcuRed. Biblioteca Pública Municipal Sandino.
- [5] Google. RE<C: Heliostat Project Overview.
- [6] Gstriatum. Energías renovables [Mensaje en un blog].
- [7] Igualada.institucio.org. n. d.
- [8] Bartolomé Sintes Marco. Historia de Python, 2018.
- [9] Oepm. Helióstato con sensor de reflexión, 2011.
- [10] OpenCV. OpenCV, 2018.
- [11] C Pons. Gemasolar [Mensaje en un blog], 2012.
- [12] PSA. Instalaciones de la PSA.
- [13] PSA. Instalaciones de la PSA.
- [14] PSA. La instalación CESA-1 de 7MWt.
- [15] PSA. La instalación SSPS-CRS de 2,5 MWt.
- [16] Lucía Rincón. Funcionamiento de la central solar, 2013.
- [17] Jonathan Préstamo Rodríguez. PS10: La megatorre sevillana de la energía solar (y cómo funciona), 2014.
- [18] SAL. Infraestructura, 2015.
- [19] Solicitudma. Energía termoeléctrica.
- [20] SourceForce.net. n. d., 2018.
- [21] Ubbaa.net. OpenCV - Processing and Java Library.
- [22] Wikipedia. File: "PS10 solar power tower.jpg", 2018.
- [23] Wikipedia. Instalación de energía solar Ivanpah, 2018.
- [24] IEEE Xplore. IEEE Xplore, 2018.

[25] Yahoo. OpenCV en Español.

