Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

Grado en Ingeniería Mecánica

Desarrollo de un programa en Python para seguimiento de imágenes de llamas esféricas

Autor:

López García, Beatriz Isabel

Tutores:

Reyes Serrano, Miriam
Ingeniería Energética y Fluidomecánica

Sastre Zamora, Rosaura
Ingeniería Energética y Fluidomecánica

Valladolid, junio de 2024.

RESUMEN

La mejora en la detección de imágenes es importante para la investigación en el campo de la combustión, permitiendo detectar y calcular, por medio de imágenes de alta velocidad, parámetros que caracterizan el comportamiento de la combustión en cada instante. Este proyecto surge de la necesidad de mejorar un programa ya existente, ejecutado en Matlab, que analizaba las llamas esféricas desarrolladas en una combustión para obtener datos de interés para el análisis del proceso, como son el radio del frente de llama o la densidad celular.

Con este fin, se ha decidido desarrollar un programa utilizando el lenguaje de programación Python para resolver las carencias del programa de Matlab, empleando la librería OpenCV, que permite detectar los contornos y clasificarlos para obtener los parámetros que caracterizan el proceso.

Finalmente, se han llevado a cabo estudios comparativos entre ambos para comprobar las mejoras que ha supuesto realizar el proyecto.

Palabras clave: Python, radio frente de llama, densidad celular, detección de contornos, fotograma.

ABSTRACT

Improving image detection is essential for combustion research, enabling the detection and calculation of parameters through videos that characterize combustion behavior at each moment. This project arises from the need to enhance an existing program, run in Matlab, which analyzed combustions to obtain data of interest in the analysis of combustion processes such as flame front radius and cellular density.

A program was developed using the Python programming language to address the shortcomings of the Matlab program, employing the OpenCV library to detect image contours and classify them to obtain parameters of the flame front radius and cellular density.

Finally, comparative studies were conducted between both programs to verify the improvements achieved by the project.

Keywords: Python, flame front radius, cellular density, contour detection, frame.

ÍNDICE

1. INTRODUCCIÓN

El análisis de video ha sufrido mejoras en los últimos años que incorporar nuevas técnicas diseñadas y adaptaciones en el procesamiento permitiendo implementar avances en la forma de estudiar, comprender y calcular parámetros que suceden dentro de los procesos de combustión. El análisis de videos es una herramienta fundamental para el campo de la ingeniería y las ciencias aplicadas siendo un refuerzo que permite comprobar resultados obtenidos de manera experimental con parámetros obtenidos por medio del procesamiento de imágenes.

El avance tecnológico en el procesamiento de imágenes y la disponibilidad de emplear hardware de captura y procesamiento de video en alta resolución han permitido abrir nuevas posibilidades para el análisis detallado de muchos procesos entra los que se encuentran los procesos de combustión. Existen numerosos lenguajes de programación que incorporan herramientas de análisis y procesamiento de video permitiendo utilizar nuevas las funciones para resolver problemas cada vez más complejos debido a los avances de la sociedad.

Los lenguajes de programación permiten calcular parámetros y variables que facilitan el análisis y estudio posterior. Además, la exactitud de este tipo de herramientas se debe a las mejoras en resolución de la imagen, al aumento de la calidad de la imagen, analizando los elementos más finos que se puedan encontrar en la imagen, así como la capacidad de la que disponen este tipo de programas para mejorar la velocidad y eficiencia de los procesos gráficos procesando grandes volúmenes de datos y realizando cálculos complejos de manera rápida. También, reduce los errores humanos siendo esto último una aplicación muy importante a nivel industrial y de investigación donde la consistencia y precisión de los datos es crucial.

Estas herramientas permiten diseñar programas donde se implementen funciones y algoritmos orientados a los objetivos principales para los que están pensada la aplicación, es decir, permiten su personalización en función de las necesidades del análisis, las acciones que logren obtener los parámetros y variables necesarias que den lugar a los resultados buscados por el procesamiento de imágenes.

La aplicación de estas técnicas para analizar los procesos de combustión es una tarea compleja que requiere un enfoque sistemático y uso de librerías específicas dentro del lenguaje de programación que permita obtener de manera automática y precisa algunos parámetros críticos relacionados con la combustión. La obtención de estos parámetros permite compararlos posteriormente con el análisis realizado a partir de los resultados obtenidos de manera experimental. Estos procesos permiten caracterizar y entender mejor los procesos de combustión, independientemente del tipo de combustible que su utilice para realizar la combustión en el interior de la instalación.

El enfoque obtenido utilizando el análisis del procesamiento de imágenes tiene potencial de mejorar significativamente la investigación y desarrollo de los procesos de combustión y permite analizar de manera visual lo que ocurre dentro del cámara de combustión caracterizando cómo es el proceso completo.

1.1. Motivación y antecedentes

Este Trabajo de Fin de grado surge de la investigación que se realiza en el GIR MYER (Motores térmicos y Energías Renovables) del departamento de Energética y Fluidomecánica de la Universidad de Valladolid, cuyo actual objetivo es el estudio de procesos de combustión de combustibles alternativos. En el contexto de dicha investigación, se detectó la necesidad de mejorar un programa de procesamiento de imágenes programado en el lenguaje de Matlab desarrollado por M. Rodríguez Gonzalez [1], que realiza el estudio visual de las combustiones que se desarrollan en el interior de la bomba cilíndrica a volumen constante que posee el Grupo.

Atendiendo a la necesidad manifestada, se desarrolló un nuevo programa utilizando el lenguaje de programación de Python que detectara y analizara parámetros importantes que caracterizan una combustión esférica, que son la velocidad del frente de llama y la celularidad del frente de llama.

1.2. Objetivos

El objetivo principal perseguido en el presente Trabajo de Fin de Grado es la creación de un nuevo programa de tratamiento de imágenes utilizando el lenguaje de programación de Python que analice los videos de procesos de combustión que tienen lugar en el interior de una bomba cilíndrica a volumen constante y permita obtener parámetros que caractericen la combustión como son la velocidad y la celularidad del frente de llama.

A continuación, se exponen los objetivos secundarios que han servido de guía para alcanzar el objetivo principal del proyecto:

- Aprender en mayor profundidad a programar en el lenguaje de programación de Pythony aprender a implementar y utilizar los algoritmos que están incorporados en la librería OpenCV para la detección de contornos de la imagen.
- Comparar los resultados obtenidos con el nuevo programa con los resultados obtenidos para el mismo experimento, pero utilizando el programa de Matlab.
- Tratamiento de fotograma para obtener en mayor resolución los parámetros que caractericen el comportamiento del frente de llama utilizando gráficas y tablas que visualicen por pantalla el comportamiento de la combustión en cada fotograma.

- Descargar los datos calculados por el programa para cada fotograma en un formato de archivo de tratamiento de datos.

1.3. Estructura del trabajo

Este Trabajo de Fin de Grado se ha estructurado en 6 capítulos de la siguiente manera:

1. Introducción: se presenta el campo al que está orientado el proyecto, los objetivos que se buscan con el proyecto y los contenidos que se van a desarrollar en él.
2. Estado del arte: explicación de los diferentes conceptos que se van a utilizar dentro del proyecto, así como qué tipo de procesos se van a analizar. Además, se desarrollan los problemas principales que han llevado a la realización de este proyecto y el tipo de lenguaje que se ha utilizado para desarrollarlo.
3. Desarrollo de la aplicación: se expone y se explica la forma en la que se ha desarrollado la aplicación y el porqué del uso de ciertas funciones.
4. Procedimientos utilizados para el análisis del fotograma: se desarrollan las distintas opciones que habían barajado para el desarrollo de la aplicación y por qué no se han seleccionado.
5. Resultados: se presentan distintos estudios realizados para comprobar que el programa funciona bajo las prestaciones expuestas desde el comienzo.
6. Conclusiones

2. ESTADO DEL ARTE

2.1. Definición de combustión

La combustión es uno de los procesos más utilizados para generar energía teniendo como fuente de alimentación combustibles que pueden estar en cualquiera de sus tres estados más comunes, sólido, líquido o gas. Dependiendo del tipo de aplicación y sector al que va destinado el proceso, el tipo de combustible que se usa puede variar, así como la cantidad de energía que se genera durante el proceso. Las aplicaciones del proceso de combustión pueden estar orientadas hacia el sector industrial, siendo una de sus aplicaciones los quemadores industriales; hacia el sector del transporte y movilidad sostenible, como es su aplicación en los motores de combustión interna alternativos (MCIA); hacia el sector de la investigación y enseñanza, realizando ensayos con condiciones y con parámetros controlados que permitan mejorar los futuros procesos; o hacia el sector doméstico, usado principalmente en las calderas. Para que todas estas aplicaciones se puedan llevar a cabo, es necesario realizar un estudio detallado de cada una de las etapas de la combustión y llevar un control exhaustivo de todas las condiciones y parámetros que intervienen en ellas.

Para que se pueda producir la reacción de oxidación conocida como combustión, se necesitan dos componentes que son el combustible, que es el elemento que se oxida, y el comburente, el oxígeno que proviene de la atmósfera que actúa como agente oxidante, para producir una reacción química exotérmica que libera grandes cantidades de energía térmica. Para que este proceso se pueda llevar a cabo, se necesitan una serie de condiciones de presión, temperatura y composición determinadas para que la reacción sea estable. El proceso comienza tras aportar la energía mínima para que la reacción se pueda desarrollar posteriormente de manera espontánea, este proceso se denomina ignición que se puede llevar a cabo de varias formas. Entre los objetivos que tiene el empleo de las reacciones de combustión está la de producir energía en forma de calor o trabajo. En la combustión, no sólo están involucrados los fenómenos químicos, sino que también tienen mucha relevancia los fenómenos físicos [2].

Para poder comprender mejor el presente proyecto y lo que ocurre en los procesos de combustión, es necesario definir una serie de conceptos técnicos que afectan a la combustión, para poder comprender la mejora realizada con dicho proyecto y los resultados finales obtenidos.

2.2. Conceptos

Para comprender mejorar los procesos que se van a calcular en este proyecto, es necesario tener una serie de conceptos que ayudará comprender el objetivo que se busca en el trabajo.

2.2.1. Tipos de MCIA

Dependiendo de la forma en la que se encuentren los reactivos en el interior de la cámara de combustión y la forma en la que se desencadene la reacción de combustión, se pueden clasificar la combustión de la siguiente forma:

- *Motor de encendido provocado* o *MEP*: El combustible y el comburente se mezclan antes de que se produzca la combustión, permitiendo obtener una mezcla homogénea. La combustión se inicia por el salto de una chispa eléctrica. El proceso de combustión finaliza cuando el frente de llama llega a las paredes de la cámara de combustión. La proporción que entra de combustible y comburente debe de garantizar el proceso de combustión sea estable. Para variar la cantidad la cantidad de combustible se regula por medio de la proporción de comburente con que se mezcla[3].
- *Motor de encendido por compresión* o *MEC:* La combustión se produce por intercambio de partículas en las proximidades a la reacción, siendo solo el comburente lo que se encuentra dentro de la cámara de combustión. Posteriormente se inyecta el combustible a alta temperatura y alta presión provocando que el proceso de autoinflame y comience la combustión. La cantidad de combustible que se quema se controla durante el periodo de tiempo que se está inyectando el combustible, por el contrario, no se puede controlar la cantidad de comburente [3].

2.2.2. Dosado

El dosado, F, es el parámetro que caracteriza la mezcla de aire-combustible. Este parámetro relaciona el gasto másico o la masa de combustible con el gasto másico o la masa de aire, el comburente [3]. Este parámetro es adimensional. La ecuación que los relaciona es la siguiente:

$$F = \frac{Masa\ Combustible}{Masa\ de\ aire} = \frac{\dot{m}_f}{\dot{m}_a} = \frac{m_{fcc}}{m_{acc}} \qquad (Ec.\ 1)$$

Definiendo:

- $\dot{m}_f$ hace referencia al flujo másico de combustible, se mide en kg/s.
- $\dot{m}_a$ hace referencia al flujo másico de aire, se mide en kg/s.
- m_{fcc} hace referencia a la masa de combustible por ciclo y por cilindro, se mide en kg.
- m_{acc} hace referencia a la masa de aire por ciclo y por cilindro, se mide en kg.

El dosado estequiométrico, Fe, es el dosado que tiene que haber en una mezcla aire-combustible, para que en la reacción de combustión no sobre aire ni sobre combustible. Es muy importante realizar un buen ajuste de la reacción ya que el dosado estequiométrico es una propiedad del combustible y no depende de otras condiciones externas a la reacción [3]

$$C_nH_mO_p + X_{est}(O_2 + 3.76\,N_2) \rightarrow nCO_2 + \frac{m}{2}H_2O + \left(n + \frac{m}{4} - \frac{p}{2}\right) 3.76\,N_2 \quad \textit{(Ec. 2)}$$

$$X_{est} = n + \frac{m}{4} - \frac{p}{2} \quad \textit{(Ec. 3)}$$

$$F_e = \frac{12\,n+m+16p}{X_{est}\,(32+3.76*28)} \quad \textit{(Ec. 4)}$$

Por último, se define el concepto de dosado relativo, Fr, o también llamado riqueza. Este concepto relaciona el dosado con el dosado estequiométrico indicando si tenemos o no exceso de combustible o de aire [3].

$$F_r = \frac{F}{F_e} = \frac{m_f}{m_{fe}} \quad \textit{(Ec. 5)}$$

Si

- Fr > 1, se tiene un exceso de combustible y se considera una mezcla rica.
- Fr = 1, se tiene un equilibrio estequiométrico.
- Fr < 1, se tiene un defecto de combustible y se considera una mezcla pobre.

2.2.3. Velocidad de combustión laminar y la velocidad del frente de llama

La velocidad de combustión es la velocidad del frente de llama con respecto a los reactivos o gases sin quemar. Está relacionada con la transformación de los productos inquemados (mezcla fresca) en productos quemados, siendo una propiedad del combustible que domina el proceso de combustión en los Motores de Combustión Interna Alternativos (MCIA) [4].

$$u_n = \frac{\dot{m}_q}{\rho_{sq*}\,A_f} \quad \textit{(Ec. 6)}$$

Donde $\dot{m}_q$ es el gasto de quemados que sale de la zona donde se produce la reacción, ρ_{sq} es la densidad de la mezcla fresca y A_f es el área del frente de llama de estudio.

Para el caso particular de estudio se emplea la hipótesis de un frente de llama esférico, el cual encierra en su interior los productos de la combustión o quemados y deja en el exterior los reactivos o gases sin quemar. El frente de llama se correspondería con la superficie de la esfera. Teniendo en cuenta esta hipótesis, se define la velocidad de avance del frente de llama, S_L^K, como la derivada de su radio respecto del tiempo [5].

$$S_L^K = \frac{dr}{dt} \quad \textit{(Ec. 7)}$$

Cuando la combustión se desarrolla en el seno de un fluido en reposo donde no existen fenómenos de turbulencia externos al sistema, su velocidad de combustión se dice que es laminar. Este proceso depende de que la mezcla aire-combustible sea homogénea. La velocidad de combustión laminar es una propiedad de la propagación de la mezcla aire-combustible que depende de la presión, de la temperatura, de los componentes inertes que no intervienen en la reacción y de la relación que existe entre el combustible y el aire (dosado).

La velocidad de combustión laminar aumenta cuando lo hace la temperatura media del proceso que se ve elevada por el aumento de la temperatura de la mezcla fresca. Para relaciones de aire-combustible cercanas a lo estequiométrico, la velocidad de combustión laminar es máxima [6].

2.2.4. Inestabilidades y celularidad

El frente de llama en una combustión puede verse afectado por ciertos efectos que provocan que puedan tener carácter de estabilización o desestabilización del frente. Si estos efectos tienen un carácter estabilizador, se generaría un frente de llama liso. Por el contrario, si tienen un carácter desestabilizador, pueden provocar perturbaciones en el frente de llama llamadas inestabilidades, dando lugar a un frente de llama rugoso.

Las inestabilidades se pueden clasificar dependiendo de su origen y del efecto que generan sobre el frente de llama.

- Por fuerzas de volumen: estas inestabilidades se producen por una diferencia de densidades que actúan en contra de las fuerzas de volumen presentes en la llama (principalmente debido a la gravedad). La diferencia de densidades se produce entre los productos, de menor densidad, y los reactivos con mayor densidad. Esta diferencia de densidades provoca que el frente de llama tenga una velocidad de avance no uniforme y de lugar al arrugamiento del frente [5].

- Por efecto hidrodinámico: estas inestabilidades también aparecen por diferencia de densidades entre los reactivos y los productos en el frente de llama porque los productos sufren un proceso de expansión debido al aumento de temperatura durante el proceso. Cuando aparece la discontinuidad, generado por diversos factores, en el frente de llama comienzan a aparecer arrugamientos u ondulaciones que rompen con la estabilidad del frente y comienzan a crecer de manera rápida. Este crecimiento provoca una diferencia de velocidades entre los quemados y los sin quemar, siendo más rápida la velocidad de los quemados y generando ondulaciones hacia el interior del frente de llama. Estos arrugamientos a su vez generan que la velocidad de los quemados también aumente, dando lugar

a que la reacción se desarrolle de manera más rápida y el frente de llama sea más inestable [5].

- Por efecto termo-difuso: a diferencia de los dos efectos desarrollados anteriormente, las inestabilidades causadas por este efecto pueden estabilizar o desestabilizar el frente de llama. Para comprender mejor la influencia de este efecto, es necesario conocer que en el frente de llama es el lugar donde se produce la transformación de los reactivos en productos, este proceso de transformación tiene que ver con transporte de masa (difusividad másica) y con el transporte de calor (difusividad térmica). La relación entre ambas difusividades es de las causas de este efecto y, dependiendo de qué efecto domine el proceso, si el transporte de masa o de calor, el efecto termo-difusivo tenderá estabilizar o desestabilizar el frente de llama. A medida que aumenta el frente de llama esférico, la influencia de este efecto disminuye con respecto a la de otros, como puede ser el hidrodinámico.

  Si el efecto que prevalece es el transporte de calor, el frente de llama tiende a estar más liso produciendo un efecto estabilizador tanto en la llama como en el frente. Si, por el contrario, el efecto dominante es el transporte de masa, se produce una transformación de reactivos en productos a mayor velocidad, incrementando la velocidad del frente de llama que a su vez genera que disminuya la temperatura de los reactivos y dé lugar a la aparición de arrugas en el frente de llama haciendo que este se vuelva inestable [5].

Las inestabilidades en llamas esféricas se pueden dar debido a los efectos mencionados anteriormente, pudiendo llegar a provocar un frente de llama rugoso si la suma de todos los efectos es de carácter desestabilizador

La celularidad es el proceso por el cual la morfología del frente de llama se modifica en una combustión laminar debido a las inestabilidades que aparecen en él. Tras la aparición de la celularidad en la superficie del frente de llama, la velocidad del frente de llama aumenta debido a que la superficie que separa la mezcla fresca de los quemados se ve aumentada, y, por tanto, aumenta el gasto másico de quemados debido a este aumento de la superficie de la llama . La aparición de la celularidad depende de si la suma de efectos es desestabilizante y del tiempo de desarrollo de las inestabilidades, Las inestabilidades afectan a la combustión durante todo su desarrollo, pero se hacen notorias cuando consiguen romper visiblemente la estructura del frente de llama, generando pequeñas celdas que hacen más inestable todo el proceso y modifican su velocidad de combustión [5].

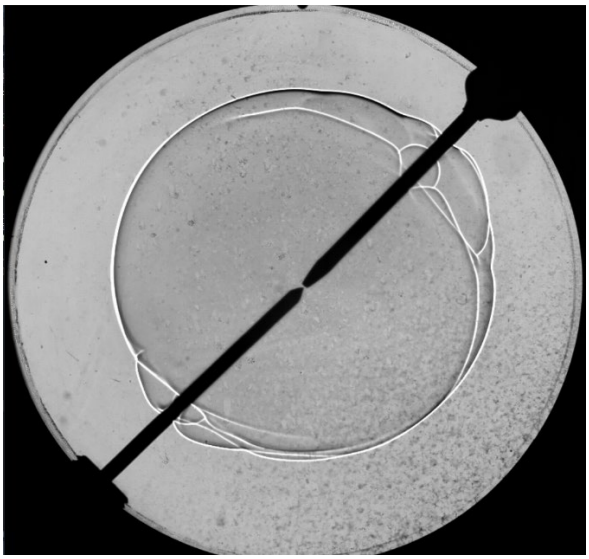


Figura 2.1: Frente de llama liso para una combustión de etanol Fr = 1,4.

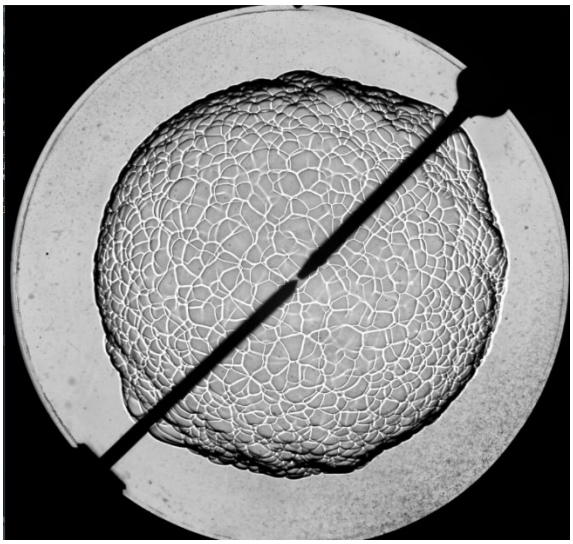


Figura 2.2: Frente de llama celular para una combustión de etanol Fr = 1,01.

En procesos de combustión con frentes de llama esféricos cuando las inestabilidades se hacen perceptibles rompiendo el frente de llama, este pasa de ser liso a que aparezcan rugosidades que aumentan de manera progresiva, pasando a un estado que se denomina celular. La inestabilidad aparece cuando la perturbación crece a un ritmo más rápido que el ritmo de crecimiento de llama [6].

Se presentan dos tipos de celularidad distintos que se clasifican dependiendo de cuando las inestabilidades rompen el frente de llama. En general, se podría decir que:

- Si la celularidad aparece al comienzo de la combustión cuando los radios son pequeños y esta evoluciona de manera progresiva durante toda la combustión, se dice que la combustión se ve influenciada por fenómenos termo-difusivos. Este efecto es el único que puede tener un carácter estabilizador. Si se da el caso de que este también es desestabilizante, las inestabilidades no se verán amortiguadas por ningún factor y crecerán desde el inicio de la combustión. El frente de llama acaba siendo muy celular como el de la *Figura 2.2*.
- Si la celularidad aparece al final de la combustión cuando los radios son mucho más grandes, es decir, que el frente de llama tarda más en hacerse inestable, se dice que la combustión se ve influenciada por fenómenos hidrodinámicos. Esto se da cuando el efecto termo-difusivo tiene un carácter estabilizador y es capaz de amortiguar el crecimiento de las inestabilidades en las primeras etapas de la combustión. Sin embargo, a medida que aumenta el radio el efecto hidrodinámico se verá potenciado y podría llegar a generar celularidad tipo, por lo general con celdas más homogéneas. El frente de llama es laminar durante casi toda la combustión como en la *Figura 2.1*.

2.3. Instalación experimental y tratamiento de imágenes

La instalación utilizada para obtener los datos y los videos de forma experimental se ubica en el Departamento de Ingeniería Energética y Fluidomecánica, en concreto en el Laboratorio de Motores, ubicado en las instalaciones de la Escuela de Ingenierías Industriales sito en la Sede del Paseo del Cauce, en la Universidad de Valladolid.

La instalación está compuesta por un prisma ortogonal de acero con un vaciado cilíndrico en el interior que actúa de bomba cerrada a volumen constante en cuyo interior se desarrolla el proceso de combustión. Para que se puedan alcanzar las condiciones adecuadas en el interior de la bomba esta debe de estar cerrada por ambos laterales por medio de unos cristales de silicio fundido transparentes capaces de soportar las altas presiones y las altas temperaturas que suceden durante el proceso de combustión. Además, estos cristales son transparentes para poder grabar todo lo que sucede en el interior de dicha cámara de combustión [5].

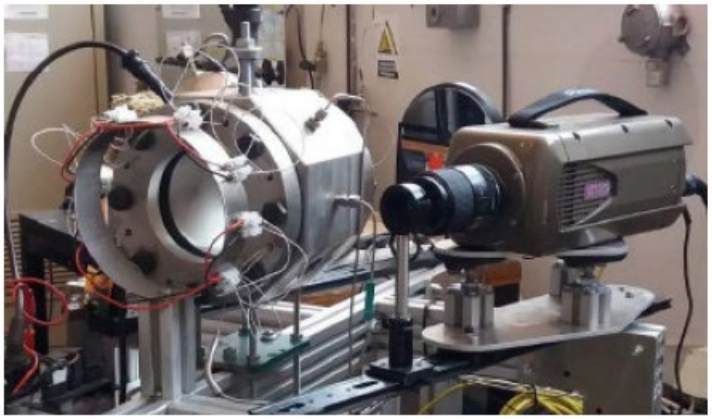


Figura 2.3: Instalación de la bomba cilíndrica a volumen constante y la cámara de alta velocidad [5].

Para poder introducir el combustible líquido en el interior de la cámara de combustión, se encuentra un pequeño orifico en la parte superior de dicha bomba. El combustible se introduce a través de una jeringuilla por una válvula de admisión que impide que aire y el combustible se escapen del interior, esto hace que todo el conjunto sea estanco. Además, conecta la bomba con una botella a la presión del combustible.

El aire, elemento que actúa como comburente, se introduce en el interior de la bomba de combustión por medio de un circuito neumático cerrado y regulado por válvulas que abren y cierran los distintos accesos, tanto de acceso del aire al interior de la cámara, como de evacuación de los productos hacia el exterior de la misma, ambos circuitos son independientes.

El tipo de combustión que se genera en el interior de la bomba es premezclado de encendido provocado ya que es necesario que el oxígeno y el combustible se

encuentren mezclados de manera homogénea. Para que la combustión se pueda producir, son necesarios dos electrodos conectados a una fuente de electricidad, es decir, que para que se pueda producir la ignición de la reacción es necesario un encendido provocado de la misma. Los electrodos son de grafito y se encuentras colocados en el interior de la cámara de combustión enfrentados entre ellos para que cuando circule la electricidad, se genere un arco eléctrico que inicie el proceso de combustión. Los electrodos están colocados de manera estratégica para que la combustión se inicie lo más aproximadamente posible al centro geométrico de la cámara de combustión, ver *Figura 2.3*.

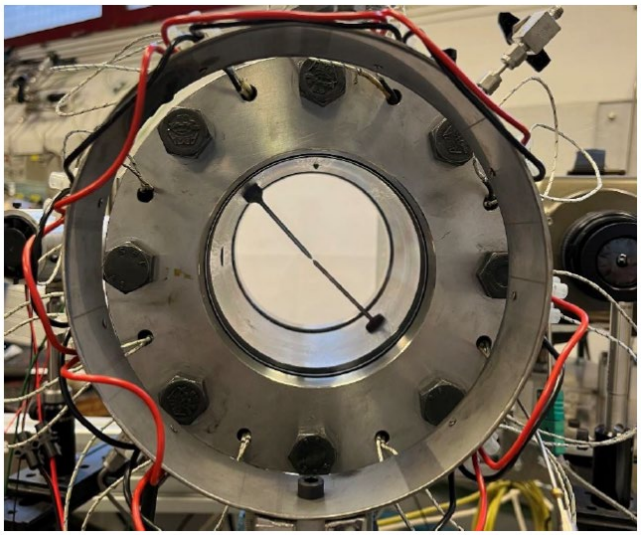


Figura 2.4: Electrodos del interior de la bomba cilíndrica a volumen constante y válvula de admisión del gas en la esquina superior derecha. [7]

Gracias a todos los elementos que componen la instalación de la bomba, se pueden generar llamas esféricas que se expanden desde el centro de la cámara de combustión hasta las paredes de la misma y se puede grabar todo el proceso, por medio de una cámara especializada, para su posterior estudio y análisis.

2.3.1 Técnica Schlieren

La técnica Schlieren se basa en la comprensión e implementación de la fotografía shadowgraph. La fotografía shadowgraph consiste en hacer que atraviese un haz de luz por el campo de estudio haciendo que se proyecten las sombras resultantes permitiendo captarlas posteriormente en un video. La sombra captada, y posteriormente analizada, se genera debido a la desviación del haz de luz al encontrarse con una diferencia de densidades en el medio que atraviesan. Existe una diferencia de densidades entre los productos y los reactivos que da lugar a la desviación de los rayos y se registra en la cámara de alta velocidad.

Gracias a las mejoras ópticas, se pueden analizar con gran nitidez la morfología, percibir la velocidad del frente de llama, el crecimiento de la llama y la aparición de

las inestabilidades en un instante de tiempo determinado. Para poder estudiar la velocidad del frente de llama, es necesario relacionar la evolución del radio del frente de llama, captado con gran nitidez en video, con el tiempo en el transcurre el video al completo. Además de estudiar la evolución del radio del frente de llama, también nos permite analizar los arrugamientos que aparecen en el frente de llama, es decir, estas técnicas permiten analizar las inestabilidades y la celularidad con gran nivel de detalle [5].

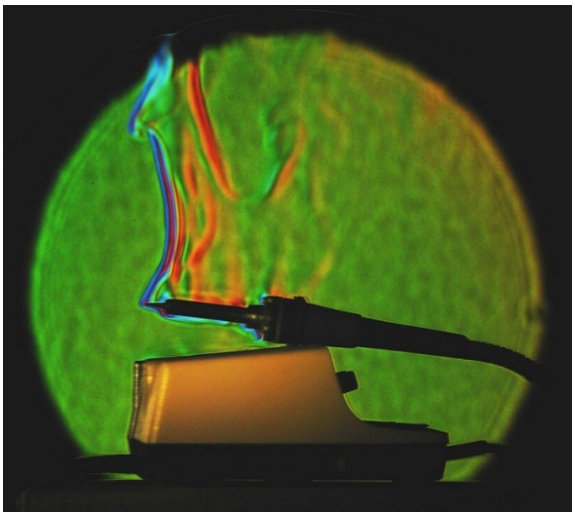


Figura 2.5: Técnica Schlieren [8].

Para poder implementar esta técnica en la instalación donde se van a realizar la parte experimental, es necesario un equipamiento específico y una disposición adecuada y determinada del mismo. La configuración más típica de esta técnica consiste en colocar todo el equipo asemejándose a una Z, por lo que esta disposición se denomina tipo Z, ver *Figura 2.5*. El procedimiento de utilización de esta disposición consiste en proyectar un haz de luz en dirección a un espejo cóncavo que se encarga de proyectar la luz que le llega hacia el interior de la cámara de combustión, atravesándola, durante el desarrollo de la combustión, hasta llegar a otro espejo cóncavo que refleja, por última vez, hacia un elemento colocado entre este último espejo y la cámara que graba todo el proceso. El elemento intermedio se denomina filo y su función consiste en eliminar la radiación de fondo antes de que quede registrado en la cámara. Tras pasar el filo, el haz de luz llega hasta la cámara CCD (dispositivo de carga acoplada) de alta velocidad. Este dispositivo es una cámara Phantom V210 que entre sus especificaciones destaca que tiene una velocidad de captura máxima de 3100 fps (fotogramas por segundo) y una resolución de 2000 fps [1]. Las imágenes captadas con esta cámara, pasan posteriormente por un programa de análisis de imágenes que permite recopilar la información de los videos proporcionados.

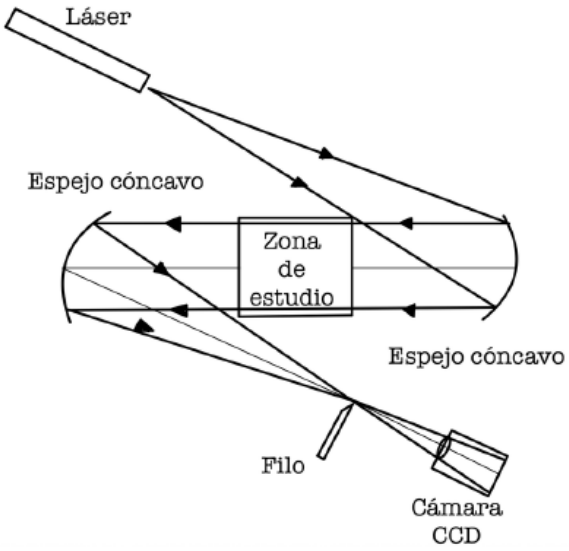


Figura 2.6: Esquema en "Z" de un equipo Schlieren [5]

2.4. Programa de Matlab

Este Trabajo de Fin de Grado se basa en un programa creado por M. Rodríguez González [1] que, aparte de realizar mejoras en la disposición de los electrodos del interior de la bomba cilíndrica a volumen constante antes descrita, creó un programa en el lenguaje de programación de Matlab que analizaba los videos de combustiones que se desarrollan en el interior de la bomba.

El programa de Matlab tiene el mismo objetivo que el que se busca en este proyecto: analizar videos de combustión para obtener de ellos los parámetros que permitan caracterizar la velocidad del frente de llama y la celularidad que aparece en el frente de llama durante la combustión. Para que el programa de Matlab funcione de manera correcta se utilizó el método Ransac que consiste en un método iterativo que permite definir las circunferencias correspondientes al radio del frente de llama y la cámara de combustión. Para realizar la interacción, el programa selecciona 3 puntos al azar de fotograma, dentro de una serie de tolerancias, y genera una circunferencia que pasa por esos 3 puntos; estos puntos a su vez definen el rango y se vuelven a seleccionar 3 puntos próximos a estos y generar la circunferencia, y así sucesivamente hasta lograr que se ajuste lo máximo posible a radio del frente de llama que se detecta en la imagen. Para calcular los radios de los fotogramas siguientes, se realizar una resta de fotogramas en base a los datos obtenidos del fotograma anterior y la nueva zona donde va a detectar los puntos para el cálculo de la circunferencia, dando como resultado una diferencia de pixeles que coincide con la diferencia entre los dos fotogramas.

El principal problema, y principal motivación para realizar una mejora del programa en un nuevo lenguaje de programación, surge en la forma de generar las circunferencias, ya que al elegir 3 puntos al azar y después realizar iteraciones consecutivas, el programa puede seleccionar esos 3 puntos sin coincidir con el frente

de llama buscado y por tanto el radio que va a obtener puede estar sobredimensionado o subestimado respecto al radio del frente de llama deseado. Además, al realizar la resta de fotogramas, el sobredimensionamiento o subestimación detectadas influyen obteniendo valores del frente que no se ajustan al valor real del radio para cada instante de tiempo. Los datos de celularidad o del radio del frente de llama que se calculan usando este programa no resultan en ocasiones tan certeros como para poder compararlos con los resultados experimentales.

Con el nuevo programa desarrollado se busca resolver los problemas que surgen con el programa de Matlab aplicando el método de detección de contornos para cada fotograma individualmente en un nuevo lenguaje de programación.

Es importante disponer de un modelo de análisis bueno ya que los datos obtenidos por medio de cámaras de alta velocidad permiten no solo obtener datos referentes al radio del frente de llama y a la celularidad, sino información crucial con la que comparar los datos obtenidos con los elementos experimentales y poder ratificar o no que los resultados son fiables para el estudio. Si el modelo de análisis de imágenes no es adecuado, la comparativa no será adecuada y no se podrán utilizar de manera óptima los resultados para obtener conclusiones.

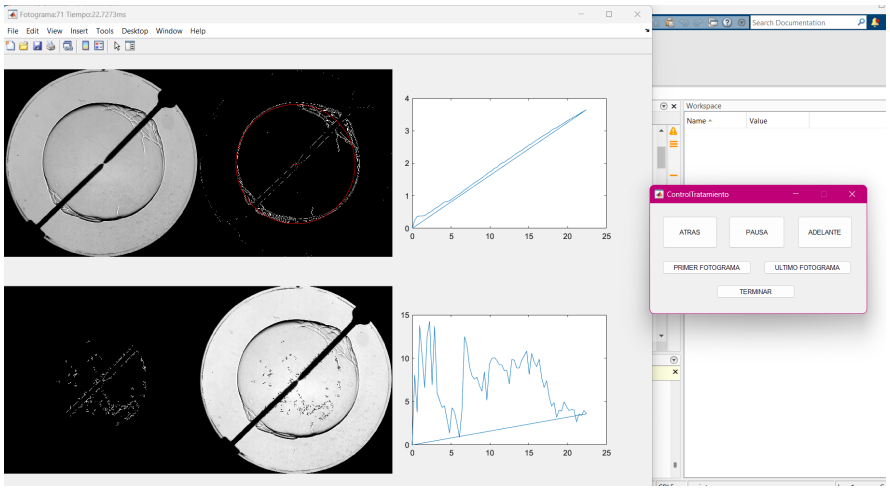


Figura 2.7: Interfaz visual del programa de Matlab.

2.5. Lenguaje de programación Python y la librería Open CV

Python es un lenguaje de programación de código abierto de alto nivel aplicable de forma binaria o de forma de código fuente. Este tipo de lenguaje está orientado a objetos aplicables, programación imperativa y programación funcional destinado principalmente al desarrollo de páginas web y realización de aplicaciones.

El creador de este lenguaje de programación es Guido Van Rassum, un programador holandés. Se graduó en Matemáticas y Ciencias de la Computación por la Universidad de Amsterdam en 1982. Desarrolló el lenguaje de programación entre

los años 1989 a 1991, cuando publicó la primera versión del lenguaje en el Instituto Nacional de Investigación en Matemáticas y Ciencias de la Computación. Tras esta primera publicación, se han realizado PEPs o Propuestas de Mejora de Python gracias a las cuales se han descrito cambios en el código y se han incluido nuevas actualizaciones [9].

La sintaxis utilizada es limpia, legible y fácil de aplicar y comprender teniendo un amplio parecido al lenguaje humano. Esto hace que, para las posibles modificaciones que se den de los programas que han sido realizados haciendo uso de este lenguaje de programación, sea más sencilla su modificación y realizar mejoras en los apartados o puntos que sean necesarios. Este sistema está soportado por módulos y paquetes que resultan al conformar programas diseñados para diversas modalidades y muy distintos entre ellos. Además, dispone de múltiples librerías orientadas a realizar diferentes acciones ya que están formadas por numerosos comandos y funciones que permiten realizar diversos trabajos dando lugar a las aplicaciones. Una de las librerías más utilizadas en este lenguaje es la librería numpy, detallada más adelante en el apartado en él se hace una explicación más detallada del desarrollo del programa.

Python es un lenguaje de programación que se emplea para diseñar cualquier tipo de programa o aplicación por simple o por complejo que sea. Se trata de un lenguaje interpretado, esto quiere decir, que emplea un formato de comprensión de cara al usuario sencillo y sobre todo legible que posteriormente el ordenador traduce en tiempos de ejecución cortos sin necesidad de pasar por el proceso de compilación. Para que el ordenador procese de manera adecuada el programa, debe pasar por capas intermedias entre las instrucciones introducidas por el programador y las instrucciones que el ordenador sabe interpretar, esta capa intermedia transforma las instrucciones a código binario.

Este tipo de lenguaje puede emplearse para procesar texto, interpretar número, codificar imágenes, resolver ecuaciones y problemas matemáticos o interpretar y guardar datos. En la programación de Python no es necesario declarar el tipo de variable ya que el propio intérprete se encarga asignarlo. Algunos de los datos que más se usan son el tipo entero, el tipo decimal o de punto flotante, el de tipo complejo, el de cadena de texto, los de tipo Booleano, las listas o las matrices entre otros [10].

2.5.1. Librería OpenCV

La librería OpenCv significa *Open Computer Vision* (en español Visión Artificial Abierta) por lo que se trata de una librería de computación visual para el procesamiento de imágenes que contiene algoritmos y funciones que permiten el procesamiento de imágenes. Esta librería fue desarrollada por la empresa Intel y su primera versión fue lanzada en línea en enero de 1999. El lenguaje de programación en el que fue desarrollada su primera versión C++ haciendo uno de una serie de

plantillas con contenedores STL. Posteriormente, se desarrolló en otros lenguajes de programación como Java o Python siendo este último uno de los más relevantes para el desarrollo de esta librería ya que es de código abierto y numerosas empresas desarrollan sus aplicaciones haciendo uso del mismo [11].

Al igual que el lenguaje de programación Python, esta librería es de código abierto lo que permite modificar su código fuente dependiendo de las necesidades. Se trata de una librería especializada en visión artificial y *machine learning* siendo su principal función la de proporcionar infraestructura común para aplicaciones de visión artificial a las grandes empresas como Google, Youtube, Intel o Microsoft entre otras. Cuenta con 2500 algoritmos y herramientas de aprendizaje automático que permiten realizar acciones como identificar objetos, encontrar imágenes, eliminar objetos, seguimiento de movimientos, clasificar acciones o extraer modelos 3D entre otras. Otra de sus modalidades más llamativas es la incorporación de algoritmos que permiten desarrollar aplicaciones en tiempo real y realizar el seguimiento de objetos que modifican su posición a tiempo real [12].

De las ventajas de esta librería se pueden destacar las siguientes:

- Es una librería que proporciona soporte para infraestructuras de aplicaciones de visión artificial utilizadas en nuestro día a día.
- Es un sistema multiplataforma puesto que la librería que se puede ejecutar en cualquier sistema operativo como Windows, Mac, Linux, Android o IOS.
- Se ejecutó por primera vez en el sistema de programación de C++ pero actualmente, y dadas sus múltiples aplicaciones, está implementada para otros tipos más empleados actualmente como son Java o Python entre otros.
- La versión utilizada por primera vez ha sufrido mejoras del sistema, pero la base de sus algoritmos y funciones siguen siendo las mismas desde los años 2000.
- Su desarrollo y mejoras en los algoritmos son constantes, lo que justifica la eficiencia y las múltiples aplicaciones de esta librería.
- Es una de las librerías más documentadas y actualizadas de todas las desarrolladas por las empresas ya que una de las principales preocupaciones de los desarrolladores es que todos los programadores tengan acceso a todas modalidades y explotar el máximo el potencial del que dispone.

Figura 2.8: Ejemplo de Funciones que se pueden realizar con la librería OpenCV en el lenguaje de programación de Python [13]

3. DESARROLLO DE LA APLICACIÓN

Para poder realizar el análisis de los videos de las combustiones realizadas en la parte experimental, se ha desarrollado un programa en el lenguaje de programación Python que se encarga de emplear la inteligencia artificial para obtener la información visual necesaria para la caracterización de las llamas de las combustiones realizadas de manera experimental.

En concreto, este proyecto se ha desarrollado para realizar los cálculos recogidos en la parte experimental de la tesis doctoral de R. Sastre Zamora [5], que estudia el comportamiento del combustible etanol (C_2H_6O) a distintas diferentes presiones iniciales y modificando el dosado para estudiar el comportamiento del combustible y qué tipo de inestabilidad que aparece en cada combustión, manteniendo constante siempre la temperatura inicial en todos los experimentos.

El programa que se ha generado consta de dos objetivos fundamentales. Uno de ellos busca obtener la evolución del radio del frente de llama a lo largo de la combustión y el otro busca realizar un análisis del crecimiento de la celularidad y el instante de tiempo en el que aparece transformando el frente de llama de liso a rugoso. A medida que se produce la evolución de ambos procesos, que no se realizan de manera simultánea, se van graficando los datos obtenidos mostrándolos por pantalla y se guardan en un documento Excel.

Para poder obtener de manera precisa cada uno de los datos del proceso, es necesario aplicar a cada fotograma de manera independiente los cálculos que se realizan para obtener cada uno de los datos. Esto quiero decir que cada fotograma del video pasa por todo el proceso de análisis y grafica su evolución de manera independiente a los demás fotogramas, para un instante de tiempo determinado.

A continuación, se va explicar de manera detallada cómo se ha creado el programa, las librerías que se han ido implementando, las funciones que se han utilizado de las librerías y el porqué de su implementación en cada parte, cómo se muestran por pantalla cada uno de los datos obtenidos en el instante de tiempo correcto y cómo varían los datos de un experimento a otro.

El proyecto está dividido en varias partes, en concreto en tres: la primera de todas corresponde a la importación de las librerías necesarias y apertura y análisis del video a estudiar, la segunda se corresponde con la parte que realiza el cálculo del radio de la cámara de combustión, la fase de análisis y obtención de los datos buscados y por último la parte que se encarga de guardar los datos que se obtienen en las fases previas. Es necesario saber que se han desarrollados dos programas independientes que calculan datos distintos. Uno es el que calcula la evolución del radio del frente de llama y otro de ellos es el que calcula la celularidad. Estos programas no son totalmente diferentes, ya que comparten una parte general que son las fases de las librerías, el cálculo del radio de la cámara de combustión y la fase de recogida y guardado de datos.

A lo largo de todo el programa se pueden observar que hay partes que no se ejecutan y que emplean delante de ellas el símbolo "#" esto significa que lo que aparezca seguido de este símbolo es un comentario. Los comentarios proporcionan información de los procesos y funciones que se están desarrollando en cada línea de código. Actúan de elemento aclaratorio para la persona que está leyendo el código y permiten una comprensión más rápida del mismo.

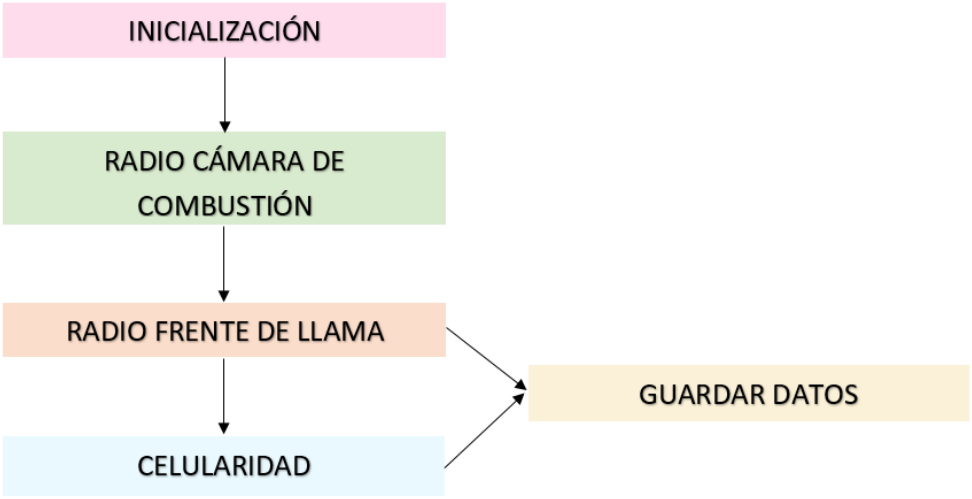


Figura 3.1: Diagrama de flujo reducido de la estructura del programa

3.1. Estructura del programa

3.1.1. Apertura de librerías

En programa creado para el análisis de los videos de las combustiones se han empleado una serie de librerías que han permitido realizar acciones sobre los videos para obtener la información necesaria. Las librerías que se han empleado son las siguientes.

```
1   #  LIBRERÍAS
2   # Apertura de las librerias que vamos a usar en el desarrollo del programa
3   import cv2
4   import numpy as np
5   import matplotlib.pyplot as plt
6   import pandas as pd
7   import os
```

Figura 3.2: Librerías empleadas en el programa de análisis de imágenes.

Cada una de las librerías que se muestra en la *Figura 3.2* se emplean en las distintas líneas de código desarrolladas posteriormente para realizar las acciones necesarias para obtener la información de interés. A continuación, se va a realizar una explicación de cómo se emplean cada una de estas librerías.

1. *Librería cv2:* hace referencia a la librería Open CV que se utiliza principalmente para el procesamiento de imágenes ya que permite modificar las imágenes y videos para obtener información como su tamaño, brillo, convertirlas a otro formato de color o aplicar filtros que permitan eliminar el fondo de la imagen. Esta librería tiene funciones que permiten realizar la detección de contornos de objetos mediante un algoritmo de inteligencia artificial. Además, permite procesar vídeos analizando cada fotograma de manera aislada y visualizarlos como una secuencia de imágenes en otro vídeo con la información obtenida. En este trabajo de fin de grado, ha sido la librería más empleada para realizar las acciones de umbralizar, detección de contornos, transformación de píxeles o para generar elementos sobre la imagen que sirven de guía visual como son las elipses y las circunferencias. [14] [15]

2. *Librería numpy*: es una de las librerías más importantes cuando se trabaja con el lenguaje de programación Python ya que permite realizar operaciones matemáticas y trabajar con matrices de una manera rápida y eficiente. Esta librería permite almacenar la información en los "array" que hacen la función de listas que se pueden usar para realizar otro tipo de operaciones. Además, esta librería ofrece una mayor accesibilidad a funciones matemáticas como son las sumas, restas, el número pi, etc. Esta librería ha sido útil para delimitar la región de estudio para la detección de contornos permitiendo calcular de ella el radio del frente de llama y la celularidad. [16]

3. *Librería matplotlib.pyplot*: es otra de las librerías empleadas en este trabajo. Esta librería se emplea para la visualización por pantalla de los datos que está recogiendo el programa. Permite visualizar datos y estadísticas en gráficos en 2D del tipo que se necesite, presentar los datos de manera más eficiente o analizar esos datos usando tendencias o relaciones de datos que permitan una mejor comprensión del gráfico. Además, permite personalizar los gráficos en función de las necesidades desde el color que se emplea, el tipo de título, la tipografía o las leyendas de los ejes coordenados. En este trabajo ha sido una de las librerías fundamentales que han permitido comprobar si las acciones realizadas por la primera librería descrita eran las adecuadas o por el contrario necesitaba realizar mejoras. Una vez que se ejecutaban las acciones sobre el fotograma correspondiente, se guardan los datos en unas listas y con ayuda de esta librería se visualiza, en la interfaz, el gráfico con la información obtenida. [17]

4. *Librería pandas*: esta librería dispone de dos estructuras de datos que son las Series y los DataFrame, en este proyecto solo se usan los DataFrame. Los DataFrame son estructuras de datos bidimensionales a la que se les asocian etiquetas por filas y columnas. Pandas dispone de la función de escribir los datos en formatos como Excel que para este proyecto ha sido el seleccionado,

pero, además, también puede realizar un análisis de los datos y realizar una limpieza si es necesario. Para este trabajo, se ha usado para guardar los datos, que se han ido generando en cada fotograma, en un DataFrame que posteriormente se ha transcrito a formato Excel usando otro comando de esta librería. [18]

5. *Librería os*: es una librería que permite al usuario interactuar con el sistema operativo en el que se está ejecutando el programa, permitiendo manipular, gestionar y obtener la información necesaria de los archivos utilizados en el programa. En concreto para este proyecto se ha usado para poder manipular el archivo de video permitiendo dividirlo en dos partes. Una de las partes es la de su nombre base y en la otra parte se encuentra la extensión del video, permitiendo posteriormente guardar un archivo de Excel que recoge todos los datos con nombre base. [19]

3.1.2. Obtención de datos del video

Tras introducir todas las librerías necesarias que van a permitir tener accesos a todos los algoritmos precisados para ejecutar correctamente el programa, se procede a desarrollar los comandos necesarios que permiten ejecutar los procesos adecuados.

```
9   #  LECTURA DEL VIDEO
10  # Abrir el archivo de video
11
12  nombre_video = "C2H60100_Fr1_P264_T70_5.1.cine"
13  nombre_video_sin_extension = os.path.splitext(nombre_video)[0]
14  captura = cv2.VideoCapture(nombre_video)
15
16  # Crear el objeto BackgroundSubtractor
17  fgbg = cv2.createBackgroundSubtractorMOG2()
```

Figura 3.3: Lectura del video a analizar por el programa.

Como se puede observar en las líneas de código de la *Figura 3.3*, en la línea 12 se declara una variable “nombre_video” donde se llama al vídeo del cual se quiere obtener la información necesaria, ya sea para calcular el radio del frente de llama o para calcular la celularidad que vaya apareciendo a lo largo de todo el proceso de combustión. Es necesario saber que para que el programa pueda funcionar de manera correcta, es muy importante introducir bien el nombre del video. Hay dos modos de compilar este proceso: puede ser que el programa de Python con extensión “.py” se encuentre en la misma carpeta que los videos recogidos en la parte experimental (como es en el caso de la *Figura 3.3)*, o bien que el programa se encuentre en una carpeta diferente a donde estén guardados los videos y por tanto en la zona de color verde de la línea 12 lo que se debe de introducir es la ruta exacta del vídeo de estudio

Tras introducir el video que se desea analizar, en la línea 13 de la *Figura 3.3* se utiliza la librería "os" con uno de sus algoritmos para separar la variable "nombre_video", donde se ha guardado el video previamente, en dos que son el nombre base del video y su extensión, quedándose guardada únicamente el nombre base en una variable "nombre_video_sin_extensión". Esto se hace para posteriormente usar el nombre del vídeo para guardar los resultados de los datos que vaya obteniendo en un Excel, asociando el nombre del mismo con esta nueva variable.

Se procede a crear un objeto haciendo uso de uno de los algoritmos de la librería OpenCV, abreviada en el código bajo las siglas cv2, denominado "VideoCapture" cuya función es la de abrir el archivo de video definido previamente en la variable "nombre_video". Este algoritmo permite trabajar con una interfaz de archivos de video. Guardamos la acción en una variable denomina "captura" con la que se va a ir trabajando en el resto del programa.

En la línea de 17 de la *Figura 3.3*, se realiza la acción de generar un objeto que se encargue se sustraer el fondo del video mediante la función de "BackgroundSubtractorMOG2" de la librería OpenCV. Para poder realizarlo, se aplica el algoritmo "MOG2" (basado en el método Gaussiano); con él se pretende modelar un fondo del video a través de una distribución gaussiana segmentando el video. Lo que se pretende es que el video original y el video que se muestra por pantalla con todos los elementos que se analizan se parezcan lo máximo posible en cuando a posibles cambios de iluminación permitiendo posteriormente detectar los cambios de movimientos del proceso con mayor facilidad.

3.1.3. Comprobación de extensión del video

Antes de obtener las propiedades buscadas del video, es necesario realizar comprobaciones en cuanto al formato de video, para que el análisis posterior del mismo se haga con la máxima calidad y el mayor número de fotogramas para obtener unos resultados más precisos.

```
27  # Propiedades del video original
28  if nombre_video.endswith(".cine"):
29      fps = int(captura.get(cv2.CAP_PROP_FPS))  #Fotogramas que tiene el video
30  else:
31      fps = 3100
```

Figura 3.4: Comprobación de la extensión del video que se analiza.

En la *Figura 3.4*, se puede observar que se emplea una verificación para asegurar que todos los experimentos están bajo las mismas condiciones. La mayor calidad que se puede reproducir en el video se obtiene del formato ".cine", ya que en este formato es en el cual se guardan los video recogidos con la cámara Phantom V210 que graba los videos a 3100 fps. Por lo tanto, en estas líneas de código lo que se hace es una verificación para comprobar qué tipo de extensión es la que tiene el video guardado en la variable "nombre_video". Para ello se emplea, sobre esta

variable, un método, “endswith()”, donde si la extensión es “.cine”, se ejecuta la acción que hay dentro del “if” donde se obtienen el número de fotogramas que tiene el video haciendo uso de la librería OpenCV con una propiedad, “cv2.CAP_PROP_FPS” y se guarda en una variable definida como “fps”.

Por el contrario, si la extensión del video no acaba en “.cine”, se ejecuta la acción de definida en el “else” donde se predefinen los “fps” en los máximos que puede dar la cámara instalada en el laboratorio, que son 3100 fps. De esta manera se asegura de que todos los videos que se analizan lo hacen en las mismas condiciones.

3.1.4. Propiedades del video

En este punto se procede a realizar las operaciones sobre el video original para que el video que se muestra por pantalla se obtenga con las mismas características que el video original.

```
33    ancho = int(captura.get(cv2.CAP_PROP_FRAME_WIDTH))  #Ancho del video original
34    alto = int(captura.get(cv2.CAP_PROP_FRAME_HEIGHT))  #Alto del video original
35    #Número de fotogramas que tiene el video
36    n_fotogramas = int(captura.get(cv2.CAP_PROP_FRAME_COUNT))
37
38    # Redimension del video que se muestra por pantalla
39    nuevo_ancho = ancho // 2
40    nuevo_alto = alto // 2
```

Figura 3.5: Obtención de las propiedades del video original.

En esta parte del programa se obtienen tres propiedades del video original utilizando el video guardado en el objeto o variable “captura”.

En la línea 33 de la *Figura 3.5*, se genera una variable “ancho” que utiliza el argumento “CAP_PROP_FRAME_WIDTH”, función de la librería OpenCV, por la cual se obtiene el ancho del video original. Esta variable se transforma en un número entero a través del comando “int()”.

En la siguiente línea de código de la *Figura 3.5*, se ejecuta la misma acción que en la línea 33 pero en este caso se define la variable alto que guarda el número entero referente al alto del video original obtenida haciendo uso del argumento “CAP_PROP_FRAME_HEIGHT”.

Por último, se define la variable “n_fotogramas” donde se guarda el número entero referente al número total de fotogramas que tiene el video. Para ello se usa el argumento “CAP_PROP_FRAME_COUNT” que hace un recuento total del número de fotogramas que tiene el video. El valor recogido en esta variable va a ser distinto al dato albergado en la variable “fps”, que se puede visualizar en la *Figura 3.4,* puesto que en ella hace referencia a la velocidad con la que se está reproduciendo el video, mientras que en la variable “n_fotogramas” se trabaja con el número total de fotogramas en los que se compone el video.

Para poder mostrar por pantalla lo que ocurre en el mismo fotograma, pero con dos tipos de tratamientos de imágenes e imprimir por pantalla los gráficos que aportan información de los datos que se están generando y una tabla donde se muestra para cada fotograma lo que sucede los valores de las variables que se están calculando, es necesario redimensionar las imágenes que se muestran por pantalla, para ello se calculan dos variables que dividen a la mitad los datos almacenados en las variables "ancho" y "alto" definidas previamente. Estas nuevas variables aparecen en las líneas 39 y 40 de la *Figura 3.5.* El fotograma se divide a la mitad para que se puedan visualizar por pantalla toda la información que se vaya calculando en el desarrollo del código, es decir, que se muestre el fotograma que se está analizando junto con el mismo fotograma en escala umbralizado, las gráficas con los parámetros que se están calculando y una tabla con los datos calculados en cada fotograma. Si no se dividiera a la mitad, el fotograma ocuparía toda la pantalla y no se visualizaría correctamente el resto de la información.

3.1.5. Variable de pausa y creación del video de salida

Se necesita establecer una variable para poder parar el video en el instante que se desee durante el análisis del mismo por el programa. Además, es importante crear un video de salida que se va a mostrar por pantalla con las mismas características y propiedades que el video original.

```
42  # Variable para definir el botón de pausa
43  pausa = False
44
45  # Crear el video de salida con la informacion procesada con los datos recogidos en el bucle
46  out = cv2.VideoWriter('Resultado.cine', cv2.VideoWriter_fourcc(*'XVID'), fps, (ancho, alto))
```

Figura 3.6: Variable de pausa y características del video que se muestra por pantalla.

En la línea 43 de la *Figura 3.6*, se muestra una variable llamada "pausa" que se inicializa en "False", esto se debe a que en líneas posteriores del código si que toca la barra espaciadora del teclado, la variable "pausa" cambiará a "True" y el video se para en el fotograma. Se inicializa en "False" indicando que el video transcurre con normalidad.

En la *Figura 3.6*, se genera un objeto de salida que hace referencia al video de salida que se va a mostrar por pantalla donde se van a reproducir todas las operaciones que se ejecutan en procesos posteriores. Para ello, se emplea la acción de la librería OpenCv denominada "VideoWriter" donde en su interior se ejecutan los siguientes argumentos. El video de salida se va a denominar "Resultado.cine", se guarda con la misma extensión con la que analizan los videos de los experimentos generalmente. Seguidamente se utiliza el argumento "XVID" que usa un códex para comprimir el video de salida en un archivo de video. Para que el video de salida tenga las mismas propiedades que el video original, se deben pasar los datos recogidos en las variables "fps", obtenida en la *Figura 3.4*, y los datos de ancho y alto que se muestran en la *Figura 3.5*. No haría falta pasar el número de fotogramas que tiene el video

puesto que cuando el video original se acabe de reproducir, el video de salida se detendrá también.

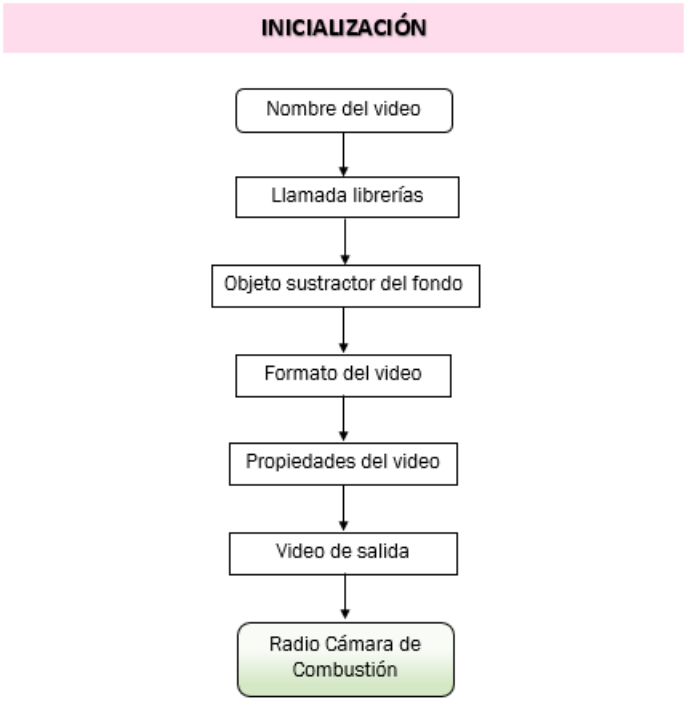


Figura 3.7: Diagrama de la de flujo de la apertura del video.

3.2. Radio de la cámara de combustión

Tras introducir en el apartado anterior todos los datos técnicos necesarios para poder obtener un video de salida por pantalla igual que el de entra, en este apartado se procede a explicar qué pasos se han seguido para obtener los datos necesarios sobre la cámara de combustión que son imprescindibles para hacer el cálculo posterior del radio del frente de llama o el cálculo de la celularidad. Para el cálculo de ambos procesos, es necesario realizar previamente el cálculo del radio de la cámara de combustión.

3.2.1. Obtención del primer fotograma y operaciones de transformación del fotograma

Para que el programa ejecute las operaciones de manera coherente, es necesario realizar un proceso previo donde se almacene únicamente el fotograma que se va a analizar en esta parte del programa.

```
49  #  RADIO CAMARA DE COMBUSTION
50  # Tratamiento del radio camara de combustion
51
52  ret, primer_fotograma = captura.read()
53
54  fotograma_inicial = 1
```

Figura 3.8: Parámetros de obtención del radio de la cámara de combustión.

En la *Figura 3.8*, se puede observar que en la línea 52 se emplea el método "read()" sobre la variable "captura" del video que se ha definido previamente para analizar. Este método se encarga de leer fotograma a fotograma del video y aplicar las funciones que aparecen en las líneas de código posteriores. Los parámetros que devuelve son dos valores. El comando "ret" hace referencia a un valor booleano que indica si el video se ha ejecutado de manera correcta, indicado con "True" si lo ejecuta. En caso de que no se haya no se haya podido ejecutar el video, sale de manera automática del programa. En la variable "primer_fotograma" se almacena el primer fotograma del video representado con una matriz de pixeles.

En la línea de código 54 de la *Figura 3.8*, se establece un contador inicial. Esto sirve para que, tanto en los bucles establecidos para el cálculo del radio del frente de llama, como para el cálculo de la celularidad, no se registren los datos asignados al primer fotograma y que directamente comiencen en el fotograma siguiente que es cuando realmente comienza la combustión.

Una vez almacenado el fotograma en una variable, se realizan varias operaciones para prepararlo y transformarlo para poder ejecutar las operaciones necesarias para obtener los datos de interés.

```
55  # Transformamos a escala de grises el fotograma almacenado
56  grises = cv2.cvtColor(primer_fotograma, cv2.COLOR_BGR2GRAY)
57
58  # Obtener la imagen umbralizada
59  umbral = cv2.threshold(grises, 100, 255, cv2.THRESH_BINARY)[1]
60
61  # Encontrar contornos
62  contornos, jerarquia = cv2.findContours(umbral, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)
```

Figura 3.9: Parámetros de obtención del radio de la cámara de combustión.

Con el primer fotograma almacenado en una variable, se procede a calcular los datos necesarios. En la línea 56 de la *Figura 3.9*, se transforma la variable "primer_fotograma" a escala de grises utilizando para ello la función "cvtColor()", de la librería OpenCV. Para poder realizar de manera correcta la transformación, se pasa de una imagen en formato "BGR" (Blue, Green and Red) a escala de grises. Dentro de la función "cvtColor", primero se establece la variable que se desea transformar, "primer_fotograma" y seguidamente se aplica el parámetro a esa variable por medio de otro argumento de la librería OpenCv que se denomina "COLOR_BGR2GRAY". Todo ello se guarda en una variable denomina "grises". Este paso podría no ser necesario ya que los videos que recoge la cámara de alta velocidad de la instalación lo hacen en escala de grises, pero es un paso que se realiza a modo de comprobación primera.

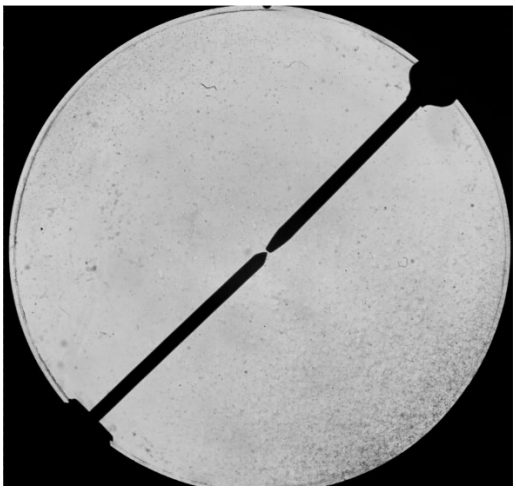


Figura 3.10: Imagen en escala de grises del primer fotograma referente a la cámara de combustión para el experimento BC_E15.

Seguidamente a ese proceso, todo referenciado en la *Figura 3.9*, se aplica un umbral a la variable "grises" que recoge el primer fotograma en escala de grises. Para poder aplicar este umbral, se usa la función, de la librería OpenCV, denominada "threshold()" donde en su interior se puede observar que el primer parámetro es la variable a la que se desea aplicar el umbral, se establece un umbral de 100 que es el valor de intensidad al que se van a mostrar los pixeles, siendo el máximo de 255 que representa los pixeles en blanco. La imagen que se va a recoger debe de estar en formato binario por lo que se hace uso de la función "THRESH_BINARY". Como toda la línea de código se recoge sobre una matriz, es necesario indicarle que solo debe de guardar en la variable "umbral" lo obtenido en la posición "[1]" de la lista, la cual corresponde a una matriz en la que está contenida la información referente a los píxeles del "primer_fotograma".

En la línea del código 62 de la *Figura 3.9*, lo que se busca es encontrar los contornos de la imagen umbralizada. Para ello se hace uso de una de las funciones de la librería OpenCV por la que se ha desarrollado este proyecto que es la de "findCountours". En ella se deben pasar los parámetros de la imagen que se desea analizar como entrada. Con el comando "RETR_LIST", se crea una lista donde se almacenan de manera desordenada todos los contornos que se detectan en la imagen de entrada. Para que la representación de los contornos que se encuentran sea más sencilla de comprender y con la que sea más sencilla establecer posteriormente los cálculos, se almacenan en la lista que se ha generado una versión simplificada de los contornos, para ello usamos la función "CHAIN_APPROX_SIMPLE" para ejecutar esta acción. Todos estos parámetros se recogen en una variable denominada "contornos" y en otra denominada "jerarquía" que lo que hace es ordenar la lista de los contornos de mayor a menor tamaño.

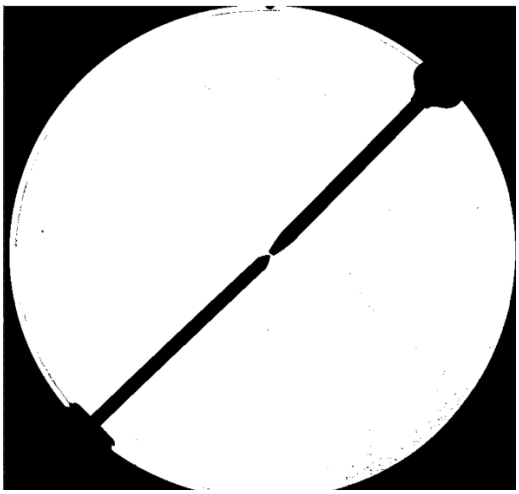


Figura 3.11: Imagen umbralizada del primer fotograma referente a la cámara de combustión para el experimento BC_E15.

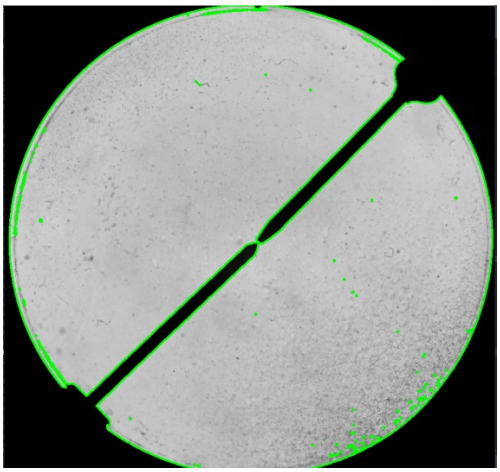


Figura 3.12: Imagen de los contornos detectados en el primer fotograma referente a la cámara de combustión para el experimento BC_E15.

3.2.2. Operaciones para la obtención del radio de la cámara de combustión

Tras preparar el fotograma con los procesos desarrollados en el punto anterior, se pueden ejecutar las operaciones necesarias que permiten obtener los valores del radio de la cámara de combustión y del centro de esta.

```
64  # Encontrar la región de mayor área
65  max_contorno = max(contornos, key=cv2.contourArea)
66
67  # Ajustar elipse a la región
68  elipse = cv2.fitEllipse(max_contorno)
69
70  # Obtener centro y radio de la elipse
71  centro, ejes, angulo = elipse
72  eje_mayor, eje_menor = ejes
73  radio = (eje_mayor + eje_menor) / 4
74  centro = (int(centro[0]), int(centro[1]))
75  print("El centro de la elipse es {}" .format(centro))
76  print("El radio de la camara de combustiones de {}" .format(int(radio)))
```

Figura 3.13: Detección de contornos máximos, ajuste a una elipse y obtención de los parámetros de la elipse.

Una vez detectados todos los contornos de la variable “primer_fotograma”, se procede a encontrar las regiones de esos contornos donde el área sea máxima, como se muestra en la línea 65 de la *Figura 3.13*. Para ello es necesario indicarle al programa que debe de encontrar los contornos máximos, “max()” función de la librería numpy, donde se introducen las variable contornos, variable que se desea analizar, y se obliga a que haga un cálculo del área de todos los contornos que haya detectado por medio de la función “contourArea” de la librería OpenCV. Esta línea de código permite eliminar los contornos de menor importancia considerados ruido de

fondo en los videos y que no aportan información relevante para el cálculo del radio de la cámara de combustión. Los contornos de mayor área se guardan en una variable llamada “max_contorno” que genera una lista donde ordena los contornos de mayor a menor.

Posteriormente, se procede a realizar un ajuste de todos los contornos detectados y que se encuentran almacenados dentro de la variable “max_contorno”. Para que este ajuste sea óptimo se ha seleccionado ajustar los contornos al área de una elipse puesto que si se mira la cámara de combustión desde el exterior de la instalación se observa que se puede asemejar a una circunferencia, Para poder realizar este ajuste, se ha usado la función de la librería OpenV denominada “fitEllipse()” y se le aplica la lista de contornos. El resultado se almacena en una variable llamada “elipse”.

A pesar de realizar el ajuste a las elipses para que se ajustara de manera correcta a los contornos de mayor área, lo que realmente interesa es que se pueda ajustar esta elipse a una circunferencia equivalente, correspondiendo así a una aproximación del contorno de la cámara de combustión. No sería una circunferencia de área equivalente ya que para calcular el radio de dicha circunferencia equivalente se realiza una suma de los diámetros de la elipse y se divide todo ello entre 4. Para ello necesario obtener los principales parámetros de la elipse como se muestra en la *Figura 3.13*. Se van a almacenar en tres variables distintas los datos recogidos de la variable “elipse”, siendo estos el centro de la elipse, los ejes de la elipse y el ángulo de inclinación de dicha elipse. Los datos que permitir ajustar la elipse a una circunferencia son principalmente los ejes, mayor y menor, de la elipse puesto que como se muestra en la línea de código 73 se realiza la suman de ambos valores y se divide entre 4. El resultado de la operación se almacena en la variable “radio”.

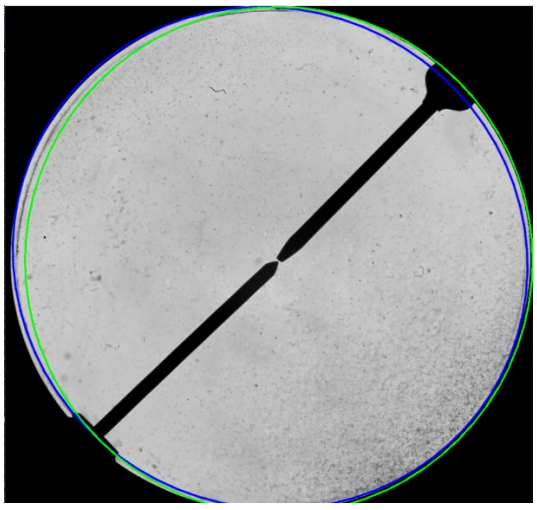


Figura 3.14: Imagen de la elipse, en color azul, y la circunferencia, en color verde, que se aproximan a los contornos detectados en el primer fotograma referente a la cámara de combustión para el experimento BC_E15.

También es importante saber que la variable donde está almacenada el centro de la elipse, los datos están en pixeles tomados con respecto a la esquina superior izquierda del fotograma. Para poder trabajar posteriormente con esta variable, es necesario transformar los pixeles a número entero por medio de la función “int()” y almacenarlos de nuevo en la misma variable, sobrescribiendo los datos anteriores. Para que el programa detecte cuál es cada una de las posiciones que ocupan los datos dentro de la variable y transformarlos de manera individual, se debe de asignar la posición [0] a la coordenada x y la posición [1] a la coordenada y.

Las líneas de código 75 y 76, de la *Figura 3.13*, hacen que se muestre por la consola los datos recogidos en las variables “centro” y “radio” calculadas previamente. Esto se realiza a través del comando “print”. Este tipo de comandos permiten llegar un control de por dónde se está ejecutando el programa y permite conocer, si el programa sufre algún problema, hasta qué punto se ejecuta de manera correcta y desde dónde es necesario realizar una revisión del programa.

3.2.3. Calibración del radio y ajustes del contorno

Una vez ejecutadas todas las operaciones sobre el fotograma para obtener los valores almacenados en variables denominadas “radio” y “centro”, referentes ambos a datos de la cámara de combustión, se calculan unos factores de ajuste que servirán posteriormente para otros cálculos referentes a la obtención del radio del frente de llama y contabilización de la celularidad.

```
78  # Calculamos la variable calibracion que nos sirve de variable de control para el proceso del calculo del radio del frente de llama
79  calibracion = radio/5.7
80  print("La calibracion es de {}" .format(calibracion))
```

Figura 3.15: Cálculo de la variable calibración.

Tras recoger en variables los datos referentes a la cámara de combustión, siendo estos el radio y el centro de la cámara de combustión, se procede a comparar los datos obtenidos con la medida real del radio de la cámara de combustión. Esta comparación, mostrada en la *Figura 3.15*, se realiza por medio de la variable “calibración” buscando obtener un valor adimensional que permita relacionarlo posteriormente con los radios del frente de llama en los diferentes fotogramas. Lo que se busca es comparar el radio original de la cámara de combustión, dato obtenido en la instalación de la bomba, con un valor de 5,7 cm. Dividimos el valor obtenido por análisis de imágenes entre el valor original. Este dato suele ser muy similar en todas las combustiones, no obstante, puede haber factores exteriores que puedan modificar este dato como puede ser suciedad de los cristales o algún reajuste realizado en la cámara Phanton V210.

Posteriormente, se muestra por pantalla el valor almacenado en la variable “calibración” mediante el comando “print”. Esto permite llevar un seguimiento de cómo se modifica esta variable en las diferentes combustiones.

```
85  # Calculamos el centro que vamos a considerar real
86  centro_real = (int(centro[0]) + 14, int(centro[1]) + 5)
87  print("El centro de la camara de combustion es {}" .format(centro_real))
88  cv2.circle(primer_fotograma, centro_real, int(radio), (255, 0, 0), 2)
```

Figura 3.16: Obtención del centro real de la cámara de combustión y representación del radio de la cámara de combustión.

Una vez obtenidos todos datos necesarios que hacen referencia al primer fotograma del video, se dibuja una circunferencia con el comando "circle" de la librería OpenCV y representado en la *Figura 3.16* en la línea de código 88. Para que este comando pueda ejecutar correctamente, es necesario indicar el lugar en el que se desea mostrar el resultado, en este caso en el fotograma almacenado en la variable "primer_fotograma", puesto que hace referencia al fotograma original que no ha sufrido las modificaciones que se han ido realizando posteriormente.

Seguidamente a este dato, se encuentra una variable denominada "centro_real". Esta variable, calculada previamente en la línea 86 de la *Figura 3.16*, se ha ajustado de manera visual, puesto que tras realizar la instalación de la bomba y colocar los electros lo más próximos al centro geométrico de la cámara de combustión, este proceso es lo más preciso para el ojo humano, pero para la cámara de alta velocidad, se pueden producir pequeñas desviaciones con respecto al centro que el programa considera como bueno. Para evitar que se desvíe el programa más de lo considerado, pudiendo afectar esto a procesos posteriores, se hace un ajuste visual de la posición en la que están colocados los electros, donde salta realmente la chispa que produce la ignición del proceso. Para poder realizar este ajuste lo más precioso posible, se han pasado por el programa varios videos de los diferentes experimentos y se ha decidido sumarle a la coordenada "x" 14 pixeles y a la coordenada "y", 5 pixeles. Si para futuros experimentos se modifica la posición de los electros, sería necesario realizar el mismo reajuste visual para que el programa pueda funcionar de la manera más correcta y aproximada posible. Posteriormente se muestra por la consola el dato almacenado en la variable "centro_real" haciendo uso del comando print.

Una definida el centro real que se considera para el resto del programa, se pasa al comando encargado de dibujar el radio de la cámara de combustión, el radio de la circunferencia, almacenado en la variable "radio". El int delante de este dato, lo que hace es transformarlo a un número entero para poder trabajar con un dato que sepamos manejar. Los siguientes dos datos lo que definen es el color, siendo el azul para el siguiente formato (255, 0, 0), y el grosor de línea con el que se va a representar la circunferencia en el fotograma.

3.2.4. Visualización de resultados

Tras obtener todos los datos referentes al cálculo de la cámara de combustión, se procede a mostrar por pantalla los resultados.

```
90    # Mostrar imagen resultante
91    cv2.imshow('Contorno de la cámara de combustión', primer_fotograma)
```

Figura 3.17: Representación del primer fotograma con la circunferencia del radio de la cámara de combustión.

Para finalizar el cálculo del radio de la cámara de combustión, se representan las modificaciones realizadas en la imagen original, es decir, en la variable almacenada al principio de este apartado denominada "primer_fotograma". Como sobre esta variable se ha ejecutado el algoritmo descrito en la *Figura 3.16*, cuando se muestre por pantalla, haciendo uso de la función "imshow()", también se mostrará el resultado de la circunferencia calculada. Para que se ejecute de manera adecuada, es necesario poner un título a pantalla emergente que en este caso se denomina 'Contorno de la cámara de combustión' y pasarle al algoritmo el comando que se desea representar.

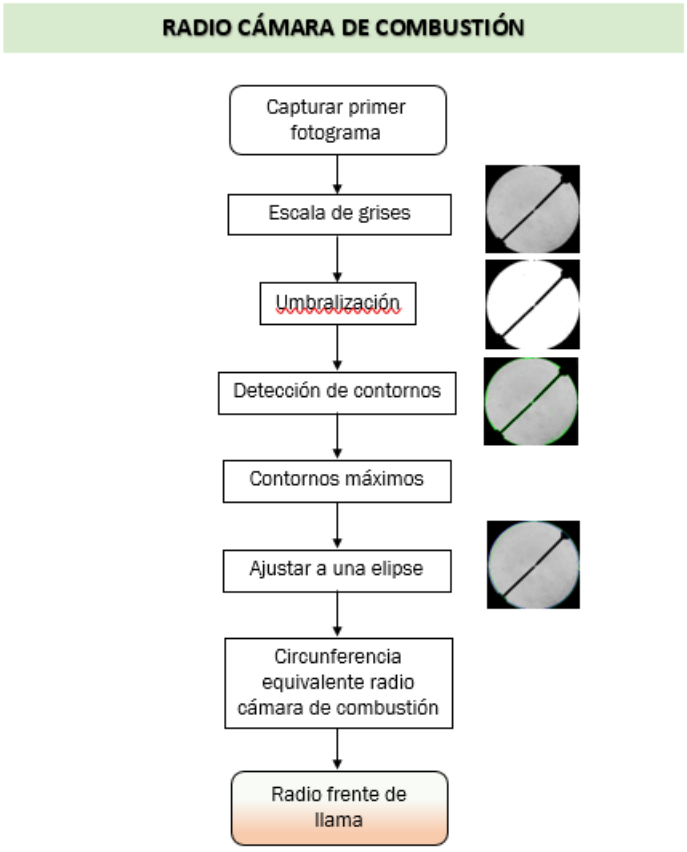


Figura 3.18: Diagrama de la de flujo de la detección y cálculo de los parámetros iniciales de la cámara de combustión.

3.3. Cálculo del radio del frente de llama

Una vez explicada la manera de calcular los parámetros necesarios que permitirán posteriormente graficar los resultados, se explica el procedimiento utilizado para calcular el radio del frente de llama en cada fotograma.

Los procesos seguidos para realizar el seguimiento del radio del frente de llama son similares a los explicados anteriormente en el análisis del primer fotograma para obtener el radio de la cámara de combustión. La diferencia principal radica en que, mientras que el radio de la cámara era único, el radio del frente de llama evoluciona a lo largo del tiempo, por lo tanto para este caso, es necesario introducir todos los comandos dentro de un bucle iterativo, “while”, que aplicada todos los algoritmos de manera consecutiva a cada uno de los fotogramas del video.

3.3.1. Declaración de variables y contador de fotogramas

Como el proceso que se sigue para el cálculo del radio se encuentra definido dentro de un bucle cerrado, es necesario declarar una serie de variables previas que son las encargadas de ir registrando todos los datos sin que se vaya sobrescribiendo uno sobre otro.

```
94   #  RADIO FRENTE DE LLAMA
95   # Tratamiento de fotogramas
96
97   #Contador de fotogramas
98   contador_fotogramas = 1
99
100  # Listas que almacenan los datos generados del radio y el tiempo
101  x_tiempo_radio = []
102  y_radio = []
103  x_tiempo_celularidad = []
104  y_celularidad = []
```

Figura 3.19: Declaración de variables antes del bucle “while”.

Como se puede observar en la *Figura 3.19*, se inicializa una variable denominada “contador_fotogramas” asignada con el valor inicial igual a la unidad. Esta variable permite realizar un conteo de fotogramas en el interior del bucle, permitiendo que el programa pase de un fotograma a otro una vez finalizadas las operaciones necesarias sobre el mismo y registrados los valores buscados. Tras realizar este proceso el programa suma 1 a dicha variable contador.

En las líneas 101, 102, 103 y 104 del código se muestran unas variables. Estas variables tienen una característica especial que es la de definir una estructura de datos en formato de lista, permitiendo albergar en su interior cualquier tipo de datos. Para hacer referencia a que se va a usar una variable de lista de datos se define

haciendo uso de corchetes. Para el caso particular del programa desarrollado, se definen cuatro listas de variables diferentes, una de ellas, “x_tiempo_radio”, se encarga de guardaren su interior los valores referentes al tiempo, se explicará cómo se obtiene más adelante, en otra, “y_radio”, se registran los valores referentes a la evolución del radio del frente de llama. Seguidas de las listas que almacenan los datos referentes al radio, aparecen la lista “x_tiempo_celularidad” que almacena los mismos valores que la lista “x_tiempo_radio” pero al graficar los datos eran necesario que fueran listas independientes; y por último, la lista “y_celularidad” que almacenan los datos referentes a la variable “densidad_celular” con la información del número de celdas que se recogen para cada fotograma.

3.3.2. Apertura del bucle “while” y definición de parámetros referentes a la captación de fotogramas

El proceso de análisis para obtener los datos del radio del frente de llama es un proceso iterativo por lo que se necesita introducir todo el programa referente al cálculo de variables y datos dentro de un proceso iterativo.

```
107   while True:
108     if not pausa:
109       ret, fotograma = captura.read()
110
111
112       # Salir del bucle cuando se haya procesado todo el video
113       if not ret:
114           break
115       fotograma = cv2.resize(fotograma, (nuevo_ancho, nuevo_alto))
116       copia = fotograma.copy()
117       contador_fotogramas += 1
```

Figura 3.20: Apertura del bucle iterativo y establecimiento de parámetros referentes al video.

En la línea de código 107 de la *Figura 3.20,* se inicia el bucle “while” mencionado que ejecutará las acciones repetidamente hasta que encuentre con una instrucción “break”, o el programa se detenga porque haya finalizado el video. Dentro de este bucle “while” se van a encontrar todas las acciones y operaciones que se realizan para poder obtener los datos del radio del frente de llama partiendo de un fotograma.

Tras la inicialización del bucle “while”, es importante conocer que se van a ejecutar todas las acciones si el botón que pausa el video no se ha presionado, es decir, se ejecutan las acciones dentro de un bucle “if not pausa:” ya que la variable “pausa” se ha definido, línea 43 *Figura 3.6*, desde el comienzo en “False” hasta que no se presiona la tecla.

En la siguiente línea de código de la *Figura 3.20*, se lee un fotograma del objeto que está almacenado en la variable “captura”, que en este caso es el archivo de video que se está analizando. Tras leer el fotograma del video, se obtienen el valor de “ret”,

que puede ser “True” o “False”, y el valor de “fotograma” donde se encuentra almacenado el fotograma que se va a estudiar por medio de análisis de imágenes.

Seguidamente a la obtención de ambas variables (“ret” y “fotograma”), aparece un bucle de verificación empleando el algoritmo de otro bucle denominado “if not”. Si el valor de la variable “ret” es “True” significa que la lectura del video tuvo éxito y no entraría dentro del bucle “if not” planteado y continúa ejecutando el código. En caso contrario, si el valor de “ret” es “False”, como está establecido en la condición “if not”, significaría que la operación de lectura de fotogramas no ha sido exitosa o que no hay más fotogramas del video que se lean; entonces se emplea la acción “break” para salir del programa indicando que este ha terminado de procesarse.

Con el fotograma almacenado, se reescala por medio de la función de la librería OpenCV denominada “cv2.resize()” a la cual se le introduce la imagen que se desea modificar y los valores del nuevo ancho y nuevo alto que se sean establecer, en este caso son la mitad de los del video original, pero los datos están almacenados en las variables “nuevo_ancho” y “nuevo_alto” definidas en las líneas 39 y 40 de la *Figura 3.5*. Además, se hace una copia de ese fotograma, usando la función “.copy()”, ya que se va a usar para generar una máscara que permita detectar la celularidad en una determinada zona.

Por último, se puede observar que vuelve aparecer la variable “contador_fotogramas”. La acción que se lleva a cabo en este caso consiste en incrementar su valor en una unidad para vez que se produce una iteración del bucle. Esto permite llevar un seguimiento del número de fotogramas que se han procesado.

3.3.3. Procesamiento del fotograma

Una vez almacenado el fotograma en la variable “fotograma”, se procede a ejecutar los parámetros de análisis de contornos que permiten preparar el fotograma para poder ejecutar posteriormente las operaciones para encontrar el radio del frente de llama.

```
120     # Aplicar el sustractor de fondo para obtener la imagen de los objetos en movimiento
121     mascara = fgbg.apply(fotograma)
122
123     # Aplicar un umbral para obtener solo los píxeles más brillantes en la imagen
124     umbralizacion = cv2.threshold(mascara, 127, 255, cv2.THRESH_BINARY)[1]
125
126     # Encontrar los contornos en la imagen binaria
127     contornos, hierarchy = cv2.findContours(umbralizacion, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
```

Figura 3.21: Funciones de procesamiento de imágenes.

La primera operación de procesamiento del fotograma que se muestra en la *Figura 3.21,* consiste en aplicar sobre el fotograma un sustractor de fondo. Esta técnica consiste en separar los objetos en movimiento del fondo estático. Para ello llama al elemento “fgbg” definido en los parámetros del video, en el apartado 3.1.2. Al

sustraer el fondo estático de la imagen, se genera una máscara que resalta las regiones en las que se detecta movimiento en relación con el fondo estático. En la variable “mascara” se almacena el fotograma con los pixeles resaltados respecto al fondo estático.

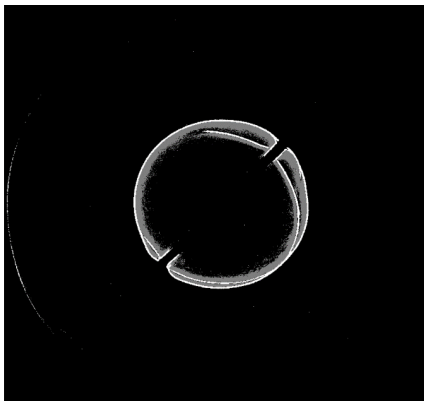


Figura 3.22: Imagen pixeles resaltados de un fotograma para el cálculo del radio del frente de llama del experimento BC_E15.

Tras resaltar los pixeles en movimiento con respecto a los del fondo, se aplica la función umbral, “threshold” de la librería OpenCV, donde se dividen los pixeles que superan el umbral con respecto a los que están por debajo del mismo. Para este caso, se ha empleado un umbral de 127, como valor límite inferior, y 255, como valor límite superior, siendo estos valores definidos por el usuario como un rango. Se ha establecido como valor superior 255 porque no hay ningún valor más alto, mientras que se ha tomado como valor límite inferior un número intermedio evitando así que se cojan únicamente pixeles muy cercanos al blanco que no aportan información relevante de estudio, solo la proporcionan los pixeles oscuros. Para que se puedan aplicar de manera adecuada los procedimientos mencionados anteriormente se emplea el tipo de umbralización que se busca conseguir sobre el fotograma, en este caso “THRESH_BINARY” (explicado anteriormente en el apartado 3.2.1.) siendo este el método más sencillo y común donde se evalúan los pixeles en función de los valores umbral y el umbral máximo establecido anteriormente. El resultado del procesamiento del fotograma se almacena en la variable “umbralizacion”.

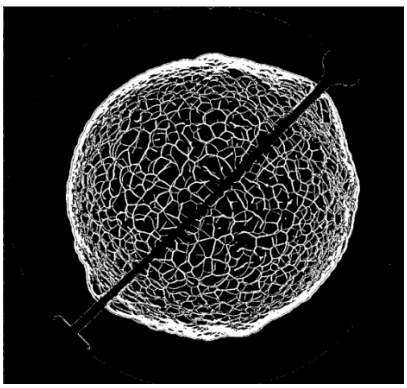


Figura 3.23: Imagen umbralizada de un fotograma para el cálculo del radio del frente de llama del experimento BC_E15.

La imagen binaria umbralizada permite detectar con mayor facilidad los contornos del fotograma. Para la detección de contornos que se muestra en la *Figura 3.21*, se hace uso de la función "findContours()" perteneciente a la librería OpenCV. Para que el algoritmo reconozca un contorno del fotograma, este debe ser una curva cerrada o una serie de puntos que representan los límites de un objeto. Para que la función ejecute la labor para la que está planteada, es necesario establecer como variable de entrada el fotograma, con el procesamiento de pasos anteriores realizado, almacenado en la variable "umbralizacion". Seguidamente a este parámetro se definen los modos que permiten encontrar los contornos, ambos pertenecientes a la librería OpenCV. El modo "RETR_EXTERNAL" especifica que solo se buscan contornos externos dentro de la imagen, es decir, no contabiliza los contornos que se encuentren dentro de otros. El segundo algoritmo, "CHAIN_APPROX_SIMPLE" realiza una aproximación de los contornos, permitiendo al programa ahorrar almacenamiento e ir más rápido, eliminando los redundantes. La información detectada en esta línea de código se almacena en la lista denominada "contornos". Por otro lado, la variable "hierarchy" es una matriz que contiene información de la topología de los contornos, es decir, almacena los valores que indican cómo se conectan unos contornos con otros y cómo se organizan estos dentro de la imagen. En concreto, esta variable no se usa en el programa, pero es importarte tener almacenada esta información por si fuera necesario recurrir a ella en futuras modificaciones de mejora del programa.

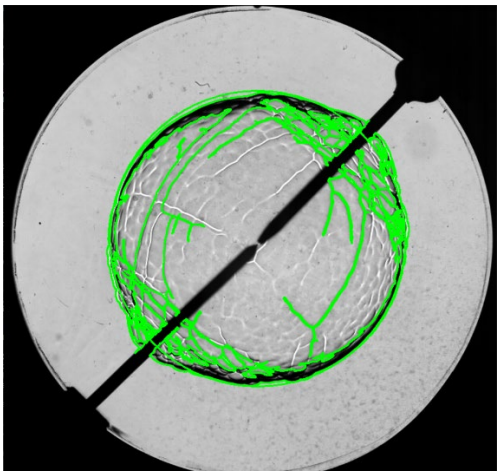


Figura 3.24: Imagen de los contornos más grandes de un fotograma para el cálculo del radio del frente de llama del experimento BC_E15.

3.3.4. Operaciones de análisis de contornos

Tras aplicar todos los procedimientos de análisis del fotograma para obtener de él la información buscada, se procede a realizar operaciones de procesamiento de la información para poder trabajar con ella de manera adecuada para lograr el objetivo de detección del radio del frente de llama.

```
129     # Ordenar los contornos por área de mayor a menor
130     contornos = sorted(contornos, key=cv2.contourArea, reverse=True)
131
132     # Tomar los dos primeros contornos (los de mayor área)
133     modificacion_area = 3
134     contornos_grandes = contornos[:modificacion_area]
```

Figura 3.25: Algoritmos de análisis de la información recogida en la variable contornos y selección de los datos necesarios.

En la parte del código que se muestra en la *Figura 3.25* se realizan operaciones relacionadas con los contornos detectados en el fotograma binarizado previamente (en la variable "umbralización"). Primero se ordenan los contornos almacenados en la lista "contornos" y posteriormente se ordenan en función de su área.

Para poder realizar la primera de las acciones, como se muestra en la línea de código 130 de la *Figura 3.25*, es necesario pasarle al algoritmo planteado con la función "sorted()" la lista que se desea ordenar, en este caso donde se encuentran almacenados los datos de los contornos. La función "sorted()" permite ordenar los datos en base a un orden, en concreto se busca que los contornos más grandes aparezcan al comienzo de la lista, es decir, se busca ordenar la lista de mayor a menor área, se establece este orden para que posteriormente se tomen solo los contornos más grandes se considera que estos los que van hacer referencia al frente de llama mientras que los contornos más pequeños sobre todo contienen información del fondo del fotograma o ruido y perturbaciones que no se consideran en el estudio. Para poder ejecutar de manera correcta ese orden se usa el parámetro "reverse = True". El parámetro que aparece después de pasar la lista "contornos", "key = cv2.contourArea" se encarga de calcular el área de los contornos detectados, donde solo necesita para su cálculo la lista "contornos" devolviendo el valor del área del contorno en pixeles al cuadrado. Como conclusión, la función planteada en la línea de código 130 calcula el área de los contornos y los ordena de mayor a menor en una lista denominada "contornos".

Tras ordenar el área los contornos, se inicializa una variable denominada "modificación_area" que establece el número de áreas que se van a usar para aproximar el radio del frente de llama. Este dato se puede ir modificando en función de las necesidades del video que se está analizando, permitiendo una aproximación más fina en los cálculos posteriores. No se puede trabajar con un número menor de 2 contornos ya que le programa no tiene suficiente información para realizar los cálculos e indica un mensaje de error por la consola indicando que no se ha podido realizar la operación del cálculo de la variable "radio", definida más abajo. Por lo tanto, para poder trabajar con el programa es necesario seleccionar para la variable "modificacion_area" 2 o más contornos dependiendo de las necesidades del video que se analiza.

Esta variable definida en la línea de código 133 de la *Figura 3.25* sirve para indicar cuántos contornos se van a tomar de la lista que almacena todas las áreas de los contornos, para el caso mostrado en la *Figura 3.25* solo se están considerando tres. Como la lista "contornos" está ordenada, se van a almacenar en otra lista, con la que posteriormente se van a realizar los cálculos, el número de contornos establecidos por el usuario por medio de la variable "modificación_area".

3.3.5. Cálculo de los parámetros necesarios para la detección del radio del frente de llama

Una vez realizadas las operaciones de análisis de contorno, se procede a realizar una aproximación de los mismos en función de los contornos escogidos de la variable "modificación_area" explicada en la *Figura 3.25.*

A nivel general la acciones que se van a llevar a cabo en el apartado que se va a desarrollar consiste en encontrar un rectángulo que contenga en su interior todos los contornos escogidos en la variable "modificación_area".

```
136     # Encontrar la elipse circunscrita
137     if len(contornos_grandes) == modificacion_area:
138         rectangulo = cv2.minAreaRect(np.vstack(contornos_grandes))
139         centro_rectangulo = rectangulo[0]
140         diametro_mayor = max(rectangulo[1])
141         diametro_menor = min(rectangulo[1])
142         angulo = rectangulo[2]
143         radio = (diametro_mayor * diametro_menor) / (diametro_mayor + diametro_menor)
144         radio_celularidad = radio * 0.8
```

Figura 3.26: Procedimiento de cálculo para la detección del rectángulo que contiene en su interior las elipses que forman los contornos.

Para poder realizar las operaciones de manera correcta, es necesario realizar la comprobación que permita poder calcular la elipse circunscrita por lo que nos tenemos que asegurar que se han encontrado los suficientes contornos. Todo el procedimiento que se va a desarrollar a continuación y en apartados posteriores está establecidos bajo esta premisa, por lo que, si el programa no encuentra el número suficiente de áreas de contornos, el sistema va a registrar valores nulos y por pantalla se muestra el video, pero sin visualizar el seguimiento del radio del frente de llama, puesto que los valores que detecta son nulos. La premisa que se muestra en la línea de código 137 de la *Figura 3.26,* referente a lo mencionado anteriormente, se va a cumplir en todos los casos, puesto que los videos que se analizan van a detectar aunque sea dos contornos con los que poder ejecutar la acción. En concreto lo que se desarrolla en la línea de código mencionada es que la cantidad de contornos de la lista "contornos_grandes" es igual a la cantidad establecida por medio de la variable "modificación_area". Como la acción que se va a realizar es una condición que se puede cumplir o no, se emplea el condicionante "if" para ejecutar las operaciones.

Si se cumple la condición del bucle planteada, se procede a realizar el cálculo del rectángulo de área mínima que envuelve o circunscribe los contornos establecidos. Para ello se usa la función "minAreaRect()" de la librería OpenCV que se muestra en la línea de código 138 de la *Figura 3.26,* que permite calcular el rectángulo de área mínima que recoge en su interior los contornos establecidos antes. La función toma como entrada una matriz de la librería Numpy que contiene los puntos de los contornos almacenados en la lista "contornos_grandes". La función que recoge todos los puntos de los contornos analizados en una sola matriz vertical a lo largo de su eje es "np.vstack()", es decir, esta función agrupa los contornos uno tras otro. Con esta nueva matriz generada, se calcula el área mínima que alberga en su interior a todos los puntos de los contornos evaluados y la información calculada se recoge en la variable "rectangulo" que contiene la información referente al centro, diámetros y ángulo de rotación del rectángulo.

La información almacenada en la lista "rectangulo" se va a repartir en variables para poder realizar operaciones posteriores de una manera más sencilla, como se muestra de las líneas 139 a la 142 de la *Figura 3.26*. En la lista "rectangulo" el primer valor que se almacena corresponde al centro del rectángulo, por ello se almacena en una variable "centro_rectangulo" y se hace referencia al primer valor de la lista "rectangulo" asignado la posición del dato en la lista entre corchetes, "[0]". Para los diámetros del rectángulo, los datos están almacenados en la segunda posición de la lista "rectangulo" y para hacer referencia a ella se emplea su posición entre corchetes, "[1]". Puesto que en esa lista están almacenados numerosos datos relacionados con los contornos, es necesario asegurarse de que se toman los valores máximos, por la función "max()", y mínimos, por la función "min()", asignándoselos al diámetro mayor y al diámetro menor respectivamente. Por último, se recoge en una variable el ángulo de rotación del rectángulo ya que este va a ir variando, dependiendo de cómo sean los contornos evaluados en cada fotograma y se almacena en la variable "angulo" haciendo referencia a la tercera posición, "[2]", de la lista "rectangulo".

En la línea de código 143 de la *Figura 3.26,* se calcula una nueva variable, "radio" a partir de los datos almacenados correspondientes a los diámetros. Esta variable es muy útil para realizar comparaciones y cálculos en puntos posteriores como se detalla más abajo. Consiste en realizar la inversa de la suma de las inversas de los diámetros mayor y menor captados en las variables mencionadas anteriormente, esto permite obtener un radio que coincide con el radio de la circunferencia circunscrita dentro del rectángulo y coincidiendo de una manera muy aproximada con el radio del frente de llama para cada fotograma.

$$\frac{1}{R} = \frac{1}{D_{mayor}} + \frac{1}{D_{menor}} \quad \rightarrow \quad R = \frac{D_{mayor} * D_{menor}}{D_{mayor} + D_{menor}} \qquad \textit{(Ec. 8)}$$

Por último, se define la variable "radio_celularidad" en la línea 144 de la *Figura 3.26*, es el 80% del radio calculado por el método anterior. Se hace de esta manera para

garantizar que, en el interior de este nuevo radio, aunque se desvíe, se encuentren siempre celdas, en el caso de que se produzca la celularidad en el frente de llama.

3.3.6. Dibujar radio de aproximación de contornos

Tras obtener los datos necesarios y almacenarlos en las variables con las que poder trabajar, se muestran sobre el fotograma los contornos detectados y la aproximación por medio de una circunferencia del radio del frente de llama.

```
146         # Dibujar los contornos que está tomando como ejemplo
147         cv2.circle(fotograma, (int(centro_rectangulo[0]), int(centro_rectangulo[1])), int(radio), (0, 255, 0), 2)
```

Figura 3.27: Dibujar los contornos sobre el fotograma analizado por el programa.

En la línea de código 147 de la *Figura 3.27*, se muestra la forma de ejecutar la circunferencia que representa la aproximación del radio del frente de llama sobre el fotograma. Para poder llevar a cabo este procedimiento se utiliza la función ".circle()" donde se deben de pasar una serie de datos. En primer lugar, se pasa la variable "fotograma" que almacena el fotograma que se está analizando. Seguido de ello se pasa el centro de la circunferencia, su posición en el eje de las x y en el eje de las y, que en este caso es el centro del rectángulo que contiene la circunferencia circunscrita. Este dato se debe pasar como un número entero para que el sistema comprenda mejor el dato con el que se está trabajando. Posteriormente, se pasa el dato almacenado en la variable "radio" explicada en el apartado 3.3.5. que almacena en un formato con el que el sistema no puede trabajar, por lo que puede entenderlo si se pasa a número entero. Los dos últimos parámetros se refieren al color en el que se dibuja la circunferencia, siempre en el parámetro de colores RGB, que por su código (0, 255, 0) se representa en color verde y por último, se establece el grosor de línea en el que se representa la circunferencia, 2. Esta operación lógica es fundamental en el programa ya que representa los datos referentes al radio del frente de llama que es lo que se busca conseguir en esta parte del código y poder tener un mayor acceso a información sobre el comportamiento del proceso de combustión que depende numerosos factores como dosado, temperatura y presión. El análisis visual de las imágenes es fundamental para una comparación con los datos obtenidos de manera experimental.

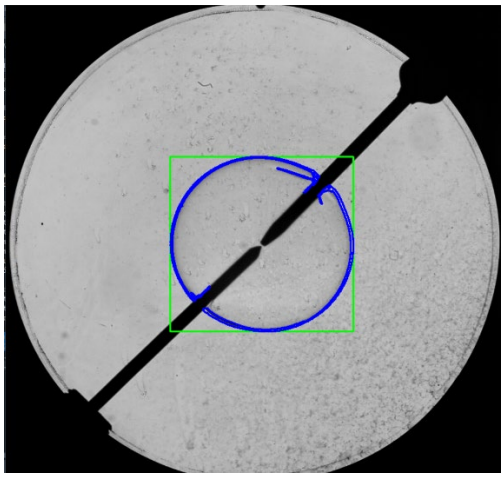


Figura 3.28: Imagen de los contornos detectados en el interior del rectángulo calculado de un fotograma para el cálculo del radio del frente de llama del experimento BC_E15.

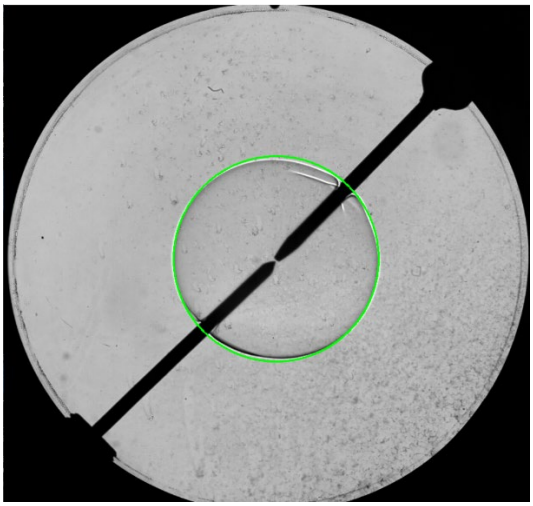


Figura 3.29: Imagen del radio del frente de llama calculado a partir de los contornos detectados de un fotograma del experimento BC_E15.

3.3.7. Recopilación de los datos del programa

En esta sección se desarrollan las funciones de los comandos que se han usado sobre el programa para obtener la información referente al radio de la cámara de combustión. Esos datos se almacenan en una serie de variables que son tratadas para conseguir uno de los dos principales objetivos con el que se ha desarrollado este proyecto.

```
149         # Comparar radio con el de la camara de combustion y cálculo de la variable tiempo
150         tiempo = (contador_fotogramas - fotograma_inicial) * (1000/fps)
151         print ("El tiempo en este instante es {}" .format(tiempo))
```

Figura 3.30: Cálculo de la variable tiempo como dato.

Como se muestra en la *Figura 3.30* en la línea de código 150 aparece definida la variable “tiempo”. Esta variable es fundamental ya que es uno de los dos datos

encargados de poder graficar la solución respecto al cálculo de la velocidad del frente de llama y posteriormente se recogen dentro del Excel. La variable “tiempo” recoge exactamente el tiempo que ha transcurridos desde el comienzo del video hasta el momento exacto en el que se está evaluando el fotograma que se está analizando. Permite analizar cómo evoluciona el radio del frente de llama a lo largo del tiempo. Se explicará más adelante cómo se grafica este dato sobre el eje de las abscisas. Para poder obtener este dato, se ha empleado una operación que consiste en realizar la resta entre la variable “contador_fotogramas” explicada en el apartado 3.3.2. en la línea de código 117 de la *Figura 3.20*, donde se observa que hace referencia al fotograma que se está analizando, y se le suma una unidad una vez se haya acabado el análisis, y se le resta la variable “fotograma_inicial”, explicada en la línea de código 54 de la *Figura 3.8* del apartado 3.2.1. que almacena únicamente el contador que hace referencia a la posición del primer fotograma, el que se ha empleado para realizar el cálculo del radio de la cámara de combustión. Este último contador no se modifica a medida que transcurre el video. Esta resta de contadores se multiplica por el tiempo que tarda en transcurrir un fotograma en milisegundos, de ahí la equivalencia de un 1 segundo en milisegundos (1000), entre la variable “fps”, explicada en la línea de código 29 de la *Figura 3.4* del apartado 3.1.3., que almacena el dato referente a la velocidad a la que se reproduce el video, para este programa va a ser siempre dependiente del formato de video.

Una vez realizada la operación anterior, se procede a imprimir por la consola el resultado de dicha operación. Es un bucle que se reproduce desde que se analiza el primer fotograma hasta que se reproduce el último. Para poder visualizar en la consola este dato, se hace uso de la función “print()” indicando en su interior el tipo de dato que se va a mostrar por pantalla mediante el algoritmo “.format()”, y en su interior la variable que se desea mostrar. Este algoritmo permite insertar valores de variables dentro de un texto definido por el usuario, en este caso en concreto inserta al final de la frase definida en la línea de código 151 de la *Figura 3.30*, el resultado obtenido en la variable “tiempo”.

```
153         if int(radio)/calibracion <= 5.7:
154             # Comparar radio con el de la camara de combustion
155             radio_video = int(radio) / calibracion
156             radio_dato = radio_video * 2
157             print("El radio (con el radio medio) del frente de llama es de {}" .format(radio_dato))
```

Figura 3.31: Bucle planteado para la obtención de radio del frente de llama.

Las líneas de código que se muestran en la *Figura 3.31*, comparamos el dato obtenido por medio de la detección de contornos de la imagen con el radio real de la cámara de combustión, indicando que, si el radio calculado a partir del video es menor que el valor del radio de la cámara de combustión siendo el valor de esta de 5,7 cm, realiza ejecuta las acciones de cálculo definidas dentro del bucle “if”. Si este radio supera el valor del radio de la cámara de combustión, no entra dentro del bucle y no guarda los datos para poder usarlos posteriormente.

Salvo en el fotograma de comienzo y en los fotogramas finales, puesto que la combustión ya llegado a las paredes de la cámara de combustión y el frente de llama se comienza a desvanecer y esos datos no son tan fiables como los recogidos durante el resto del proceso donde el frente de llama es más claro, el resto de datos que se generan a lo largo del análisis del video cumplen la condición planteada para poder ser procesados dentro del bucle "if" lo que permite que se compare el radio obtenido con la variable "calibración" explicada en el apartado 3.2.3 en la línea de código 79 de la *Figura 3.14*, que recoge el dato referente a la comparación del radio de la cámara de combustión con la medida real del radio de dicha cámara. El cálculo del radio del frente de llama que se obtiene a partir de los contornos detectados en la imagen se divide entre la variable "calibracion" para poder obtener el valor real del frente en cada fotograma de video, es decir, el valor del radio en cm para cada instante. Este dato se debe ser un número entero para poderlo comparar con la variable "calibracion", por lo que se usa el algoritmo "int()" para convertir el valor. El resultado de la división planteada se recoge en la variable "radio_video".

En la línea de código que sigue se imprime por la consola del programa el valor contenido en la variable "radio_video" precedido de un mensaje que hace referencia a dicho dato. Para poder realizar este proceso es necesario emplear la función "print()" y utilizar el algoritmo ".format()" para poder insertar el resultado que contiene dicha variable en un texto. Como se habían reducido las imágenes a la mitad del ancho y a la mitad del alto del video original, toda la información que se ha ido almacenando en la variable "radio_video" es la mitad de lo original, por lo que en la variable "radio_dato" se multiplica por dos el dato calculado en la variable "radio_video" que es la que se va a graficar posteriormente.

Las fórmulas empleadas para calcular los datos del "radio_video" y "tiempo" son las mismas que están planteadas en el programa de Matlab [1] utilizado antes para poder realizar el análisis de imágenes. Dicho programa realizaba de manera correcta la obtención de dichos datos, por lo que se ha decidido implementar estas fórmulas en el desarrollo del nuevo programa utilizando la lógica que emplea el lenguaje de programación Python.

3.3.8. Guardar los datos en listas

Tras obtener los datos fundamentales de este proyecto relacionados con el radio del frente de llama y con el tiempo en el que transcurre dicha evolución, se deben guardar estos datos en una lista de variables ya que se van obteniendo a medida que se analiza un fotograma y el dato cambia para el siguiente fotograma.

```
159             # Guardar los en cada instante de los datos recogidos por el programa
160             x_tiempo_radio.append(tiempo)
161             y_radio.append(radio_dato)
```

Figura 3.32: Metodología empleada para almacenar los datos.

Para poder almacenar los datos en las listas de datos declaradas fuera del bucle "while" como se muestran en las líneas de código 101, lista "x_tiempo_radio = []", y 102, lista "y_radio = []", de la *Figura 3.19* perteneciente al apartado 3.3.1., se utilizan los algoritmos que se muestran en la *Figura 3.32*. El algoritmo ".append()" que sigue al nombre de la lista donde va almacenar los datos, permite agregar un elemento al final de una lista existente. Dentro del algoritmo ".append()" se asigna el dato que se desea almacenar dentro de cada lista, en concreto en la lista "x_tiempo_radio = []" se desea almacenar la variable "tiempo" mientras que en la lista "y_radio = []" se almacena la variable "radio_dato". Estas operaciones están destinadas a recopilar los datos generados por programa para posteriormente analizarlos y generar gráficos con ellos.

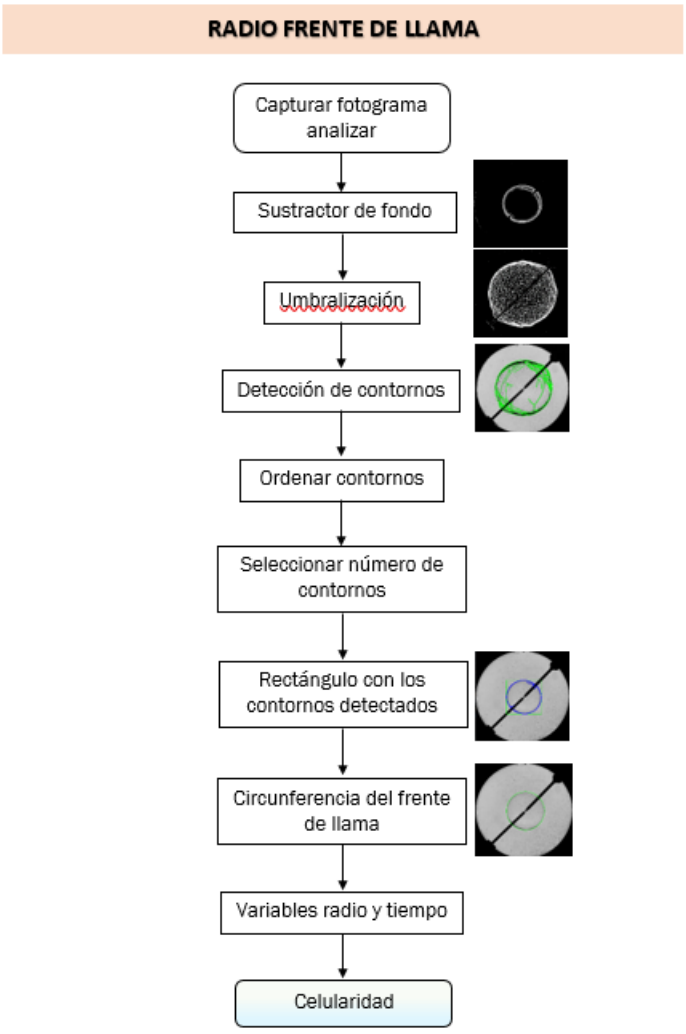


Figura 3.33: Diagrama de la de flujo de la detección y cálculo de los parámetros para obtener el radio del frente de llama.

3.4. Cálculo de la celularidad

Una vez calculados los parámetros y variables que permiten obtener el radio del frente de llama, se ejecuta el apartado donde se calcula la celularidad indicando el procedimiento seguido para ello en cada fotograma.

Los procesos seguidos para calcular la celularidad y posteriormente la variable "densidad_celular" que es la que contiene la información necesaria, son similares a los explicados anteriormente en el análisis del primer fotograma para obtener el radio de la cámara de combustión y el radio del frente de llama. Para poder calcular estos datos es necesario partir del radio calculado del frente de llama ya que en el interior de este radio es donde se genera la celularidad que interesa evaluar. Para que se pueda contabilizar de manera correcta a lo largo del tiempo y para cada fotograma, es necesario introducir todos las acciones, algoritmos y operaciones que se deben considerar dentro del bucle "while" que permite realizar los cálculos para cada fotograma y pasar al siguiente. Además, los comandos también deben de estar dentro de la condición "if not pausa:" definida en el apartado 3.3.1. que ejecuta las operaciones siempre que no se haya presionado la barra espaciadora del teclado para pausar el video.

3.4.1. Creación de una máscara circular

Antes de detectar la celularidad, es importante definir en que región del fotograma se va a detectar la celularidad si la hubiera. Para ello es importante saber que la región de interés para detectar la celularidad es la que se encuentra en el interior del radio del frente de llama que se ha definido en el apartado 3.3.

```
176         # CELULARIDAD
177         # Definir la región de interés (ROI) centrada en el centro del rectángulo mínimo
178         # Crear una máscara circular
179         mascara = np.zeros(fotograma.shape[:2], dtype=np.uint8)
180         # Esta fijado el centro con el obtenido previamente en la cámara de combustión
181         cv2.circle(mascara, (int(centro_rectangulo[0]), int(centro_rectangulo[1])), int(radio_celularidad), 255, -1)
182         cv2.circle(fotograma, (int(centro_rectangulo[0]), int(centro_rectangulo[1])), int(radio_celularidad), (0, 255, 255), 2)
```

Figura 3.34: Definición de la región de interés sobre la que se va a buscar la celularidad.

En la línea 179 de la *Figura 3.34*, se define la máscara circular. Para definir esta región usamos la función "np.zeros()", perteneciente a la librería Numpy, que crea una matriz de ceros con las dimensiones específicas, en el caso de este proyecto van a corresponder con el alto y ancho del fotograma del video, que se va a utilizar como máscara. A la matriz de ceros se le pasa el fotograma sobre el que se va a trabajar en cada instante de tiempo seguido del atributo ".shape[:2]" indicando que quiere una matriz con el alto y ancho del fotograma que se está analizando, esto permite asegurar que la máscara va a tener las mismas dimensiones que el fotograma. Seguido de este atributo, se le añade otro parámetro que especifica el tipo de elementos que almacena la máscara generada. Indicando "dtype = np.uint8" significa que cada elemento de la matriz será un número entero sin signo de 8 bits.

La matriz generada se almacena en una variable denominada “mascara” con la que se trabajará más adelante.

Tras definir la máscara de interés, utiliza la función “cv2.circle()”, explicada anteriormente, de la librería OpenCV para dibujar el círculo sobre la máscara generada anteriormente y dibujar ese mismo circulo, pero sobre la imagen original. La primera circunferencia se genera sobre la máscara generada en el punto anterior, para ello es necesario pasarle a la función “cv2.circle()” la variable que tiene almacenados los datos de la máscara que se genera, es decir, la variable “mascara”, el centro de la circunferencia que se va a generar, siendo en este caso particular el mismo centro del cual parte el radio del centro de llama, es decir, las coordenadas del centro del rectángulo explicadas en el apartado 3.3.5. Además, se debe de pasar un radio, el radio de interés para detectar la celularidad corresponde con el dato almacenado en la variable “radio_celularidad” explicada en la línea 144 de la *Figura 3.26* en el apartado 3.3.5., siendo este radio el 80% del radio del frente de llama detectado se utiliza la función “int()” para asegurar que el datos corresponda con un número entero. Seguido de ello, se asigna un valor a los pixeles de dentro del círculo, establecido en el máximo de 255 para que se activen todos ellos. Por último, se debe definir el grosor del círculo que es “-1” para que el circulo se rellene y los pixeles del interior puedan establecer su valor máximo.

De un modo similar se utiliza la función “cv2.circle()” de la línea 182 de la *Figura 3.34* para generar el circulo de interés, pero sobre el fotograma que se está analizando, para ello es necesario pasarle a la función la variable “fotograma”, el centro de la circunferencia que se va a generar y el radio, el mismo centro y radio que para el de la máscara. Se asigna un color al círculo, para este caso (0, 255, 255) define el color amarillo y un grosor de 2 pixeles para los bordes del círculo.

3.4.2. Procesamiento del fotograma

Al igual que sucedía para el análisis y obtención del radio del frente de llama, es necesario aplicar sobre el fotograma una serie de parámetros que permiten generar datos que aportan la información relevante.

```
184         # Aplicar la máscara al fotograma copiado para obtener solo los píxeles dentro de la circunferencia
185         roi = cv2.bitwise_and(copia, copia, mascara=mascara)
186
187         # Aplicar el sustractor de fondo y umbralización solo a la región de interés
188         mascara_roi = fgbg.apply(roi)
189
190         # Aplicar un umbral para obtener solo los píxeles más brillantes en la imagen
191         umbralizacion_roi = cv2.threshold(mascara_roi, 127, 255, cv2.THRESH_BINARY)[1]
```

Figura 3.35: Funciones para el procesamiento del fotograma para el cálculo de la celularidad.

En la línea 185 de la *Figura 3.35*, se realiza la función “cv2.bitwise_and()” de la librería OpenCV que consiste en realizar una operación a nivel de pixeles entre dos imágenes al que se le aplica el operador lógico “and” para conseguir que solo estén

activos los pixeles de la región de interés, en el caso de este proyecto solo los pixeles almacenados en la variable “mascara”. Para que la función se pueda ejecutar correctamente, se le debe de pasar una copia del fotograma, almacenada en la variable “copia” y la máscara definida anteriormente almacenada en la variable “mascara”, siendo esta la región de interés de estudio. Se obtiene almacena todo en una variable denominada “roi” que contiene la región de interés definida por la máscara sobre el fotograma copiado.

Tras ello se sustrae el fondo de la imagen obtenida en la línea 185 y definida en la variable “roi”. Para ello se usa la función “fgbg.apply()” que permite sustraer el fondo de la imagen dejando únicamente la máscara en primer plano de la misma. Se aplica sobre la región de interés que es la máscara circular generada a partir del 80% del radio del frente de llama. Se almacena todo en la variable “mascara_roi”.

Sobre la máscara se aplica la función “cv2.threshold()” para umbralizar la imagen. Para ello se le debe pasar como primer elemento la imagen que se desea umbralizar, que en este caso es la máscara, variable “mascara_roi”, se le aplica el mismo umbral que cuando se aplicó esta función para la detección del radio del frente de llama, un umbral mínimo de 127 para que todos los pixeles que estén por encima de este valor resalten y se puedan evaluar, y un umbral máximo que es de 255. El tipo de umbralización que se aplica sobre la variable es “cv2.THRESH_BINARY” que indica que todos los datos que se encuentre entre el rango de umbralización establecido de [127, 255] se les asigna el valor máximo de 255 para que sean los más brillantes y el resto se establecen en 0, correspondiente a un color negro. La imagen umbralizada se guarda en la variable “umbralizacion_roi” que solo almacena los pixeles más brillantes que estén activos dentro de la región de interés.

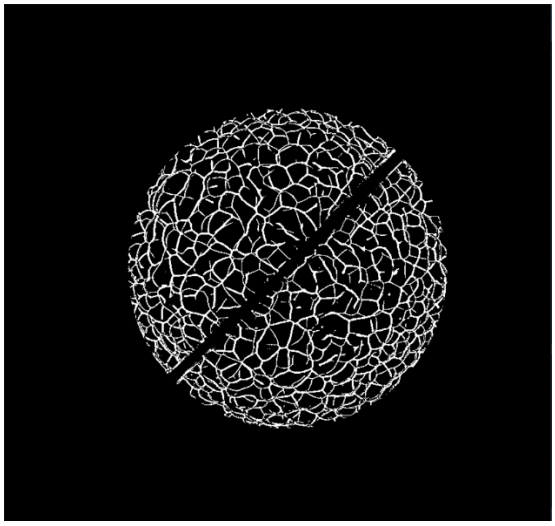


Figura 3.36: Imagen umbralizada de un fotograma para el cálculo de la celularidad para experimento BC_E15.

3.4.3. Operaciones de análisis de contornos en la región de interés

Una vez se tiene la región de interés umbralizada, se procede a detectar sobre ella contornos que definen si sobre esa región, correspondiente al frente de llama, se ha producido celularidad o no para ese fotograma. Para ello se deben de aplicar una serie de funciones específicas que acaban de definir los contornos de manera más precisa.

```
193         # Aplicar la transformación de adelgazamiento y encontrar los contornos en la imagen binaria resultante
194         thin_roi = cv2.ximgproc.thinning(umbralizacion_roi)
195         contornos_roi, hierarchy = cv2.findContours(thin_roi, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
196         # Dibujar los contornos encontrados en la imagen copiada (copia)
197         cv2.drawContours(fotograma, contornos_roi, -1, (0, 255, 0), 2)
```

Figura 3.37: Operaciones que se aplican sobre la región en la que se desean detectar la celularidad.

Sobre la imagen umbralizada almacenada en la variable “umbralizacion_roi” se le aplica la transformada de adelgazamiento que es un proceso que consiste en reducir el grosor de los objetos que se encuentran en una imagen binarizada manteniendo la conexión con los elementos de alrededor y manteniendo la forma. Consiste en representar el objeto de la manera más eficiente con la menor cantidad de pixeles para que facilite su posterior procesamiento. Buscamos reducir los elementos a formas más delgadas para que no consuma tanto del programa puesto que es un análisis muy profundo el que se realiza en este proyecto y tiene mucha carga computacional. La función que se encarga de transformar los pixeles a elementos más sencillos es el algoritmo de la librería OpenCV definido como “cv2.ximgproc.thinning()” [20]al cual es necesario introducirle la imagen que se desea analizar que en este caso está almacenada en la variable “umbralizacion_roi” que contiene los pixeles de ha simplificar resaltados en la región de interés. Los datos transformados se almacenan en la variable “thin_roi”.

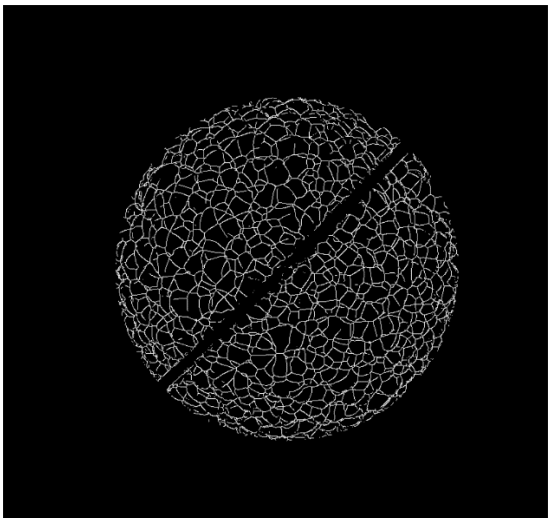


Figura 3.38: Imagen aplicada la transformada de adelgazamiento de un fotograma para el cálculo de la celularidad para experimento BC_E15.

Con los pixeles más simplificados se procede a detectar los contornos sobre la imagen almacenada en la variable “thin_roi” como se muestra en la línea del código 195 de la *Figura 3.37*. Para ello se aplica la función “cv2.findContours()” de la librería OpenCV que se encarga de realizar las operaciones internas para detectar los contornos dentro de la imagen. Necesita la imagen que sobre la que se desean encontrar los contornos, datos almacenados en la variable “thin_roi”, el modo de detección de esos contornos que se realiza por medio del algoritmo “cv2.RETR_TREE” que detecta los contornos en la imagen, en este caso solo devuelve los contornos detectados en la región de interés y reconstruye la jerarquía de los mismos una vez detectados. Seguido de este algoritmo se plantea el algoritmo que almacena solo los datos esenciales para poder reconstruir los contornos posteriormente, para ello se usa el algoritmo “cv2.CHAIN_APPROX_SIMPLE” que de los datos que se han detectado los transforma en segmentos de líneas verticales, horizontales y diagonales representando almacenado únicamente sus puntos de inicio y fin. Toda la información especificada se almacena en dos variables, en “contornos_roi” que contiene una lista donde se encuentran almacenados todos los contornos detectados y en la variable “hierarchy” que contiene información sobre la jerarquía de los contornos detectados. La última variable no va a ser útil en esta parte del estudio.

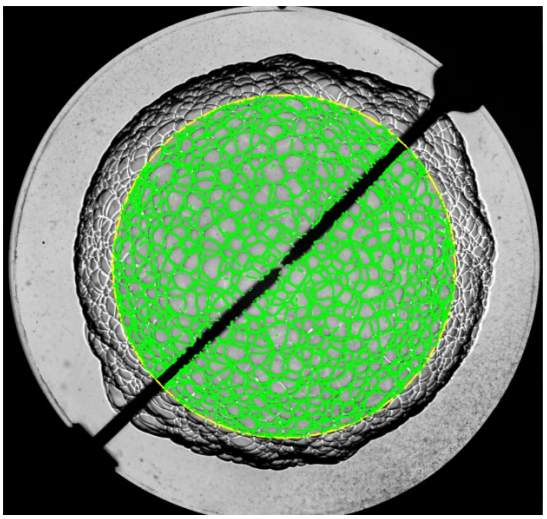


Figura 3.39: Imagen con los contornos detectados dentro del 80% del radio del frente de llama de un fotograma para el cálculo de la celularidad para experimento BC_E15.

La línea 197 de la *Figura 3.37*, explica la función que dibuja los contornos encontrados sobre la imagen de interés, en este caso la imagen de interés corresponde con el fotograma del video que se está detectando y está almacenado en la variable “fotograma”. Se aplica la función “cv2.drawCotours()” para dibujar los contornos. Es necesario asígnale una imagen para que los dibuje, “fotograma”, definir que elemento tiene que dibujar sobre esa imagen, los datos referentes a los contornos almacenados en la lista “contornos_roi”, indicar cuantos contornos de esa lista se desea dibujar, como se pretenden dibujar todos los contornos que tenga almacenados la lista se asigna el valor “-1”, el color en que se desean dibujar esos

contornos, en este caso se ha seleccionado el color verde por lo que se asigna (0, 255, 0), y por último, se define el grosor con el que se van a dibujar esos contornos, para este caso el grosor seleccionado es “2”.

3.4.4. Cálculo de los parámetros necesarios para la detección de la celularidad

Una vez se tienen todos los encontrados todos los contornos, se procede a calcular todas las variables que llevan la información necesaria para representar en los gráficos.

```
199         densidad_celular = len(contornos_roi) / (np.pi *((int(radio_celularidad)/calibracion)**2))
200         densidad_celular = densidad_celular * 2
201         contornos_color = cv2.cvtColor(thin_roi, cv2.COLOR_GRAY2BGR)
202         print("Cantidad de elementos {}".format(densidad_celular))
```

Figura 3.40: Variables donde se almacenan los datos de celularidad para cada fotograma del video.

La variable “densidad_celular” que aparece en la línea de código 199 de la *Figura 3.40,* es una variable adimensional que almacena los datos calculados a partir de los contornos detectados para cada fotograma del video. Para adimensionalizar la variable se deben de tomar la longitud de la matriz, por medio de la función “len()” donde están almacenados los contornos que se han detectado sobre el fotograma, “contornos_roi”, ya que si hay pocos contornos esta lista es pequeña, mientras que si hay mucha celularidad la lista tendrá un alto número de datos almacenados. La longitud de la matriz se divide entre el área de la circunferencia que ocuparía realmente la combustión en la instalación experimental, por ello se divide la variable que contiene el radio del frente de llama al 80%, “radio_celularidad” entre la variable “calibración” definida en la *Figura 3.15* del apartado 3.2.3. calculada a partir del radio de la cámara de combustión. Para poder obtener la variable “densidad_celular” todos los datos deben ser enteros, por lo que se usa la función “int()”. Esto permite estimar la densidad celular en la región de interés definida con anterioridad.

Como al principio se reescalaron a la mitad de su ancho y alto los fotogramas para que por pantalla se puedan visualizar sin problemas, los datos no son los reales, sino que son la mitad de los reales, por lo que en la variable “densidad_celular” de la línea 200 de la *Figura 3.40* se multiplican por dos los datos calculados en la línea 199, siendo los de la línea 200 los reales correspondientes al video original.

Para que los contornos se puedan visualizar por pantalla a color, como se ha definido en la línea 197 de la *Figura 3.40,* se deben de transformar de escala de grises a color por medio de la función de la librería OpenCV denominada “cv2.cvtColor()” donde se pasa la lista de contornos almacenada en la variable “thin_roi” y se establece el formato al cual se quien pasar los colores usando para ello la función, también de la librería OpenCV, “cv2.COLOR_GRAY2BGR” para pasar de escala de grises a código BGR.

Por último, se muestra por pantalla por la consola el dato almacenado en la variable "densidad_celular" indicando en cada fotograma el número de contornos que ha encontrado. Para mostrarlo por la consola se usa la función "print()".

3.4.5. Guardar los datos en listas

Tras obtener los datos que proporcionan información sobre la celularidad y con el tiempo en el que transcurre dicha evolución, se deben guardar estos datos en una lista de variables ya que se van obteniendo a medida que se analiza un fotograma y el dato cambia para el siguiente fotograma.

```
208         # Guardamos en unas listas los datos recogidos para cada fotograma
209         x_tiempo_celularidad.append(tiempo)
210         y_celularidad.append(densidad_celular)
```

Figura 3.41: Metodología empleada para almacenar los datos.

Para poder almacenar los datos en las listas de datos declaradas fuera del bucle "while" como se muestran en las líneas de código 103, lista "x_tiempo_celularidad = []", y 104, lista "y_celularidad = []", de la *Figura 3.19* perteneciente al apartado 3.3.1., se utilizan los algoritmos que se muestran en la *Figura 3.19*. El algoritmo ".append()" que sigue al nombre de la lista donde va almacenar los datos, permite agregar un elemento al final de una lista existente. Dentro del algoritmo ".append()" se asigna el dato que se desea almacenar dentro de cada lista, en concreto en la lista "x_tiempo_celularidad = []" se desea almacenar la variable "tiempo" que almacena los mismos datos que la lista "x_tiempo_radio = []" pero que al representar posteriormente los gráficos del radio frente al tiempo y de la densidad celular frente al tiempo, el programa no permite llamar a la misma lista la representar gráficos distintos, por lo que se almacenan los mismos datos pero en listas diferentes. La variable "tiempo" está definida en la línea 150 de la *Figura 3.30* del apartado 3.3.7. La lista "y_celularidad = []" se almacena la variable "densidad_celular". Estas operaciones están destinadas a recopilar los datos generados por programa para posteriormente analizarlos y generar gráficos con ellos.

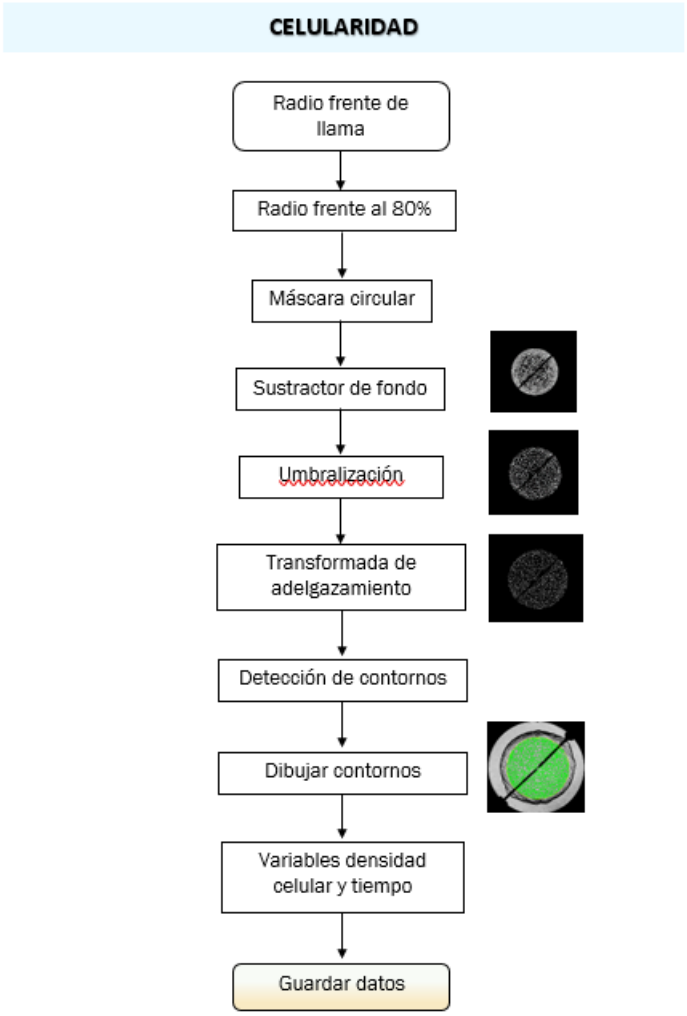


Figura 3.42: Diagrama de la de flujo de la detección y cálculo de los parámetros para obtener la celularidad.

3.5. Representación de gráficos y tabla de datos por pantalla

En este punto del código, se explica cómo se representan los datos obtenidos en las distintas variables referentes al cálculo del radio del frente de llama y al de la densidad celular en gráficas y se muestran por pantalla a la que se muestra el video con los resultados visuales de la evolución de los distintos parámetros. Además, también se visualiza por pantalla una tabla que recoge para cada fotograma el dato almacenado en las variables calculadas.

3.5.1. Representación gráfica de los datos obtenidos para el radio del frente de llama y el de la celularidad ambos frente al tiempo

Tras almacenar los datos en listas, se procede a usar esas listas para poder realizar una representación gráfica de los datos obtenidos, permitiendo obtener la relación del radio en función del tiempo que da como resultado de la evolución temporal de

la velocidad del frente de llama de la combustión y la densidad celular frente al tiempo.

```
163         plt.figure(1)
164         plt.plot(x_tiempo_radio, y_radio, color='red')
165         plt.title('Velocidad del frente de llama')
166         plt.xlabel('Tiempo (ms)')
167         plt.ylabel('Radio del frente de llama (cm)')
```

Figura 3.43: Desarrollo empleado para generar los gráficos que evalúa los resultados para el radio del frente de llama.

Se representan dos gráficos distintos que contienen información sobre el radio del frente de llama y sobre la densidad celular en cada fotograma, por lo que se deben de generar dos figuras gráficas diferentes, una para cada gráfico. El gráfico para el radio está definido en la línea 163 de la *Figura 3.43*.

Para poder generar los gráficos que muestran los datos la información recogida a lo largo del trabajo, se emplea la librería Matplotlib ya que permite generar gráficos a partir de los datos almacenados en dos listas, en concreto en las listas "x_tiempo_radio = []" y "y_radio = []" donde se almacenan los datos referentes al tiempo y al radio del frente de llama respectivamente. Para que se puedan introducir los datos a la vez que se analiza el fotograma, las líneas de código que se muestran en la *Figura 3.19*, se encuentran dentro de la condición establecida por el bucle "if" representado en la *Figura 3.19* y a su vez todo ello se encuentra dentro del bucle "while" que recoge todo el análisis del fotograma.

La generación del gráfico se realiza mediante la función "ptl.plot()", donde "ptl" es la abreviatura de la librería Matplotlib, y ".plot" es un algoritmo que permite la generación de gráficos. El primer dato hace referencia de las abscisas, x, asociado a la lista "x_tiempo_radio = []" que recoge los datos referentes a los datos generados en la variable "tiempo", mientras que el segundo dato hace referencia a las ordenadas, y, asociado a la lista "y_radio = []" que recoge los datos generados en la variable "radio_dato". Este gráfico traza una línea que conecta los puntos de manera automática. El gráfico se muestra por consola mostrando la evolución de los datos.

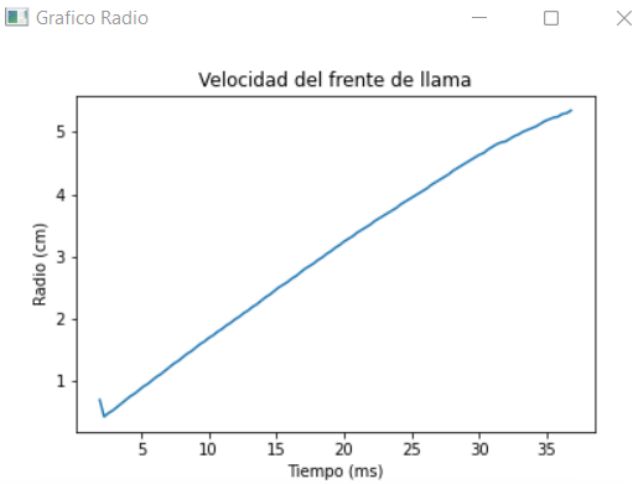


Figura 3.44: Imagen de un gráfico representando la evolución del radio del frente de llama frente al tiempo para el experimento BC_E15.

Para indicar a qué datos hace referencia cada eje y ponerle un título al gráfico donde se representan los datos, se agregan las líneas de código de la 165 a la 167 de la *Figura 3.43*. Para poner títulos se utiliza el algoritmo de la librería Matplotlib ".title" que establece el título del gráfico, este caso denominado como *Velocidad del frente de llama*. Para referenciar a los ejes de ordenadas, se establecen etiquetas mediante el algoritmo ".xlabel" e ".ylabel", ambos de la librería Matplolib. La primera de las etiquetas hace referencia al eje de las abscisas indicando como título *Tiempo (ms)* mientras que la segunda de las etiquetas hace referencia al eje de las ordenadas indicando como título *Radio (cm)*.

```
212     plt.figure(2)
213     plt.plot(x_tiempo_celularidad, y_celularidad, color ='red')
214     plt.title('Crecimiento de la celularidad')
215     plt.xlabel('Tiempo (ms)')
216     plt.ylabel('Densidad celularidad')
```

Figura 3.45: Desarrollo empleado para generar los gráficos que evalúa los resultados para la celularidad.

De un modo muy similar se ejecutan los comandos para obtener el gráfico que contiene la información que compara la densidad celular frente al tiempo, definido en la línea 212 de la *Figura 3.45*.

Para la densidad celular los datos que se deben de representar están almacenados en las listas "x_tiempo_celularidad = []" para el eje de abscisas y para el eje de ordenadas los datos almacenados en la lista "y_celularidad = []" que almacena los datos referentes a la variable "densidad_celular". Esta gráfica se representa en color rojo. Las etiquetas de los ejes y el título del gráfico también cambian como se muestra en las líneas 214 a la 216 de la *Figura 3.45*.

Figura 3.46: Imagen de un gráfico representando la evolución de la celularidad frente al tiempo para el experimento BC_E15.

Estos comandos solo funcionan si las gráficas se muestran por la consola, por lo que para mostrarlas en una pantalla emergente es necesario transformar un poco los dos gráficos.

```
169         grafico_radio = plt.gcf()
170         grafico_radio.canvas.draw()
171         grafico_radio =np.array(grafico_radio.canvas.renderer.buffer_rgba())
```

Figura 3.47: Algoritmos para adaptar el gráfico del radio del frente de llama a una ventana emergente.

```
218         grafico_celularidad = plt.gcf()
219         grafico_celularidad.canvas.draw()
220         grafico_celularidad = np.array(grafico_celularidad.canvas.renderer.buffer_rgba())
```

Figura 3.48: Algoritmos para adaptar el gráfico de densidad celular a una ventana emergente.

Ambos gráficos se deben de adaptar de manera independiente para poder visualizarlos en la pantalla emergente pero las funciones son las mismas para ambos.

La función “plt.gcf()” de la librería Matplotlib, permite obtener la figura actual almacenando en una variable la información gráfica en la que se están realizando los trazados. Se almacenan variables denominadas “grafico_radio” para la gráfica del radio del frente de llama y “grafico_celularidad” para el gráfico de densidad celular. Estas variables van a ir sobrescribiendo sus datos al aplicar otras funciones de adaptación.

A las variables se les aplica la propiedad “canva” que permite representar los trazos de las gráficas almacenadas en un lienzo de dibujo. Para dibujar los trazos sobre los lienzos se utiliza el algoritmo “draw() para dibujar los datos.

Por último, se aplica la función “np.array()” para transforma la información buscada en una matriz de pixeles que se pueda mostrar por pantalla como una imagen en una ventana emergente. Para que la función pueda actuar, es necesario acceder al renderizado que contiene la librería Matplotlib para generar figuras por medio de la función “.canvas.renderer.buffer_rgba()”. Esto se aplica a ambas variables “grafico_radio” y “grafico_celularidad”.

Para evitar que haya muchas pantallas emergentes difíciles de gestionar por la pantalla, se ha optado por unificar pantallas en función de los datos que representa, es decir, se han unido los gráficos en una sola ventana y se han unidos los videos en un solo gráfico.

```
225         # Unir los dos gráficos en una sola ventana y representarlos
226         grafico_completo = cv2.hconcat([grafico_radio, grafico_celularidad])
227         videos_representacion = cv2.hconcat([fotograma, contornos_color])
228
229         cv2.imshow('Grafico completo', grafico_completo)
230         cv2.imshow('Videos unidos', videos_representacion)
231         #cv2.imshow('Pantalla completa', conjunto_ventanas)
```

Figura 3.49: Algoritmos representar sobre una misma ventana emergente los gráficos y en otra los fotogramas. Comandos para que se muestren por pantalla.

La función de la librería OpenCV denominada "cv2.hconcat()" que aparece en las líneas 226 y 227 de la *Figura 3.49,* permite concatenar imágenes de manera horizontal. En la línea 226, se busca concatenar los dos gráficos con los datos del radio del frente de llama y los datos de celularidad, para ello se introducen las dos imágenes en pixeles de los gráficos, "grafico_radio" y "grafico_celularidad", para obtener todo en una sola imagen resultante de concatenar y se almacena en la variable "grafico_completo". En la línea 227, se busca concatenar la variable "fotograma" sobre la que se representan los datos del radio del frente de llama y la variable "contorno_color" donde se representan los contornos detectados en el frente de llama. Se unen ambas imágenes y se almacenan en la variable "videos_representación".

Por último, por medio de la función "cv2.imshow()" se muestran en las ventanas emergentes, cada una con su correspondiente título indicando lo que se representa, las variables con las imágenes concatenadas.

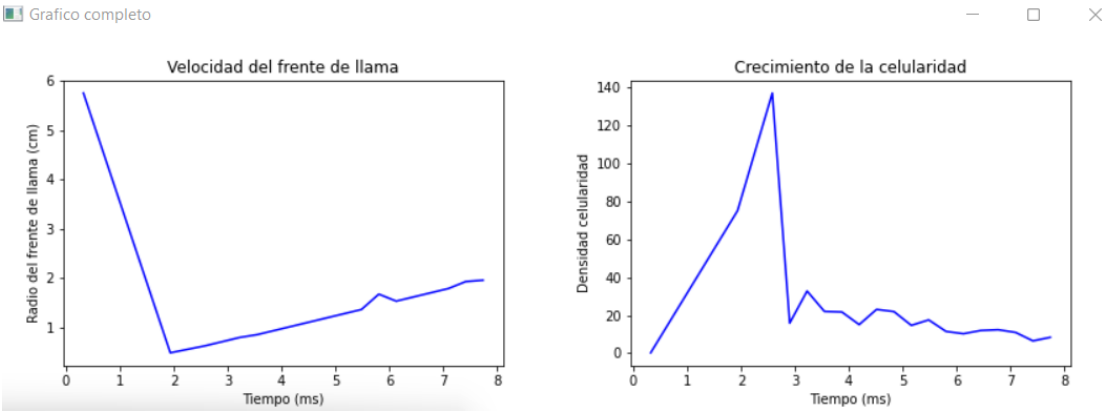


Figura 3.50: Imagen de los gráficos concatenados para la representación en pantalla para el experimento BC_E15.

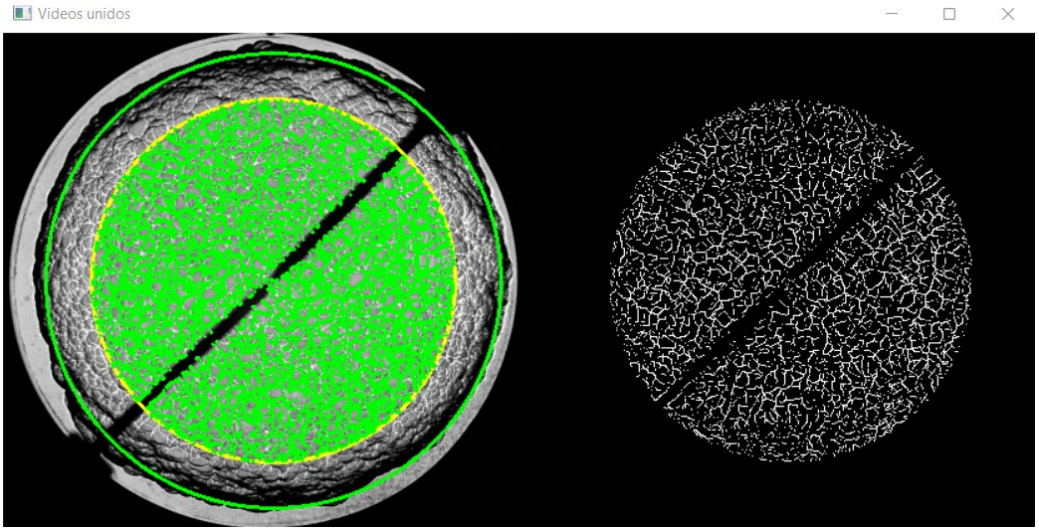


Figura 3.51: Imagen de los fotogramas concatenados para la representación en pantalla para el experimento BC_E15.

3.5.2. Tabla de datos de las variables para cada fotograma

Al igual que se muestran por pantalla las imágenes concatenadas de los datos que se están obteniendo para cada fotograma, es importante mostrar por pantalla el valor de esos datos que se están obtenido en cada instante. Para ello se genera una tabla que recoge todos los datos de las variables importantes y se actualiza para cada fotograma.

```
242     # Crear una imagen en blanco
243     ancho, alto = 180, 470
244     imagen = np.zeros((ancho, alto, 3), np.uint8)
245     imagen.fill(255)
246
247     # Posicion de la tabla y tamaño
248     a, b = 30, 30
```

Figura 3.52: Comandos para crear una imagen en blanco y variables para posicionar la tabla dentro de la imagen.

Se busca crear una imagen en blanco sobre la que se representen los datos de las variables que se van modificado para cada fotograma. Estos comandos están fuera del bucle “if” definido por el número de contornos asignados por el usuario a la variable “modificacion_area” como se ha analizado en la línea 133 de la *Figura 3.25* del apartado 3.3.4.

En las variables “ancho” y “alto” de la línea 243 de la *Figura 3.52*, se asigna por el usuario los valores que se desean que tenga la imagen en pixeles. En la siguiente línea, se utiliza la función “np.zeros()” para crear una matriz de ceros dimensionándola con las variables “ancho” y “alto”, seguido de la especificación del número de canales de color que se desea que tenga la imagen, en este caso los 3 (RGB). La función “np.uint8” especifica el tipo de datos que se van a mostrar sobre

esa imagen que son números enteros. Con la función ".fill()" se le proporciona color a la imagen generada, que al asignar el máximo valor de 255, la imagen va a tener color blanco.

En la línea 248, se definen dos variables "a" y "b" con los datos deseados por el usuario. Estos valores representan las coordenadas de la esquina superior izquierda de la tabla que se va a generar dentro de la imagen.

```
250     cv2.putText(imagen, "FOTOGRAMA", (a+5, b+20), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 1)
251     cv2.putText(imagen, "TIEMPO", (a+5, b+50), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 1)
252     cv2.putText(imagen, "RADIO", (a+5, b+80), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 1)
253     cv2.putText(imagen, "DENSIDAD CELULARIDAD", (a+5, b+110), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 1)
254     cv2.putText(imagen, str(contador_fotogramas), (a+200, b+20), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 1)
255     cv2.putText(imagen, str(tiempo), (a+200, b+50), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 1)
256     cv2.putText(imagen, str(radio_dato), (a+200, b+80), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 1)
257     cv2.putText(imagen, str(densidad_celular), (a+200, b+110), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 1)
```

Figura 3.53: Definición de las posiciones que se les asignan a cada variable dentro de la tabla generada en la imagen en blanco.

La función de la librería OpenCV, "cv2.putText()" permite agregar texto a la imagen que se ha generado. Para ello se debe de proporcionar a la función, la variable "imagen" que tiene almacenada la imagen en blanca sobre la que se van a escribir los resultados, seguida de la etiqueta o dato que se desea representar en el interior de la tabla. Si son etiquetas, se van asignar el *FOTOGRAMA*, *TIEMPO*, *RADIO* y *DENSIDAD CELULAR*, mientras que si nos referimos a datos se deben de convertir en cadenas de texto por medio del algoritmo "str()" para poder representarlos sobre la variable "imagen". Los datos que se van a representar son lo que están almacenados en las variables "contador_fotogramas", "tiempo", "radio_dato" y "densidad_celular". Seguido de estos parámetros se pasan las coordenadas con respecto a las variables "a" y "b" definidas en la línea 248 de la *Figura 3.53* sobre la que se suman o se mantienen pixeles dependiendo de donde se desee colocar la etiqueta o el dato sobre la imagen. Los últimos parámetros hacen referencia al tipo de fuente con el que se van a mostrar los datos, el tamaño de esa fuente, el color del texto que es (0, 0, 0) y se refiere al negro y el grosor del texto que es de "1" pixel.

```
259     cv2.imshow('Tabla de datos', imagen)
```

Figura 3.54: Algoritmo para mostrar por pantalla la imagen con la tabla de datos

Por último, se utiliza la función "cv2.imshow()" de la librería OpenCV para mostrar por pantalla la variable "imagen" que contiene la tabla con los datos referentes a cada fotograma. A la que se van generando nuevos datos para cada fotograma, la tabla se va actualizando por pantalla y si se pasar el programa, la tabla mostraría, de forma fija, los datos referentes al fotograma que se muestra por pantalla.

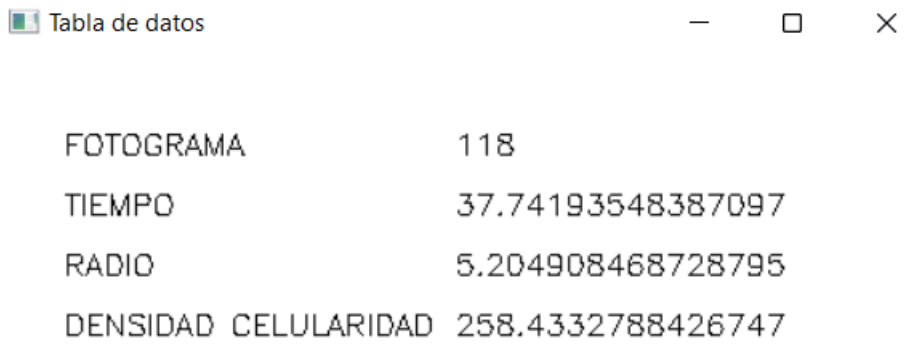


Figura 3.55: Imagen de la tabla con los datos calculados en los parámetros para la representación en pantalla de un fotograma para el experimento BC_E15.

3.6. Guardar datos y cierre del programa

Una vez realizados todas las operaciones necesarias para que el programa ejecute correctamente las acciones para las que ha sido diseñado, se procede a almacenar los datos recogidos a lo largo de todo el proceso y a cerrar el programa para que no continúe ejecutando ninguno de los comandos.

3.6.1. Procesamiento final del fotograma

Una vez ejecutados las operaciones necesarias y almacenados los datos, se debe de mostrar por pantalla los resultados, tanto del fotograma como del gráfico procesado.

```
262     # Escribir el fotograma procesado en el video de salida
263     out.write(fotograma)
264     print("Estoy en el fotograma {}" .format(contador_fotogramas))
265     # Mostrar las figuras
266     plt.show(block=False)
```

Figura 3.56: Procedimientos finales para el cierre del programa.

La parte del código que se explica a continuación, ya no se encuentra implementada dentro del bucle “if” pero sí se encuentra ejecutándose dentro del bucle infinito “while” y del “if not pausa :”.

Lo primero que se realiza en la línea de código 263 de la *Figura 3.56*, consiste en escribir el fotograma procesado en el video de salida. La variable “out” hace referencia al elemento generado en el apartado 3.1.5. como se muestra en la *Figura 3.6*. Este elemento se guarda en un archivo de video en la misma carpeta donde se encuentra guardado el video. Para poder escribir este fotograma procesado se utiliza el algoritmo “.write()” y se hace referencia al elemento que se desea mostrar, en este caso el fotograma.

A su vez se muestra por pantalla, mediante la función “print()” el contador de fotogramas que indica la posición del fotograma que se está analizando. El procedimiento que se sigue es el mismo que se ha explicado con anterioridad en otros apartados.

Por último, la línea de código 266 de la *Figura 3.18*, muestra los gráficos generados con los datos recogidos en el apartado 3.3.8. y apartado 3.4.5. Esta línea de código lo único que se encarga de ejecutar es mostrar por la consola los gráficos generados.

3.6.2. Guardar los datos en un archivo de Excel

En este apartado se explicará el procedimiento que se sigue para poder almacenar los datos recogidos en las listas de datos en un archivo de Excel.

```
272     # Guardar los datos en un archivo de excel
273     # Primero generamos un dataframe
274     df = pd.DataFrame({"TIEMPO RADIO": x_tiempo_radio , "RADIO FRENTE DE LLAMA": y_radio , "TIEMPO CELULARIDAD": x_tiempo_celularidad , "CELULARIDAD": y_celularidad})
```

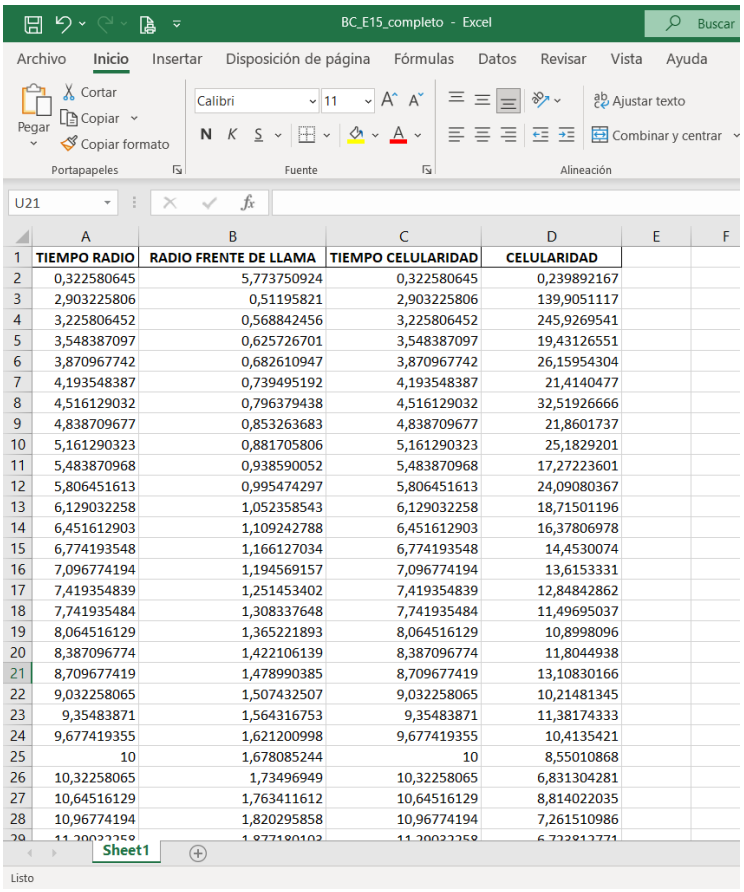
Figura 3.57: Creación del elemento que almacena los datos en un archivo de Excel.

Para poder guardar los datos en un archivo de Excel, es necesario recurrir a los algoritmos que dispone la librería Pandas explicada en el apartado 3.1.1.

Para poder guardar los datos, es necesario crear un elemento denominado “DataFrame” mediante la función “pd.DataFrame()”. Este elemento es una estructura de datos tabulares bidimensionales en la que se pueden introducir tanto datos como etiquetas que representan el tipo de elementos que se almacenan en el archivo. Este elemento se comporta de una manera similar a una hoja de cálculo o una tabla de base de datos.

El “DataFrame” que se está generando crea cuatro columnas donde en la primera de ellas, con la etiqueta *TIEMPO RADIO* registrada en la primera casilla identificando los datos que se almacenan en las filas sucesivas, se guardan los datos asociados a la lista “x_tiempo_radio = []”. En la segunda columna, se etiqueta con el nombre de *RADIO FRENTE DE LLAMA*, se almacenan los datos asociados a la lista de datos “y_radio = []”. Seguidas a estas dos columnas, se da la etiqueta *TIEMPO CELULARIDAD* que recoge los datos de la lista “x_tiempo_celularidad = []” y en la última columna, *CELULARIDAD* que almacena los datos de la variable “densidad_celular” almacenados en la lista “y_celularidad = []” como se muestra en la *Figura 3.19*.

Para que la ejecución se realice de manera correcta, se debe de pasar por orden primero la etiqueta de la columna, si la lleva, seguido de la lista que almacena los datos. Si se incluyen más columnas al Excel deben ir precedidas de una coma.



BC_E15_completo - Excel

Archivo Inicio Insertar Disposición de página Fórmulas Datos Revisar Vista Ayuda

Pegar Cortar Copiar Copiar formato Portapapeles

Calibri 11 Fuente

Ajustar texto Combinar y centrar Alineación

U21

	A	B	C	D	E	F
1	TIEMPO RADIO	RADIO FRENTE DE LLAMA	TIEMPO CELULARIDAD	CELULARIDAD		
2	0,322580645	5,773750924	0,322580645	0,239892167		
3	2,903225806	0,51195821	2,903225806	139,9051117		
4	3,225806452	0,568842456	3,225806452	245,9269541		
5	3,548387097	0,625726701	3,548387097	19,43126551		
6	3,870967742	0,682610947	3,870967742	26,15954304		
7	4,193548387	0,739495192	4,193548387	21,4140477		
8	4,516129032	0,796379438	4,516129032	32,51926666		
9	4,838709677	0,853263683	4,838709677	21,8601737		
10	5,161290323	0,881705806	5,161290323	25,1829201		
11	5,483870968	0,938590052	5,483870968	17,27223601		
12	5,806451613	0,995474297	5,806451613	24,09080367		
13	6,129032258	1,052358543	6,129032258	18,71501196		
14	6,451612903	1,109242788	6,451612903	16,37806978		
15	6,774193548	1,166127034	6,774193548	14,4530074		
16	7,096774194	1,194569157	7,096774194	13,6153331		
17	7,419354839	1,251453402	7,419354839	12,84842862		
18	7,741935484	1,308337648	7,741935484	11,49695037		
19	8,064516129	1,365221893	8,064516129	10,8998096		
20	8,387096774	1,422106139	8,387096774	11,8044938		
21	8,709677419	1,478990385	8,709677419	13,10830166		
22	9,032258065	1,507432507	9,032258065	10,21481345		
23	9,35483871	1,564316753	9,35483871	11,38174333		
24	9,677419355	1,621200998	9,677419355	10,4135421		
25	10	1,678085244	10	8,55010868		
26	10,32258065	1,73496949	10,32258065	6,831304281		
27	10,64516129	1,763411612	10,64516129	8,814022035		
28	10,96774194	1,820295858	10,96774194	7,261510986		

Sheet1

Listo

Figura 3.58: Imagen del Excel que se genera con los datos calculados para el experimento BC_E15.

3.6.3. Salida del bucle

Se diseña una última parte dentro del bucle “while” que permite al usuario para el programa en la ocasión que él crea oportuna dependiendo de las necesidades.

```
275     # Salir del bucle cuando el usuario presione la tecla 'Esc'
276     delay = int(1000/fps)
277
278 # Detectar la pulsación de la barra espaciadora para pausar o reanudar el video
279 key = cv2.waitKey(1)
280 if key == ord(' '):
281     pausa = not pausa
282 if cv2.waitKey(30) & 0xff == 27:
283     break
```

Figura 3.59: Procedimiento de ejecución de la salida del bucle por condiciones del usuario.

La última operación que ejecuta el programa es el cálculo del retardo necesario entre las iteraciones del bucle principal. Esta operación es necesaria dado que el algoritmo “.waitKey()” espera el tiempo específico en milisegundos, por lo que se dividen el dato almacenado en la variable “fps” referente a la velocidad de fotogramas, explicada en el apartado 3.1.3. en la línea de código 29 de la *Figura 3.35*, entre 1000 para transformarlo en milisegundos. Posteriormente, se transforma el resultado en

un número entero mediante el algoritmo “int()” y se almacena en una variable denominada “delay”.

A continuación, se realiza una comprobación mediante un bucle “if” que verifica si se cumple la condición que ejecuta la acción contenida en dicho bucle. La condición que debe de cumplir el programa es que la función de la librería OpenCV denominada “.waitkey()”, que espera que el usuario realice una entrada por el teclado. Esta función detiene la ejecución del programa y espera a que el usuario presione una tecla. El tiempo que espera para que el usuario ejecute la actividad por el teclado es de 30 milisegundos para cerrar el programa y 1 milisegundo para parar la ejecución del programa, si no se presiona ninguna tecla dentro de ese periodo de tiempo, el programa devuelve un valor que no corresponde con el de la condición por lo que continúa ejecutándose el programa.

Para parar el programa se debe de presionar por teclado la barra espaciadora. Si eso se produce, la variable “pausa” definida en el la línea 43 en la *Figura 3.6* del apartado 3.1.5., pasa de ser “False” a ser “True” indicando que ya no se puede ejecutar todo el código que se encuentra dentro del bucle “if not pausa:”, y el programa se para con el fotograma que se muestra en pantalla. Cuando se vuelve a presionar la tecla espaciadora del teclado, el programa continúa ejecutándose porque la variable “pausa” vuelve a valer “False”.

La expresión planteada en la ejecución del bucle planteada en la línea de código 282 de la *Figura 3.59*, obtiene el código ASCII de la tecla presionada durante el tiempo especificado. La operación lógica ejecutada se emplea para asegurar que solo se capturen los 8 bits significativos del resultado.

Si la condición se cumple porque se presiona la tecla *Esc* del teclado, correspondiente con el 27, se accede al código ASCII entrando en el bucle y ejecuta la acción “break haciendo que el programa salda del bucle “while” donde se realizan las operaciones, cerrando a su paso todas las ventanas emergentes y parando toda actividad ejecutada en la consola.

3.6.4. Guardar datos

Se van a almacenar en hoja de un archivo formato Excel, con extensión .xlsx, de datos distintos aquellos correspondientes al radio del frente de llama y a los datos obtenidos en el cálculo de la celularidad, puesto que se han realizado en programas diferentes, aunque las primeras acciones sean similares en ambos programas.

```
285    # Guardamos el dataframe en un archivo de excel
286    archivo_excel = nombre_video_sin_extension + "_completo.xlsx"
287    df.to_excel(archivo_excel, index = False)
```

Figura 3.60: Guardar datos en un archivo de Excel con los datos de tiempo, radio del frente de llama y densidad celular para cada fotograma.

Para que la acción se pueda ejecutar de manera correcta, es necesario que previamente se haya definido un "dataframe", como se muestra en la *Figura 3.60*, en la línea de código 286. En dicha línea se asocia cada una de las listas de variables, definidas previamente al comienzo de la programación de los apartados de cálculo del radio del frente de llama y del apartado de celularidad, a una columna correspondiente del archivo de Excel, en concreto se asocian por orden de aparición, como se ha desarrollado en apartados superiores.

Como se muestra en la *Figura 3.60*, en la línea 286, se genera una variable "archivo_excel" para poder definir el nombre del archivo de Excel asociado al video del experimento del cual está realizando los cálculos. Para ello toma como referencia la variable que almacena el nombre del video sin extensión, definida previamente en el apartado 3.6.2. Se establece la extensión referente al Excel, .xlsx al final de la línea 286.

Una vez definida esta variable, asocian los elementos al "dataframe" definido previamente. La acción que se realiza es la de convertir los datos almacenados en el "dataframe" en un Excel, haciendo uso del comando "to_excel" perteneciente a la librería pandas, para ello es necesario pasarle a este algoritmo el nombre del archivo con el que se va almacenar, "archivo_excel" y la acción de si se desea incluir el índice del "dataframe" que en el caso particular de este proyecto no interesa registrarlo y por ello se asocia con la variable "False".

3.6.5. Cierre del programa

Para finalizar la explicación de todo el proyecto es necesario dar una explicación de los pasos que realiza el programa para poder cerrar los archivos y dejar de ejecutar las operaciones detalladas en todos los puntos previamente explicados.

```
289    # Cerrar los objetos VideoCapture, VideoWriter y todas las ventanas
290    captura.release()
291    out.release()
292    cv2.destroyAllWindows()
```

Figura 3.61: Comandos que ejecutan el cierre del programa.

En la *Figura 3.61*, hace referencia a las tres últimas líneas de código encargadas de ejecutar los comandos que liberan las variables principales del programa y cierran el mismo.

El comando "reléase", en la línea 290 de la *Figura 3.61*, se encarga de liberar o eliminar los datos que haya almacenados en una variable, en este caso en la variable "captura" donde se sabe que se tienen almacenados todos los "frames" del video original objeto de estudio. Esta acción se realiza al final del programa para evitar que se sobrecargue la memoria del programa y se puedan generar diversos problemas.

En la línea de código 291 realiza la misma acción que la anterior, pero en este caso lo que hace es eliminar los datos almacenados en la variable “out” definida al comienzo del programa y sirve para poder mostrar por pantalla el video con las operaciones que se vayan realizando sobre él mismo y que contiene las mismas propiedades, proporciones y características que el video original. Para ello libera todos los comandos que estén asociados a la variable “out” que hayan utilizado este recurso a lo largo de todo el programa.

El comando “destroyAllWindows” ejecuta la acción de cerrar todas las ventanas que haya ido generando la librería OpenCv en la ejecución del programa como son las de reproducción del video sobre el que se desarrollas las operaciones. Este comando es muy útil ya que permite limpiar la interfaz gráfica y evita que las ventanas permanezcan abiertas después de que el programa haya terminado de ejecutarse.

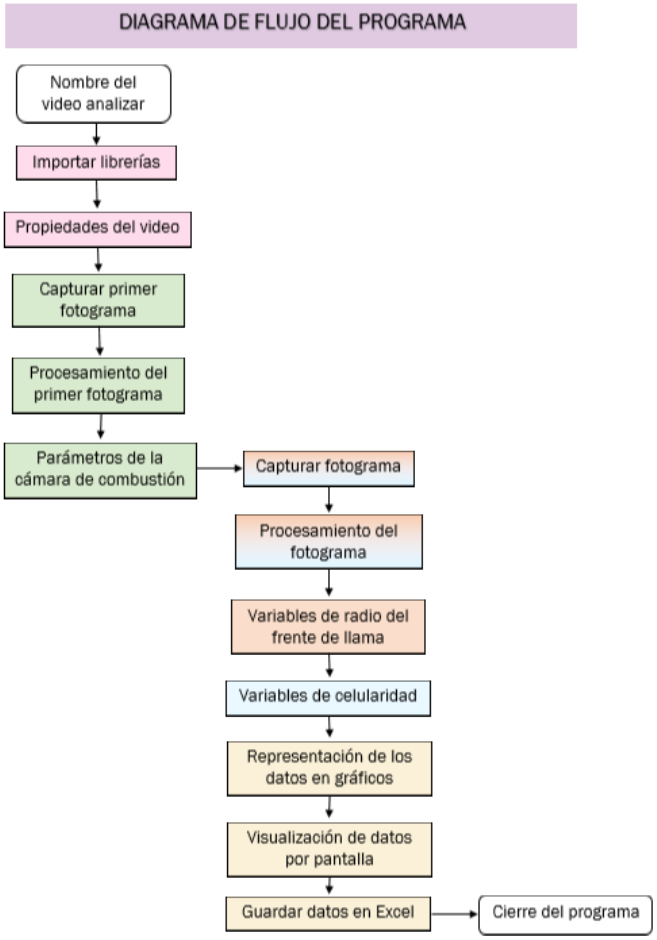


Figura 3.62: Diagrama de la de flujo del proceso completo.

4. PROCEDIMIENTOS ARA EL ANÁLISIS DEL FOTOGRAMA

En este apartado se exponen todas las opciones que se han utilizado durante el desarrollo del programa y que se han descartado para poder realizar la mejor aproximación para obtener los mejores resultados posibles en el análisis del fotograma.

Todas las operaciones que se plantean en este apartado están comentadas dentro del código para que no se realice su ejecución, pero en el caso de realizar mejoras en el futuro dentro del programa, se quedan planteadas para usarlas por si aportaran de una manera más clara información relevante de los datos obtenidos.

Se explicarán las distintas funciones planteadas para la obtención de los datos y se dará de manera detallada por qué no han sido implementadas dentro del código.

En este punto del proyecto, se desarrollan los procedimientos utilizados tanto para el cálculo del radio del frente de llama como para el cálculo de la celularidad. Respecto al resto del programa no se han realizado ningún tipo de proceso que haya necesitado ser expuesto en este apartado, por lo que no se va a entrar en detalle en ellos.

4.1. Procedimientos utilizados para el cálculo del radio del frente de llama

Los algoritmos y funciones utilizadas dentro del bucle “while” anteriores al cálculo de el rectángulo que engloba los contornos seleccionados por el usuario en la variable “modificacion_area”, no han sufrido ningún tipo de alteración respecto a la forma de calcular alguno de los parámetros.

Tras el cálculo de los parámetros y variables que se generaban tras la detección del rectángulo que englobaba los contornos designados para el estudio del video, se podían utilizar diversos procedimientos que permitían calcular el radio del frente de llama de una manera u otra. Partiendo de los datos obtenidos cabía la posibilidad de detectar los contornos mediante circunferencias o bien mediante una elipse. En el caso de que la aproximación se realizara mediante una circunferencia, surgían tres nuevas posibilidades: la detección por medio de la circunferencia que utilizaba como radio el diámetro mayor del rectángulo, la circunferencia que utilizaba como radio el diámetro menor del rectángulo o la circunferencia que se calcula como la inversa de la suma de las inversas de los diámetros.

Todas las detecciones que se explicarán a continuación deben estar inscritas dentro del rectángulo generado a partir de los contornos seleccionados para realizar el análisis.

Para visualizar que tipo de aproximación es mejor para dejarla implementada dentro del proyecto, se calculan los datos necesarios para poder visualizar por pantalla el

cuadrado que contiene a los contornos seleccionados. La explicación que se detalla a continuación del procedimiento que se ha seguido para visualizarlos sobre el fotograma, parte desde el punto que se visualiza sobre la *Figura 3.24* del apartado 3.3.5. donde se establecen las variables necesarias para realizar los cálculos y operaciones necesarias para realizar el cálculo del rectángulo, el análisis de las elipses y el análisis de las circunferencias. [21]

```
131         # Calculamos los vertices del rectangulo
132         vertices =cv2.boxPoints(rectangulo)
133         vertices = np.int0(vertices)
134         orden = np.argsort(vertices[:, 0])
```

Figura 4.1: Parámetros de cálculo del rectángulo que permiten dibujarlo sobre el fotograma.

El procedimiento que se muestra en la *Figura 4.1*, se encuentra situado dentro del bucle "if" que depende de que coincidan el número de contornos a analizar con el número de contornos que ha establecido el usuario. A su vez, este procedimiento se encuentra dentro del bucle "while", encargado de ejecutar todo el procedimiento de análisis de un fotograma tras otro.

Para poder dibujar el cuadrado sobre el rectángulo calculado en la variable "rectangulo" que se muestra en la línea de código 139 de la *Figura 3.24* del apartado 3.3.5., es necesario obtener de esa variable los vértices de dicho rectángulo. Para ello empleamos la función de la librería OpenCv definida como "cv2.boxPoints()" que recorre todos los datos almacenados en la variable "rectangulo" y devuelve las coordenadas de los cuatro vértices de este. El tipo de dato que se pasa es un rectángulo rotado. Esta función permite convertir datos complejos en su manejo, en datos más sencillos y que disponen de variables independientes a las que se puede acceder. Cada vértice se representa mediante las coordenadas x e y medido en pixeles y se almacenan en una matriz de datos definida como "vertices" que se utilizará posteriormente.

Como se ha mencionado, los datos almacenados en una matriz "vertices" no son número enteros, por lo que para poder trabajar con ellos se deben pasar los datos en pixeles a número enteros utilizando la función de la librería numpy denominada "np.int0()". Esta función se asegura que el tipo de dato sea un valor entero sin signo de 64 bits. Transformamos las coordenadas x e y establecidos al comienzo en pixeles, en valores enteros. Se almacenan en la misma lista de variable en la que se encontraban los valores en pixeles, es decir, se inscribe sobre la variable "vertices".

Por último, en la línea de código 134 de la *Figura 4.1*, se ordenan los vértices detectados de ese rectángulo rotado en función de la coordenada x. Para poder realizar esta acción, se hace uso de la función "np.argsort()" perteneciente a la librería Numpy. Para poder ordenar los vértices de la matriz buscada, en el caso del proyecto de la variable "vertices", se accede a todas las filas de la matriz, pero solo

a la primera columna que es la que contiene las coordenadas x que se desean ordenar. La matriz ordenada se almacena en un variable denominada como “orden”.

```
138             cv2.drawContours(fotograma, [vertices], 0, (0, 255, 0), 2)
```

Figura 4.2: Función que dibuja el rectángulo con los parámetros calculados previamente y sobre el que tiene que estar contenidos la elipse y las circunferencias.

El algoritmo que se utiliza para dibujar sobre rectángulo planteado sobre el fotograma es el que se muestra en la *Figura 4.2*. Para realizar esta acción, es necesario pasarle al algoritmo la imagen sobre la que se desea realizar la operación, la variable que tiene almacenado el “fotograma”. Como lo que se pretende dibujar es el rectángulo sobre el fotograma, es necesario pasarle a la función los vértices de dicho rectángulo que se encuentran almacenados en la matriz “vertices”. Los tres últimos parámetros que necesita la función para poder dibujar el rectángulo sobre el fotograma, es el elemento que permite acceder únicamente el primer contorno de la lista, dibujando únicamente el rectángulo; seguido a ello se establece los datos para poder dibujar el rectángulo con un color (0, 255, 0), que como está en código RGB, que coincide con el color verde y por último, se establece el grosor de la línea con el que se dibujan los contornos que para este caso son dos pixeles de grosor.

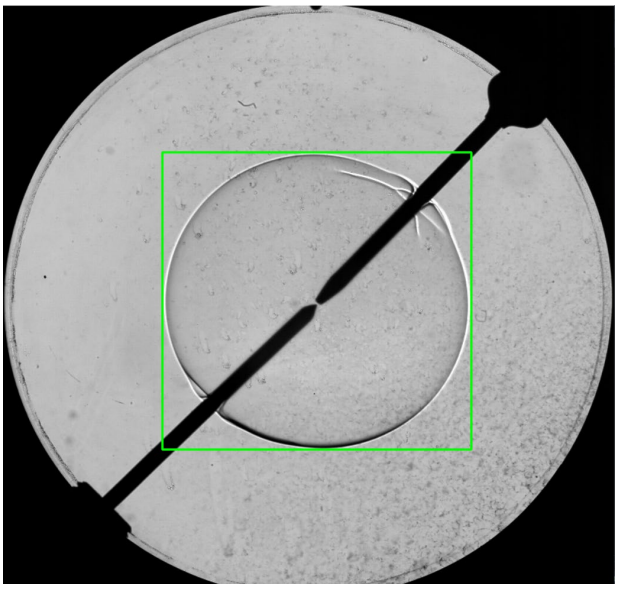


Figura 4.3: Rectángulo generado con el número de contornos establecido en la variable “modificacion_area”.

4.1.1. Análisis mediante elipses

En este apartado se explicará de manera detallada los parámetros necesarios para realizar la aproximación de los contornos detectados mediante una elipse circunscrita dentro del rectángulo que abarca los contornos.

```
139         for cnt in contornos_grandes:
140             if len(cnt) < 5.7:
141                 continue
142             try:
143                 elipse = cv2.fitEllipse(cnt)
144                 centro_elipse, (diametro_mayor_elipse, diametro_menor_elipse), angulo_elipse = elipse
145                 distancia_mayor = np.linalg.norm(np.array(centro_elipse) - np.array(centro_rectangulo))
146                 distancia_menor = max(diametro_mayor_elipse, diametro_menor_elipse) / 2
147                 diametro_mayor = max(diametro_mayor, distancia_mayor + distancia_mayor)
148                 diametro_menor = min(diametro_menor, distancia_menor + distancia_menor)
149                 angulo = angulo_elipse
150             except cv2.error:
151                 pass
```

Figura 4.4: Parámetros establecidos para la definición de la elipse contenida dentro del rectángulo para el experimento BC_E09.2.

Inicialmente, en la *Figura 4.4*, se inicializa un bucle "for" que itera sobre cada uno de los contornos que encuentra dentro de la lista "contornos_grandes". Esto permite acceder a cada uno de los datos de manera individual de los contornos almacenados en la lista y que sufran el proceso de análisis con operaciones establecidas en el interior del bucle if.

En el interior del bucle "for", se encuentra definido un bucle condicionante "if" que verifica si la longitud del contorno, número de puntos que componen el contorno, que se está analizando es menor que 5,7 cm, coincidente con el valor medido por el usuario del radio de la cámara de combustión. Si es menor, se pasa al siguiente contorno sin dejar de ejecutar el resto del código par el contorno definido. Dentro del bucle "if" se encuentra la ejecución "continue", que establece que, si no se cumple la ejecución de la condición del bucle, salta hacia fuera del bucle sin ejecutar el resto del código sobre ese contorno. Como se ha establecido el comando "try", que hace referencia a un bloque de excepción, ejecuta las operaciones contenidas en el interior de dicha excepción.

Dentro del parámetro "try" que se muestra en la línea de código 142 de la *Figura 4.4*, se encuentran los parámetros que definen la elipse que se busca en ese apartado. Para ello se hace uso de la función de la librería OpenCV denominada "fitEllipse()" donde ajusta los puntos del contorno que está analizando en ese momento de los que encuentre dentro de la lista "contornos_grandes" y los ajusta a una elipse de la mejor manera. Los datos que se calculan haciendo uso de esta operación, se almacenan en una lista de datos que se guardan en la lista definida como "elipse".

Una vez detectados los datos dentro de la elipse, se desempaquetan los resultados que contiene esa matriz y se almacenan en variables cada una referente a la posición de los datos dentro de la variable "elipse". El primer dato hace referencia al centro de la elipse, guardando su posición en el eje x e y, dentro de una variable llamada "centro_elipse". El segundo y tercer dato que están definidos hacen referencia al diámetro mayor y menor de la elipse respectivamente. Por último, la última variable que se encuentra almacenada dentro de la variable "elipse" es la referente al ángulo de inclinación de dicha elipse.

Seguido de ello se calcula la distancia entre el centro de la elipse y el centro del rectángulo por medio de la función perteneciente a la librería Numpy denominada “np.linalg.norm()” donde se deben de pasar los datos del “centro_elipse” y “centro_rectangulo” como matrices para poder realizar la operación, por ello se encuentran dentro de “np.array()”. En la línea de código 146 de la *Figura 4.4*, se calcula la “distancia_menor” al centro de la elipse, para ello evalúa el máximo de los datos definidos como “diametro_mayor_elipse” y “diametro_menor_elipse” y seguidamente el mayor de los datos lo divide entre dos. Por último, calcula el “diametro_mayor” haciendo uso del dato de “distancia_mayor” calculada anteriormente y sumamos este último dato dos veces y se busca el máximo de ambos datos, “diametro_mayor” y “distancia_mayor” obteniendo así la variable actualizada de “diametro_mayor”. De una manera muy similar se realiza la actualización del “diametro_menor”, pero realizando el mínimo entre el “diámetro_menor” y la “distancia_menor” sumada dos veces. El mínimo de ambos datos hace referencia a la variable actualizada de “diametro_menor”.

La línea de código 150 de la *Figura 4.4*, se define la línea “except cv2.error” donde se captura lo que ocurra dentro del boque “try”, es decir, define la elipse, pero si detecta sobre ella datos que generen errores lo que hace es pasar de esa ejecución.

```
152         elipse_circunscrita = ((int(centro_rectangulo[0]), int(centro_rectangulo[1])), (int(diametro_mayor), int(diametro_menor)), angulo)
153         cv2.ellipse(fotograma, elipse_circunscrita, (255, 0, 0), 2, cv2.LINE_AA)   # Elipse del contorno, color azul
```

Figura 4.5: Comandos para dibujar la elipse sobre el fotograma.

Una vez calculados los datos referentes a la elipse, se procede a dibuja sobre el fotograma la elipse que se ha calculado previamente. Para ello lo primero que se hace es introducir todos los parámetros referentes a la elipse dentro de una matriz introduciendo para ello el “centro_rectangulo”, sus coordenadas x e y, desde donde se va a generar la elipse. Seguido a ello se pasan los datos referentes al “diametro_mayor” y “diametro_menor” establecidos dentro del bloque “try” y por último, el ángulo de inclinación de la elipse. Para generar la elipse se debe de trabajar con datos enteros.

Con los datos almacenados en la matriz “elipse_circunscrita”, se dibuja la elipse. Para poder dibujarlas se necesitan la imagen sobre la que se va dibujar la elipse, siendo esta el dato almacenado dentro de la variable “fotograma”. Seguido de este dato, se pasan los datos con la información para dibujar la elipse, es decir, el dato dentro de “elipse_circunscrita”. Por último, se dan los parámetros para que aparezcan sobre el fotograma de manera visual, es decir, se establece el color (255, 0, 0) dibujando la elipse en color azul sobre el fotograma y después el grosor, que son dos pixeles. El último de los datos hace referencia al tipo de línea que se usa para dibujar la elipse que es una línea suave y poco pixelada.

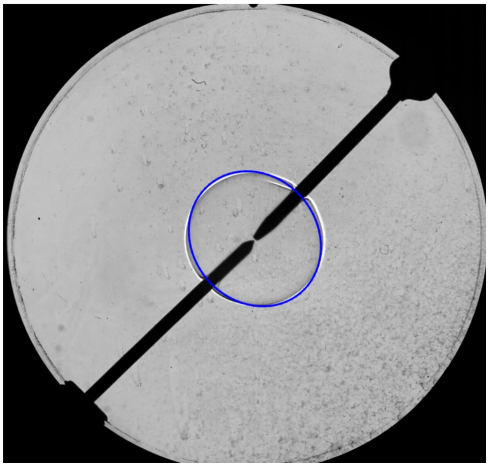


Figura 4.6: Elipse generada para una aproximación al radio del frente de llama del experimento BC_E09.2.

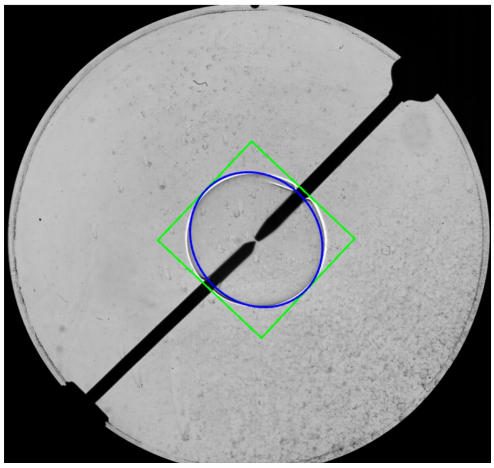


Figura 4.7: Elipse inscrita en el rectángulo formado por los contornos asignados en la variable "modificacion_area" para el experimento BC_E09.2.

4.1.2. Análisis mediante circunferencia con el diámetro mayor

Otra de las formas que podemos utilizar para realizar la aproximación por la circunferencia es la de utilizar el dato almacenado en la variable "diametro_mayor" calculada a partir del espacio generado por el rectángulo explicado en el apartado 3.3.5 en la línea de código 139 de la Figura 3.24.

```
154        cv2.circle(fotograma, (int(centro_rectangulo[0]), int(centro_rectangulo[1])) , int(diametro_mayor/2), (0, 255, 255), 2)
```

Figura 4.8: Comandos para dibujar la circunferencia circunscrita dentro del rectángulo dando como radio el "diametro_mayor" del rectángulo.

Para poder dibujar esta circunferencia, es necesario pasarle al sistema la imagen sobre la que se va a dibujar la circunferencia, "fotograma", el centro desde donde se va a dibujar la circunferencia, es decir, las coordenadas x e y del centro del rectángulo como datos enteros, "centro_rectangulo". Seguido de este centro, se establece el radio con el que se va a dibujar la circunferencia, es el valor almacenado en la variable "diametro_mayor" dividido entre 2. Seguido de ello se dibuja la circunferencia en color amarillo, (0, 255, 255) y el grosor de la circunferencia que son 2 pixeles.

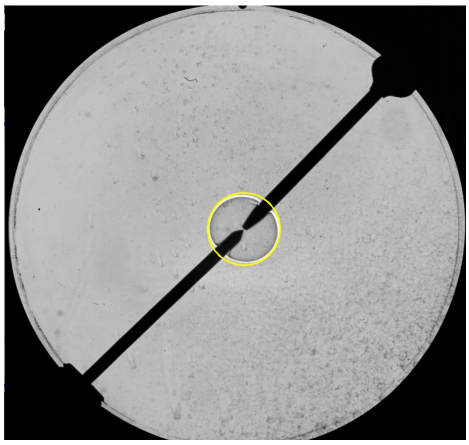


Figura 4.9: Circunferencia generada tomando como radio el diámetro mayor para una aproximación al radio del frente de llama del experimento BC_E09.2.

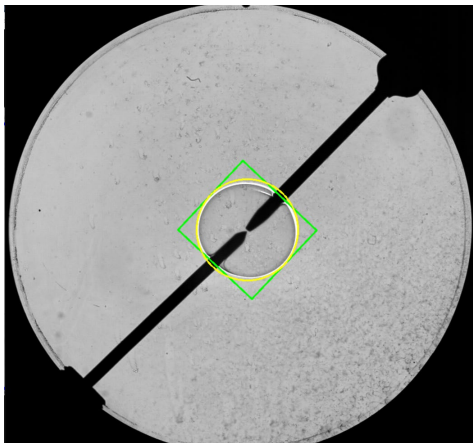


Figura 4.10: Circunferencia generada por medio del diámetro mayor del rectángulo formado por los contornos asignados en la variable "modificacion_area" para el experimento BC_E09.2.

4.1.3. Análisis mediante circunferencia con el diámetro menor

Al igual que se ha realizado con el dato almacenado en la variable "diametro_mayor", se realiza el mismo procedimiento, pero utilizando como radio el "diametro_menor" obtenido a partir del rectángulo definido en el apartado 3.3.5 en la *Figura 3.24* de la línea de código 139.

```
155    cv2.circle(fotograma, (int(centro_rectangulo[0]), int(centro_rectangulo[1])), int(diametro_menor/2), (255, 0, 255), 2)
```

Figura 4.11: Comandos para dibujar la circunferencia circunscrita dentro del rectángulo dando como radio el "diametro_menor" del rectángulo.

Para poder dibujar esta circunferencia, se utiliza al igual que en el punto 4.1.2 la función "cv2.circle()" donde se pasa la variable "fotograma" sobre la que se dibuja la circunferencia, se pasa el "centro_rectangulo" sobre el que se va a generar la circunferencia, pasando los datos como valores enteros. Los datos que se modifican con respecto son el de radio donde se pasa el valor de "diámetro_menor" entre dos y se transforma a dato entero. Seguido de ello se establece el color en que se va a dibujar que es (255, 0, 255) que es color magenta y con dos pixeles de grosor.

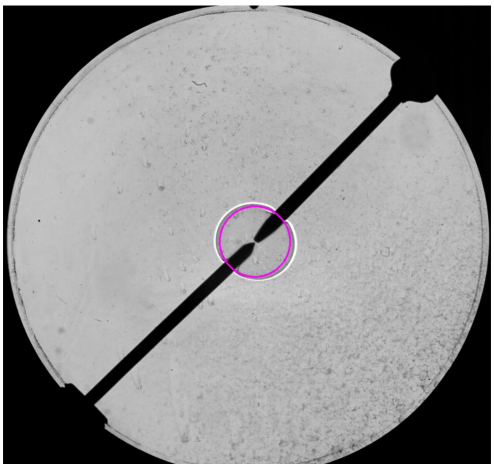


Figura 4.12: Circunferencia generada tomando como radio el diámetro menor para una aproximación al radio del frente de llama del experimento BC_E09.2.

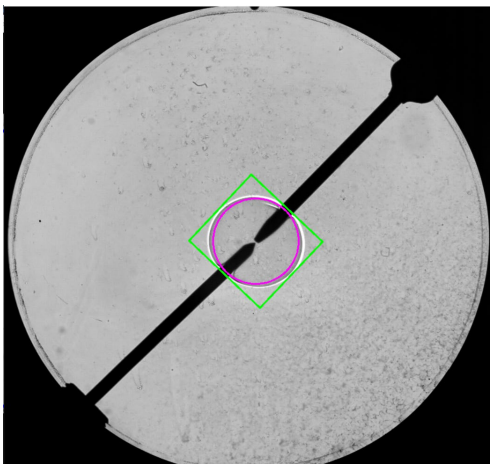


Figura 4.13: Circunferencia generada por medio del diámetro menor del rectángulo formado por los contornos asignados en la variable "modificacion_area" para el experimento BC_E09.2.

4.1.4. Análisis mediante circunferencia con la inversa del radio

El último de los análisis que se ha realizado es la utilización como radio de la inversa de la suma de las inversas de los diámetros mayor y menor del rectángulo de partida. Esta media está explicada en el apartado 3.3.5 en la línea de código 143 que aparece en la *Figura 3.24*. Este dato esta almacenado en una variable denominada "radio". Para poder obtener el valor de esta variable, se ha utilizado la siguiente fórmula.

$$\frac{1}{R} = \frac{1}{D_{mayor}} + \frac{1}{D_{menor}} \quad \rightarrow \quad R = \frac{D_{mayor} * D_{menor}}{D_{mayor} + D_{menor}} \qquad (Ec.\ 9)$$

Siendo R el valor del radio que se representa sobre el fotograma y que está inscrito dentro del rectángulo.

```
156     cv2.circle(fotograma, (int(centro_rectangulo[0]), int(centro_rectangulo[1])), int(radio), (0, 0, 255), 2)
```

Figura 4.14: Comandos para dibujar la circunferencia circunscrita dentro del rectángulo dando como radio la inversa de la suma de las inversas de los diámetros que se almacena en la variable "radio".

Se utiliza el comando perteneciente a la librería OpenCV denominado "cv2.circle()". Al igual que en los casos anteriores, se definen el fotograma donde se va a visualizar la circunferencia, el centro del rectángulo, definido por datos enteros, desde donde comienza la circunferencia. Seguido a este último dato, se establece el radio con el que se va a ir generando la circunferencia en cada uno de los fotogramas y se transforma dato entero para que el sistema pueda comprenderlo de mejor manera. Por último, se define el dato con el que queremos que aparezca sobre el fotograma que es en este caso (0, 0, 255) que corresponde con el color azul como la elipse y con un grosor de dos pixeles.

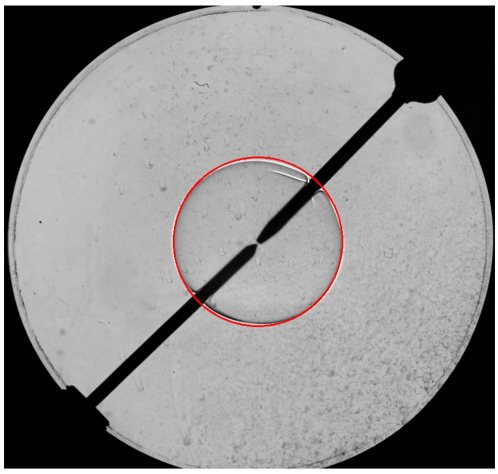


Figura 4.15: Circunferencia generada tomando como radio la inversa de la suma de las inversas de los diámetros para una aproximación al radio del frente de llama del experimento BC_E09.2.

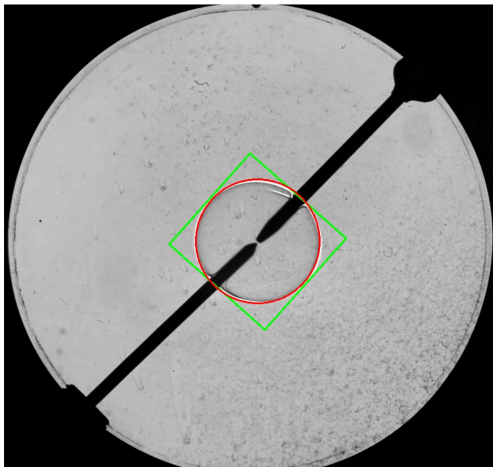


Figura 4.16: Circunferencia generada por medio de la inversa de la suma de las inversas de los diámetros del rectángulo formado por los contornos asignados en la variable "modificacion_area" para el experimento BC_E09.2.

4.1.5. Selección del análisis más aproximado al radio del frente de llama

Una vez definidos cómo se obtienen todos los parámetros para la creación sobre el fotograma de la elipse y las circunferencias, se procede a discutir por qué se ha elegido el procedimiento del análisis mediante circunferencia con la inversa de la suma de las inversas de los diámetros, como está definido en el apartado 4.3.6.

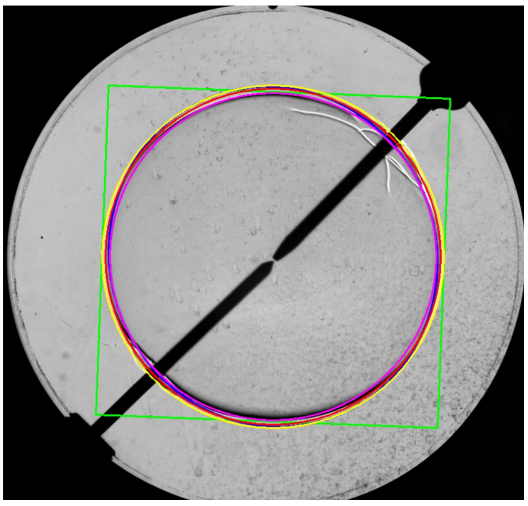


Figura 4.17: Imagen comparativa donde se comparan todas las formas de generar las circunferencias que se aproximen de manera más exacta al radio del frente de llama. Rectángulo en color verde, elipse en color azul, diámetro mayor en color amarillo, diámetro menor en color rosa e inversa de la suma de las inversas de los diámetros en color rojo para el experimento BC_E09.2.

Como se puede observar en la *Figura 4.17* donde se comprarán todos los radios, el cálculo que más se aproxima al contorno real del frente de llama que se muestra en

el fotograma la circunferencia generada a partir de la inversa de la suma de las inversas del diámetro. Las demás opciones no se alejan de los resultados, pero son menos exactas que la última que se ha explicado.

4.2. Procedimientos utilizados para el cálculo de la densidad celular

Al igual que se ha realizado con el cálculo del radio del frente de llama, para obtener la densidad celular se han aplicado diversos métodos para comprobar cuál de ellos se obtiene mejores resultados y así poder aplicarlos al análisis de los experimentos. Los algoritmos y funciones utilizadas dentro del bucle "while" anteriores al cálculo de la celularidad que engloba los contornos seleccionados por el usuario en la variable "modificacion_area", no han sufrido ningún tipo de alteración respecto a la forma de calcular alguno de los parámetros ni la forma en la que se ha obtenido el radio del frente de llama.

Tras obtener el radio del frente de llama y representarlo sobre el fotograma correspondiente, se debe de detectar la celularidad que se genera, o no, en el interior de ese radio, es decir, se deben de detectar los contornos del frente de llama para cada instante del video. Hay muchas maneras de obtener esos contornos, entre las que se encuentran las dos que se han planteado en este proyecto. Cabe la posibilidad de detectar los contornos haciendo uso de una función que adelgaza los elementos que están almacenados dentro de los pixeles o bien aplicar una función que permite detectar los contornos por medio de diferencia de intensidad en los pixeles.

Para aplicar las dos funciones, es necesario generar una máscara circular al 80% del radio utilizado para calcular el radio del frente de llama, desarrollado en los puntos anteriores, para asegurarse que en el interior de dicha máscara siempre se van a encontrar contornos que van un valor a la densidad celular. Esta máscara se copia en un nuevo fotograma. Tras ello se sustrae el fondo del resto del fotograma porque no proporciona información útil para el cálculo y se aplica la función de "umbralizacion" que proporciona intensidad a los pixeles dependiendo de sus valores.

```
179         # Crear una máscara circular
180         mask = np.zeros(fotograma.shape[:2], dtype=np.uint8)
181         # Esta fijado el centro con el obtenido previamente en la cámara de combustión
182         cv2.circle(mask, (int(centro_rectangulo[0]), int(centro_rectangulo[1])), int(radio_celularidad), 255, -1)
183         cv2.circle(fotograma, (int(centro_rectangulo[0]), int(centro_rectangulo[1])), int(radio_celularidad), (0, 255, 255), 2)
184
185         # Aplicar la máscara al fotograma copiado para obtener solo los píxeles dentro de la circunferencia
186         roi = cv2.bitwise_and(copia, copia, mask=mask)
187
188         # Aplicar el sustractor de fondo y umbralización solo a la región de interés
189         mascara_roi = fgbg.apply(roi)
190
191         # Aplicar un umbral para obtener solo los píxeles más brillantes en la imagen
192         umbralizacion_roi = cv2.threshold(mascara_roi, 127, 255, cv2.THRESH_BINARY)[1]
```

Figura 4.18: Comandos previos a la aplicación de la función que define los contornos.

Una vez aplicados los distintos parámetros, se aplica la función que detecta los contornos en la imagen.

4.2.1. Transformada de adelgazamiento

Esta técnica se usa para reducir de manera gradual el ancho de los objetos detectados en el fotograma hasta convertir los elementos detectados en líneas cuyo grosor sea de un pixel. Se busca con ello simplificar la representación de los elementos detectados sin perder la forma de los mismo. Para poder aplicar esta técnica se ha usado la función de la librería OpenCV que se observa en la *Figura 4.19*.

```
194         # Aplicar la transformación de adelgazamiento y encontrar los contornos en la imagen binaria resultante
195         thin_roi = cv2.ximgproc.thinning(umbralizacion_roi)
```

Figura 4.19: Función transformada de adelgazamiento de la librería OpenCv aplicada para la detección de la celularidad.

El algoritmo que se muestra en la *Figura 4.19* es un proceso iterativo, donde analiza unos primeros datos escaneando la imagen de apartida, en este caso la imagen donde los pixeles están más brillantes que es la umbralizada y se eleminan los pixeles que no contienen información relevante. Este proceso se realiza hasta que el fotograma que se muestra contiene todos los pixeles con la información mínima, adelgazando los elementos al máximo.

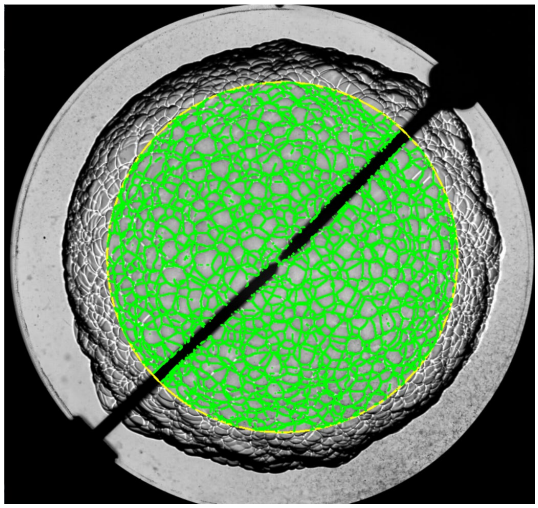


Figura 4.20: Celularidad detectada usando el algoritmo de la transformada de alargamiento para el experimento BC_E15.

4.2.2. Detección de bordes

Esta técnica se utiliza mayormente en aplicaciones donde se busca detectar los límites y discontinuidades que pueda tener la imagen. Los bordes del frente de llama presentan formas abruptas cuando aparece la celularidad sobre el mismo lo que favorece a que los pixeles en esas zonas cambien de intensidad, lo que permite detectarlos y representarlos por medio de esta técnica.

```
196         # Detección de bordes
197         canny = cv2.Canny(umbralizacion_roi, 50, 150)
```

Figura 4.21: Función detección de contornos de la librería OpenCv aplicada para la detección de la celularidad.

Al aplicar esta técnica se pasan diferentes variables a la función "cv2.Canny()" que es el algoritmo de la librería OpenCv que realiza un cálculo de los gradientes de intensidad de la imagen en base a unos parámetros establecidos por el usuario. Lo primero que se pasa a la función es el fotograma sobre el que se van a detectar los bordes, en este caso "umbralización_roi". Posteriormente se asignan dos datos, siendo el primero de ellos el valor umbral inferior y el segundo el valor umbral superior. Si los pixeles tienen mayor intensidad que 50, valor umbral mínimo que permite el sistema, se consideran bordes fuertes, mientras que los que se encuentran entre ambos límites se consideran bordes débiles. Estos datos permiten clasificar los pixeles y posteriormente identificar estudiar los de mayor intensidad que son los que van corresponder con la celularidad que se muestra sobre el frente de llama.

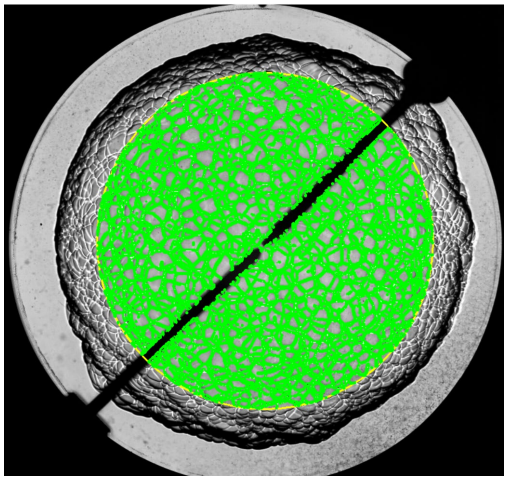


Figura 4.22: Celularidad detectada usando el algoritmo de detección de bordes para el experimento BC_E15.

4.2.3. Selección del análisis más aproximado para la detección de la celularidad

Una vez definidos cómo se aplican las dos técnicas planteadas en este proyecto para la detección de la celularidad, se procede a discutir por qué se ha elegido el procedimiento de la transformada de adelgazamiento, como está definido en el apartado 3.4.3.

Como se puede observar en la *Figura 4.21*, los contornos que se encuentran en el interior del 80% del radio del frente de llama corresponden exactamente con la celularidad que se muestra en el fotograma del experimento original. Por otro lado, en la *Figura 4.22*, los contornos detectados también coinciden con los del fotograma

original, pero detecta elementos de forma más nítida que se encuentran en segundo plano, celularidad que se produce en la parte posterior del frente de llama. La mejor selección para analizar la celularidad en este proyecto es la que emplea la transformada de adelgazamiento ya que trata los contornos de una manera más nítida y no detecta tantos contornos de la parte posterior del frente de llama.

5. RESULTADOS

Una vez desarrollada la aplicación, se realizan una serie de pruebas que permitan comprobar que funciona de manera correcta y que ejecuta las operaciones y acciones necesarias para obtener los resultados buscados.

Para comprobar dicha funcionalidad, se analizan en este apartado los diferentes resultados obtenidos del programa. Se realiza un estudio comparativo entre el programa desarrollado en este Trabajo de Fin de Grado y el programa del que se disponía previamente en Matlab que ha servido de base para el desarrollo de este proyecto; otro de los estudios consiste en comprobar cómo afecta la modificación de la variable "modificación_area" a los resultados obtenidos; y el último de los resultados planteados consiste en realizar una comparativa y análisis más dirigido al ámbito del desarrollo de la combustión de los diferentes experimentos realizados, manteniendo la presión inicial igual para todos ellos.

5.1. Comparativa de los resultados obtenidos por Matlab y Python

Tras desarrollar el nuevo programa haciendo uso del lenguaje de programación Python, se han analizado todos los experimentos realizados en el laboratorio y se han comparado los resultados obtenidos haciendo uso del nuevo programa con los resultados obtenidos cuando se analizaron los experimentos por el programa de Matlab.

Se han observado una serie de diferencias entre los resultados obtenidos por medio del programa de Matlab con respecto de los resultados obtenidos haciendo uso del nuevo programa desarrollado en Python.

Las diferencias se han observado tanto en el análisis del radio como en los análisis de objetos de celularidad.

5.1.1. Resultados relacionados con el radio

Se han analizado todos los experimentos por medio de los dos programas y se han observado una serie de diferencias entre ambos programas. Tanto el programa de Matlab, programa que ha servido de base para realizar las mejoras en el nuevo lenguaje, como el nuevo programa de Python, cumplen ambos con las funciones establecidas para su uso entre las que se encuentra la obtención del radio del frente de llama, con el cual se obtiene, de manera gráfica usando la variable tiempo también calculada, la velocidad del frente de llama. Los dos programas, dependiendo del experimento que se analice, siguen de manera correcta el radio del frente de llama, no obstante, se han observado una serie de errores que impiden, en ciertos experimentos, que los datos obtenidos para el radio sean correctos.

Se han detectado una serie de errores que comete el programa de Matlab que el programa de Python resuelve y se han detectado una serie de errores aparecen de manera puntual en ciertos datos obtenidos por el programa de Python.

Los errores más notorios que se generan en el programa de Matlab son los siguientes:

5.1.1.1. *Sobredimensionamiento – estancamiento– coincidencia*

Es uno de los errores más comunes en el análisis de realizado por Matlab. Se produce un desajuste del radio obtenido por el programa de Matlab con respecto al obtenido por el programa de Python.

El programa de Matlab comienza sobredimensionando el radio durante un periodo largo de tiempo, ver Figuras 5.2. Posteriormente, durante unos instantes de tiempo, el radio detectado por el programa de Matlab se mantiene constante, es decir, para valores distintos de tiempo el valor del radio no se modifica, aunque realmente el radio que se muestra por el video sigue creciendo. Tras la fase de estancamiento, los datos del radio detectado por Matlab y Python comienzan a coincidir.

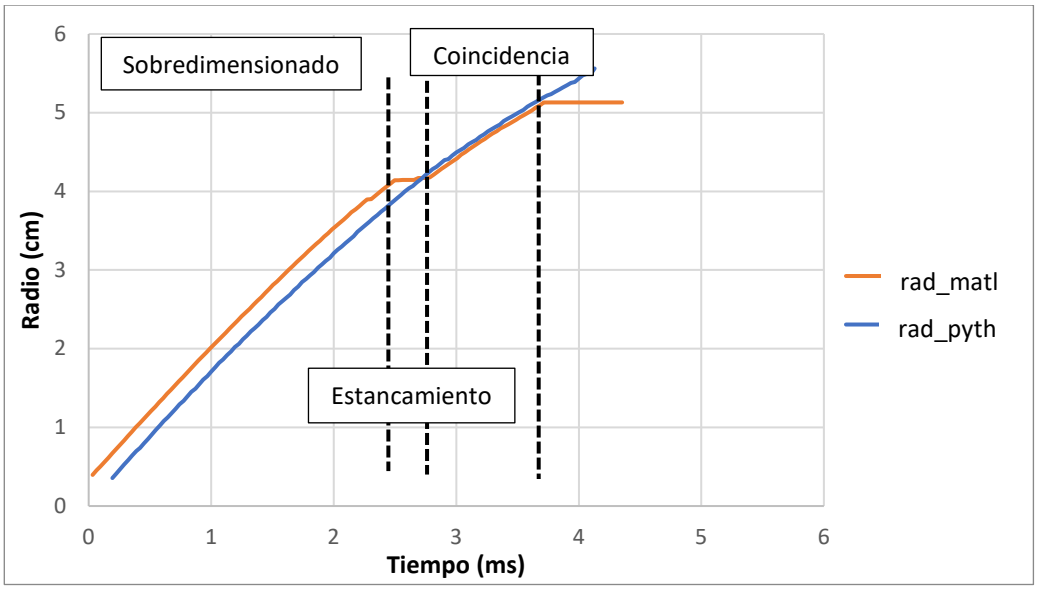


Figura 5.1: Comparación de radio obtenido por Matlab y Python para el experimento BC_E25.2

En la *Figura 5.1,* se muestra claramente lo explicado anteriormente. Desde el comienzo del análisis hasta aproximadamente los 2,5 s, los datos del radio obtenidos mediante el programa de Matlab adquieren valores superiores a los obtenidos mediante el programa de Python. Desde los 2,5 ms hasta los 2,7 ms, aproximadamente, el radio captado por Matlab se mantiene constante. Tras estas dos fases, el radio de Matlab saldría de la fase constante o de estancamiento y el radio que detecta en la resta de

fotogramas posteriores que coincide con los valores obtenidos por Python, etapa de coincidencia.

Durante la etapa de sobredimensionamiento, las llamas tienden a ser celulares lo que genera que la forma que tiene el programa de Matlab para captar el radio, realiza una resta de radios entre el fotograma en el que se encuentra y el radio obtenido en el fotograma anterior, la diferencia entre el primer fotograma y el siguiente sea grande, por lo que da lugar a un primer radio sobre el primer fotograma bastante grande y la diferencia con respecto a los posteriores se mantiene ya que detecta radios más grandes, todo esto genera que se sobredimensione el radio.
En la etapa de estancamiento suele coincidir con la aparición de la celularidad lo que provocando que el crecimiento del radio se ralentice un poco, dando lugar a una diferencia de radios entre el fotograma analizado y el anterior menor o prácticamente nula generando una región plana en la gráfica.

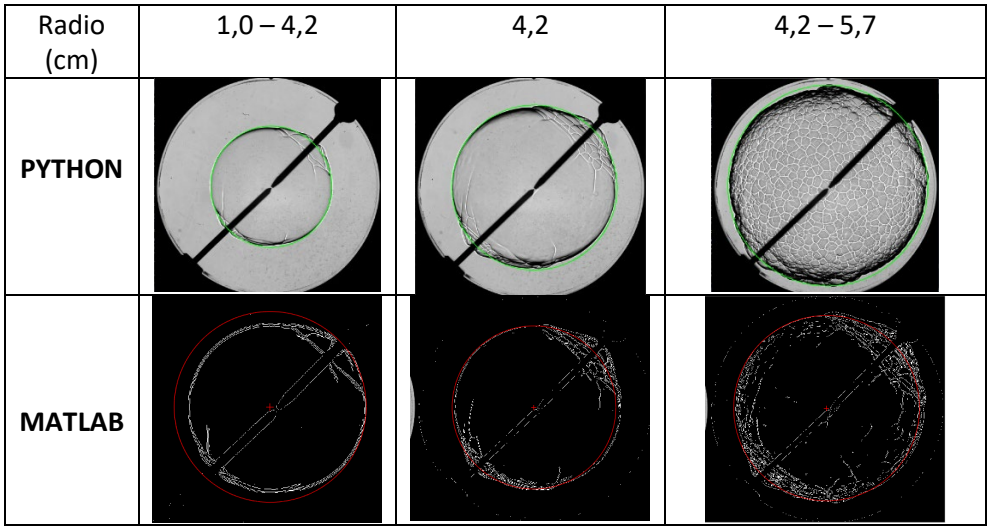


Figura 5.2: Fotogramas del experimento BC_E25.2 obtenidas por el programa de Python y Matlab en las fases de sobredimensionamiento, estancamiento y subestimación del radio.

Al salir de la fase constante, la diferencia de radios que capta el programa coincide con el radio del frente de llama del video que se analiza, es decir, comienza a captar el radio bien coincidiendo el círculo rojo que genera el programa de Matlab, por resta de radios, con la evolución del frente de llama que se muestras las imágenes.

Con respecto a la detección realizada por Python, no se restan los fotogramas, sino lo que se realiza es un análisis único del fotograma, detectando los contornos que encontrados dentro de ese fotograma y son lo que permiten calcular el radio, por lo que la progresión el radio es independiente de para cada fotograma. En el caso del uso de los resultados generados por este programa es necesario definir la variable “modificacion_area” que permite

utilizar el número justo para que el programa obtenga los mejores resultados para el radio. Para el experimento BC_E25.2 se ha establecido la variable "modificacion_area" con 3 contornos.

Posiblemente las distintas fases que se obtienen para el análisis del experimento puedan ser debido a que muchas de las llamas en su comienzo son muy laminares hasta radios elevados, próximos al valor de la cámara de combustión, donde aparece la celularidad, en esta etapa ya coinciden ambos radios, el de Python con Matlab.

5.1.1.2. *Sobredimensionamiento – estancamiento – coincidencia – subestimación*

Otro de los errores más comunes que se observan en los análisis al usar el programa de Matlab es que sucede el mismo procedimiento explicado anteriormente donde se produce una fase de sobredimensionamiento, seguido de una fase estable y posteriormente una zona de coincidencia con el radio de Matlab. Tras estos pasos hay en alguno de los experimentos donde se produce una subestimación del radio.

Se produce un desajuste del radio obtenido por el programa de Matlab con respecto al obtenido por el programa de Python.

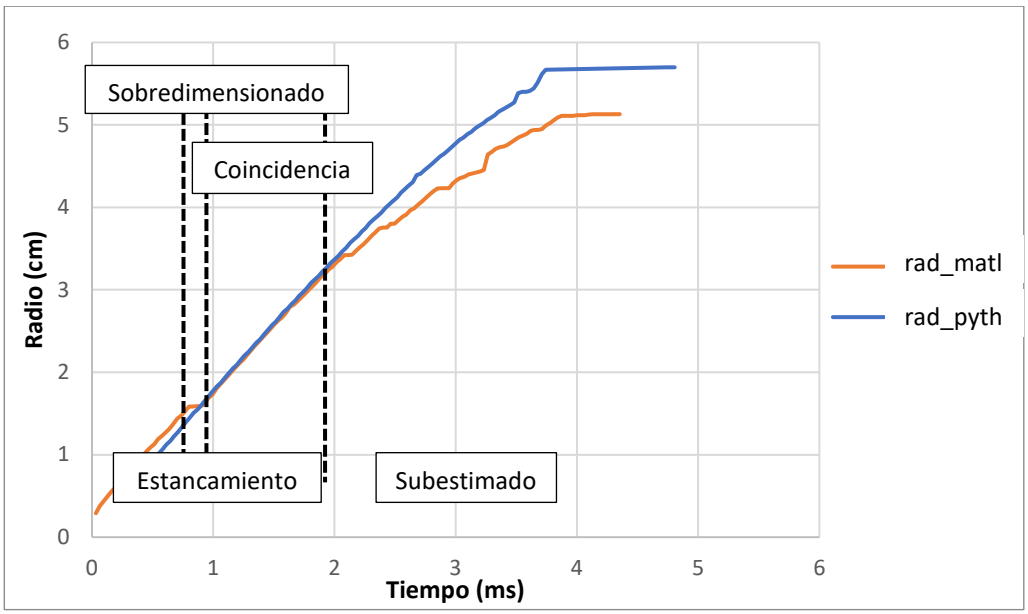


Figura 5.3: Comparación de radio obtenido por Matlab y Python para el experimento BC_E17.2

Para radios comprendidos entre 1 cm, o inferiores, hasta radios de 1,7 cm el radio esta sobredimensionado. Tras la fase de sobredimensionamiento se

produce la fase de estanqueidad para un radio de 1,7 cm que dura un periodo de 2 ms. Hay una fase donde los radios de ambos programas coinciden desde los 1,7 cm hasta los 3,5 cm. Posteriormente se produce la fase de subestimación del radio de Matlab donde los datos obtenidos de Matlab se mantienen por debajo de los obtenidos por Python.

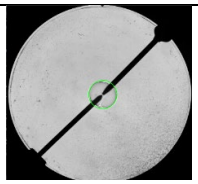
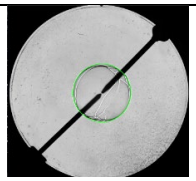
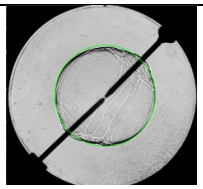
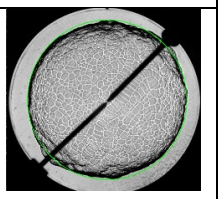
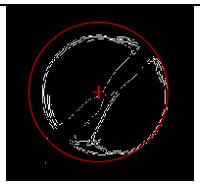
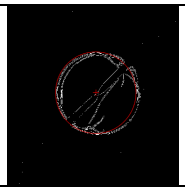
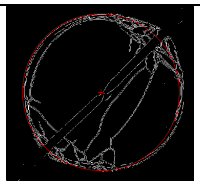
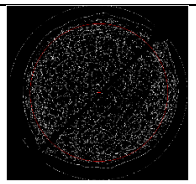
Radio (cm)	0,5 – 1,7	1,7	1,7 – 3,5	3,5 – 5,7
PYTHON				
MATLAB				

Figura 5.4: Fotogramas del experimento BC_E17.2 obtenidas por el programa de Python y Matlab en las fases de sobredimensionamiento, estancamiento, coincidencia y subestimación del radio.

En las *Figuras* representadas anteriormente, se muestran las diferentes etapas que suceden en las fases de sobredimensionamiento, la fase estable, la fase donde coinciden los resultados de ambos programas, el análisis de estas etapas es el mismo que se ha realizado en el punto anterior, y la etapa de subestimación del radio. En las etapas representadas en las *Figuras* la llama comienza siendo laminar y a medida que aumenta el radio del frente de llama comienzan aparecer grietas en el frente de llama que ralentizan la velocidad a la que se desarrolla la combustión.

En la zona donde el radio está subestimado coincide con la zona donde hay más celularidad. Posiblemente el programa de Matlab, al realizar la resta de fotogramas y buscar 3 puntos para generar una circunferencia, el programa detecte demasiados puntos con los que realizar el cálculo siendo algunos de ellos los que no coinciden con el contorno exterior del frente de llama.

Con respecto a la detección realizada por Python, no se restan los fotogramas, sino lo que se realiza es un análisis único del fotograma, detectando los contornos que encontrados dentro de ese fotograma y utilizando solo los contornos más grandes para realizar el cálculo del radio realizando la circunferencia circunscrita en el rectángulo que recoge los contornos asignados para los cálculos por medio de la variable “modificacion_area” que para el experimento BC_E17.2 se ha establecido su valor en 3 contornos.

En los experimentos en los que se ha observado que suceden estas etapas son principalmente en los que la celularidad aparece hacia la mitad de la

combustión y crece hasta el final de la misma cuando llega a las paredes de la cámara de combustión.

5.1.1.3. *Sobredimensionamiento – subestimación*

Se ha observado una peculiaridad al analizar y comparar todos los datos obtenidos para el radio del frente de llama usando el programa de Matlab y el programa de Python. El radio captado por el programa de Matlab sobredimensiona al comienzo y posteriormente lo subestima. La discrepancia se observa en el experimento BC_E19.

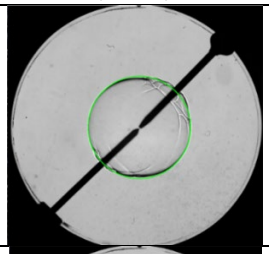
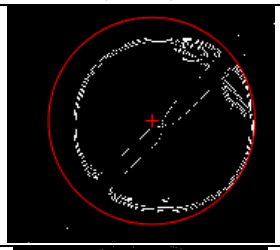
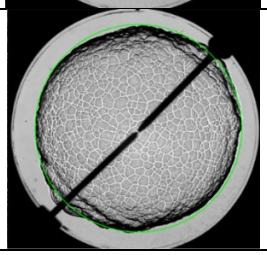
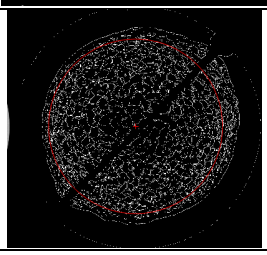
Radio (cm)	0,5 – 5,7	4,1 – 5,7
PYTHON		
MATLAB		

Figura 5.5: Fotogramas del experimento BC_E19 obtenidas por el programa de Python y Matlab en las fases de sobredimensionamiento y subestimación del radio.

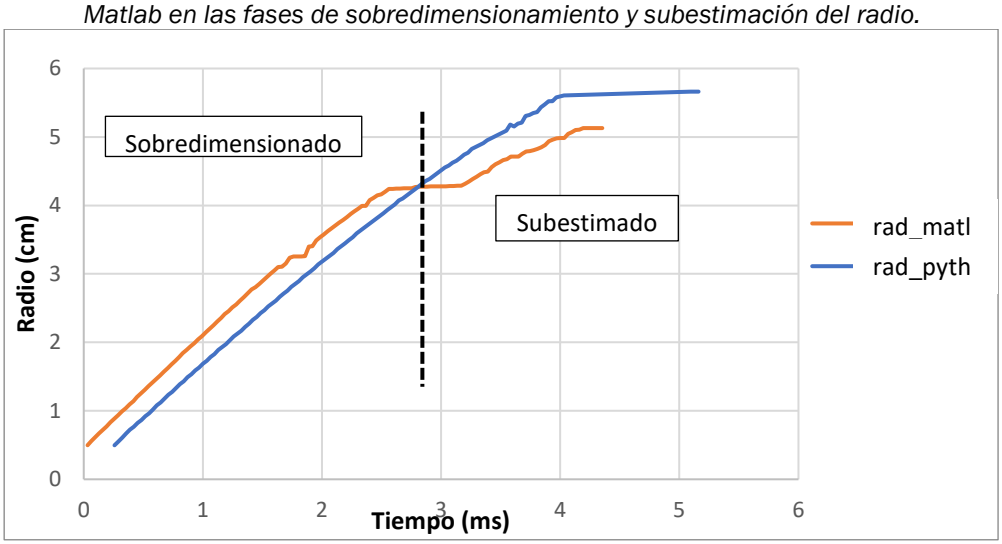


Figura 5.6: Comparación de radio obtenido por Matlab y Python para el experimento BC_E19 donde se observan las fases de sobredimensionamiento y de radio subestimado.

Para el gráfico comparativo que se observa con los resultados obtenidos en la *Figura 5.6,* se observan tres zonas muy diferenciadas: el

sobredimensionamiento del radio de Matlab, zona de estacionaria para el mismo código y la subestimación posterior del mismo. El radio del frente de llama obtenido por el programa de Python crece de manera normal como en otros experimentos analizados anteriormente estableciendo la variable "modificación_area" en un valor de 3 contornos para realizar los cálculos.

Al realizar un análisis de los datos que se obtiene por Matlab, el radio comienza sobredimensionado con respecto a los datos obtenidos por Python desde los 0,5 cm hasta los 4,2 cm coincidente con que la llama es laminar para ese rango de valores del radio. Cuando se comienzan a crecer las grietas en el frente de llama coincide con un corto periodo de tiempo donde el radio captado por Matlab se mantiene constante. Tras esta fase desde los 4,2 cm hasta los 5,7 cm, la celularidad en esta etapa ya se ha desarrollado por completo y no capta bien el programa de Matlab esa nueva fase por lo que subestima el radio para los últimos datos.

5.1.1.4. Sobredimensionamiento

El último de los fallos que se observa para al hacer uso del programa de Matlab para analizar los experimentos, se observa que el programa sobredimensiona para todos sus valores el radio captado para cada resta de fotogramas. Esto sucede para un solo para un experimento, el BC_E27.1.

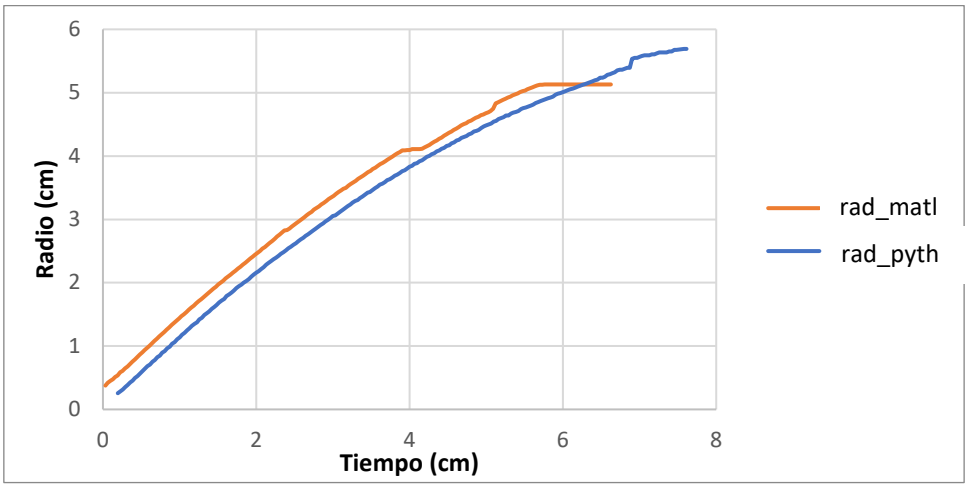


Figura 5.7: Comparación de radio obtenido por Matlab y Python para el experimento BC_E27.2 en la fase donde el radio en Matlab esta subestimado.

El experimento que se analiza con este problema tiene unas características donde desde el comienzo la llama es muy laminar, avanza muy lentamente hasta el final de combustión donde aparecen hacia la mitad unas grietas que no llegan a desarrollar la celularidad hasta instantes antes de tocar el frente de llama las paredes de la cámara de combustión.

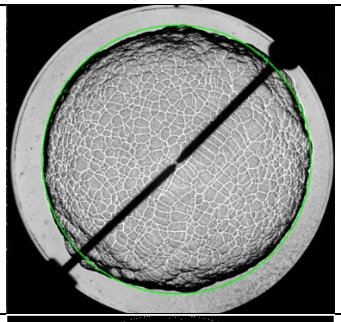
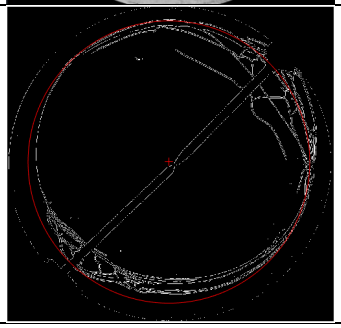
Radio (cm)	0,5 – 5,7
PYTHON	
MATLAB	

Figura 5.8: Fotogramas del experimento BC_E27.2 obtenidas por el programa de Python y Matlab en la fase de sobredimensionamiento del radio.

Posiblemente el programa de Matlab, al realizar la resta de fotogramas para obtener el radio del frente de llama, haya detectado mal algunos de los parámetros iniciales lo que genera que el radio se sobredimensione o bien que, al detectar 3 puntos para generar el radio, una vez realizada la resta de fotogramas, el rango con el que está previsto el programa este fuera del contorno del frente y los puntos que seleccione no coincidan ninguno con los del frente de llama.

Una vez conocidos los posibles fallos que se generan al usar el programa de Matlab y comprobar cómo se resuelven al usar el programa de Python, es necesario saber que al usar el programa desarrollado usando el lenguaje de programación Python también puede generar algunos fallos que se deben de conocer y observar cómo se pueden resolver. Es importante conocer que este tipo de errores se pueden dar en cualquiera de las combustiones que se analice usando el programa desarrollado en este trabajo.

Usando el programa de Python, se pueden obtener dos fallos:

5.1.1.5. Picos de sobredimensionamiento

En ciertos puntos del gráfico, el radio que se detecta por el programa para cada fotograma puede sobredimensionar el radio.

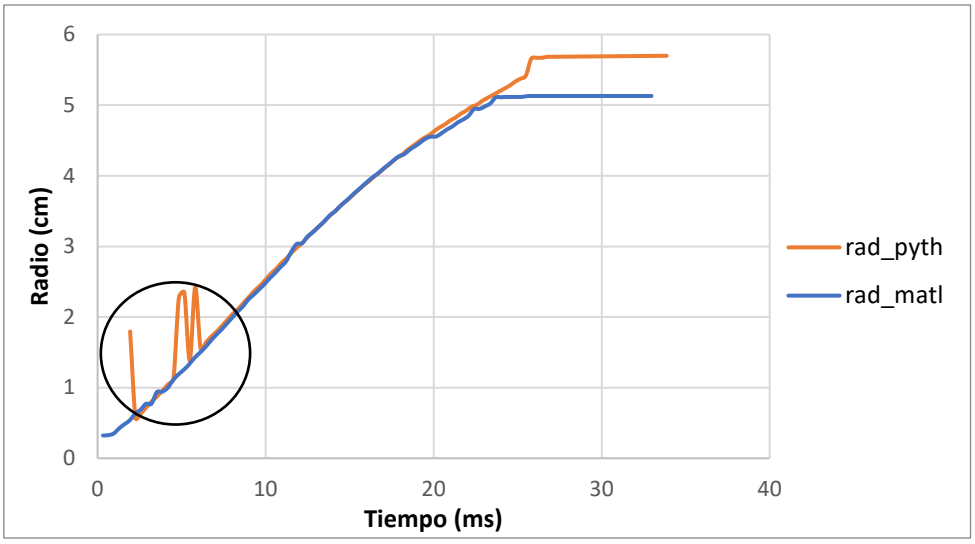


Figura 5.9: Comparación de radio obtenido por Matlab y Python para el experimento BC_E07

El error que se muestra en la *Figura 5.9* es uno de lo que más se producen en el análisis de los experimentos. Estos picos pueden aparecer al comienzo de la gráfica, hacia la mitad de la combustión o hacia el final. Estos picos surgen debido al valor que se asigna a la variable "modificacion_area" con la que se realizan se seleccionan el número de contornos que se van ha usar para calcular el radio del frente de llama. Para llamas muy celulares al comienzo de la combustión, si se seleccionan un número bajo de contornos se tienen más contornos de los necesarios y el radio se sobredimensiona para radios muy bajos. Para llamas muy laminares, si se selecciona un número alto de contornos, se tienen más de los que se necesitan y aparecen picos de sobredimensionamiento de datos a lo largo de toda la combustión.

Para cada combustión es importante seleccionar el valor apropiado de la variable "modificacion_area" para obtener el menor número de picos de sobredimensionamiento y así obtener mejores resultados.

5.1.1.6. Crecimiento final del radio cercano a las paredes de la cámara de combustión

Este tipo de error se produce instantes finales antes de llegar el frente de llama a la cámara de combustión.

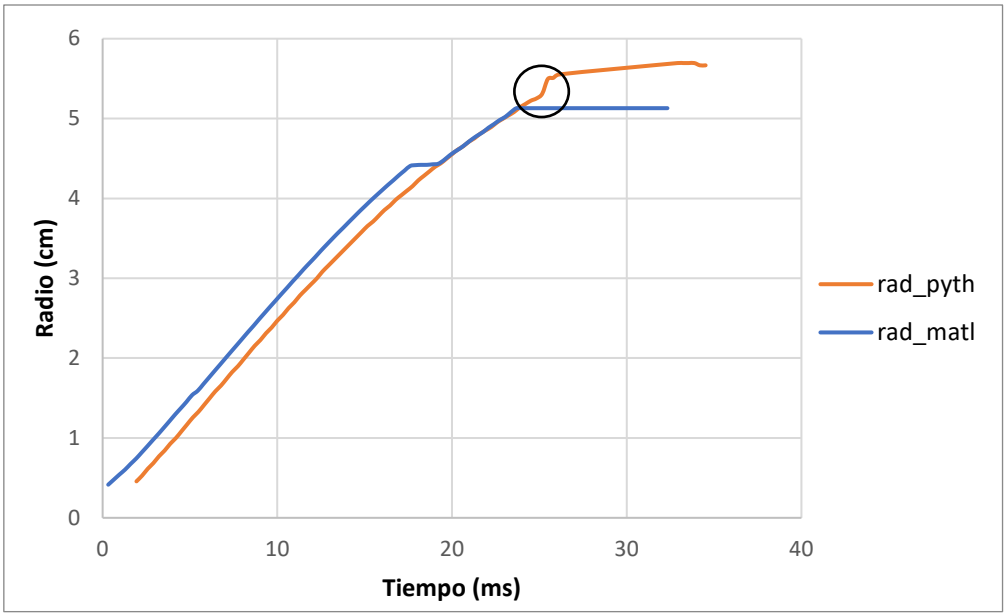


Figura 5.10: Comparación de radio obtenido por Matlab y Python para el experimento BC_E03.1.

Este error es común en experimentos muy laminares al comienzo de la combustión que no desarrollan muchas casi grietas ni rugosidades en el frente de llama, desarrollan la celularidad unos milisegundos antes de que el frente de llama llegue a las paredes de la cámara de combustión. El error que aparece es debido a que al desarrollar la celularidad al final el radio que se calcula en función del número de contornos asignado para esa combustión, se desvía ya que de manera repentina aparece en el frente de llama la celularidad y el programa no es capaz de calcular de manera correcta el radio y detecta directamente los contornos más grandes que generalmente coinciden con los de la cámara de combustión.

5.1.2. Resultados relacionados con la celularidad

Al igual que se han analizado los problemas que han surgido con la obtención del radio del frente de llama usando los dos programas, también se analizan las variaciones en los datos que se obtienen de los resultados referentes a la densidad celular. Dependiendo del experimento que se analice, los errores son más notorios con los datos obtenidos por Matlab, pero Python también genera una serie de errores que son necesarios conocer para poder resolverlos posteriormente. Los dos programas cumplen con la función de detección de la celularidad y la obtención de la densidad celular, aunque la forma de obtenerla en ambos programas sea de manera diferente.

Se han detectado una serie de errores que comete el programa de Matlab que el programa de Python resuelve y se han detectado una serie de errores aparecen en el análisis de Python que hay que considerar si se usa el programa para analizar el experimento.

El principal problema que presentan muchos de los resultados obtenidos al usar el programa de Matlab corresponde con un sobredimensionamiento de los datos recogidos.

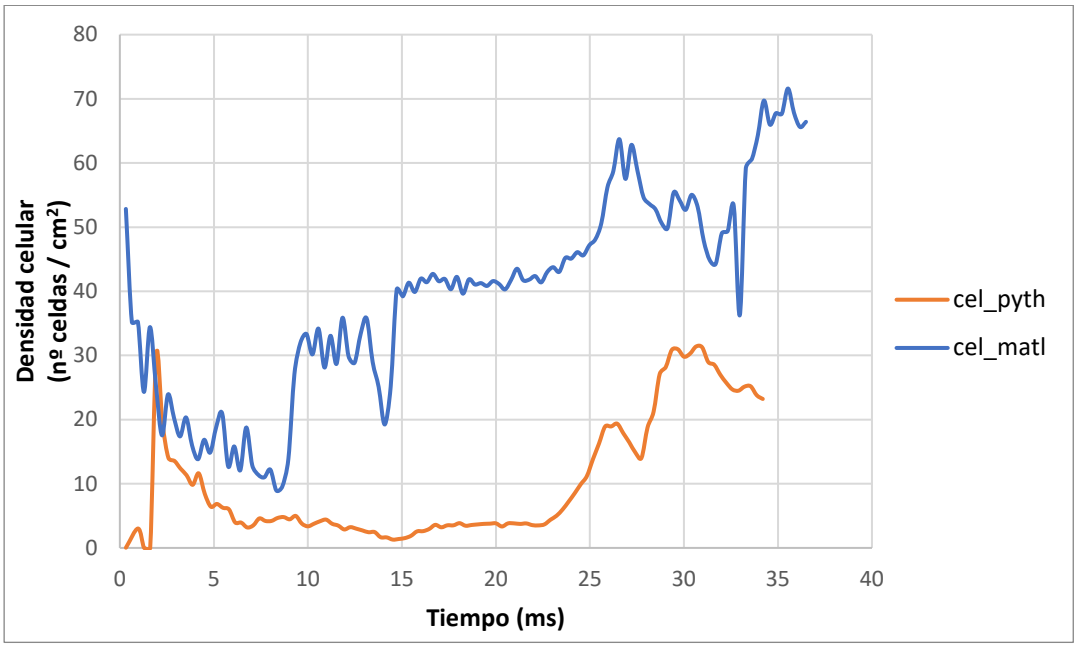


Figura 5.11: Comparación de la densidad celular obtenido por Matlab y Python para el experimento BC_E11.1.

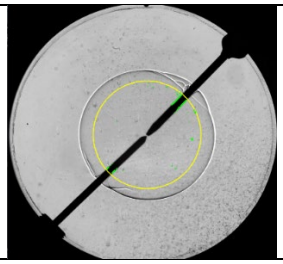
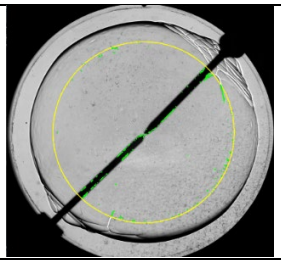
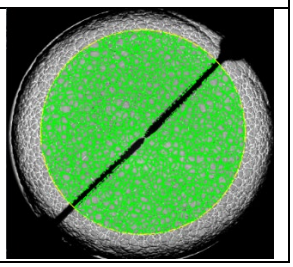
Tiempo (ms)	10	20	30
PYTHON			

Figura 5.12: Fotogramas del experimento BC_E11.1 obtenidas por el programa de Python para comprobar la evolución temporal de la celularidad.

Como se pueden observar en la *Figura 5.11*, la parte correspondiente a los resultados obtenidos al analizar el experimento usando el programa de Matlab sobredimensionan los datos representando datos que no son reales, se han sacado unas imágenes de unos instantes del proceso para comprobar para en qué instantes se desarrolla la celularidad. Como se puede comprobar por las *Figura 5.12*, la celularidad se desarrolla hacia el final de la combustión correspondiendo a un crecimiento en la gráfica de Python, mientras que el crecimiento que se muestra en la gráfica de Matlab también se representa, pero no está lo suficientemente reflejado. Para los instantes donde no se desarrolla celularidad, Python representa una gráfica casi plana interpretando que en esas zonas no hay celularidad que se deba de representar, mientras que Matlab al sobredimensionar los datos de

celularidad se puede interpretar que para esos instantes de tiempo si tiene celularidad el frente de llama cuando realmente no se está desarrollando en el frente de llama.

El problema que puede estar surgiendo en Matlab puede deberse a la forma que tiene el programa de interpretar la celularidad, entendiendo que el programa realiza un contraste entre la celda o pixel que analiza y las contiguas comparándolas para conocer qué datos se recogen en ellas. Si esos datos son similares entre las celdas, entonces lo interpreta como que hay celularidad y representa los datos sobredimensionados. El proceso que sigue Python es analizar cada pixel de manera independiente comprobando qué contornos contiene, si los tiene los representa y si no, representa valores próximos a cero en la gráfica.

Al igual que presenta problemas Matlab, Python también los presenta y se deben de tener en cuenta si se hace uso del programa para analizar los experimentos.

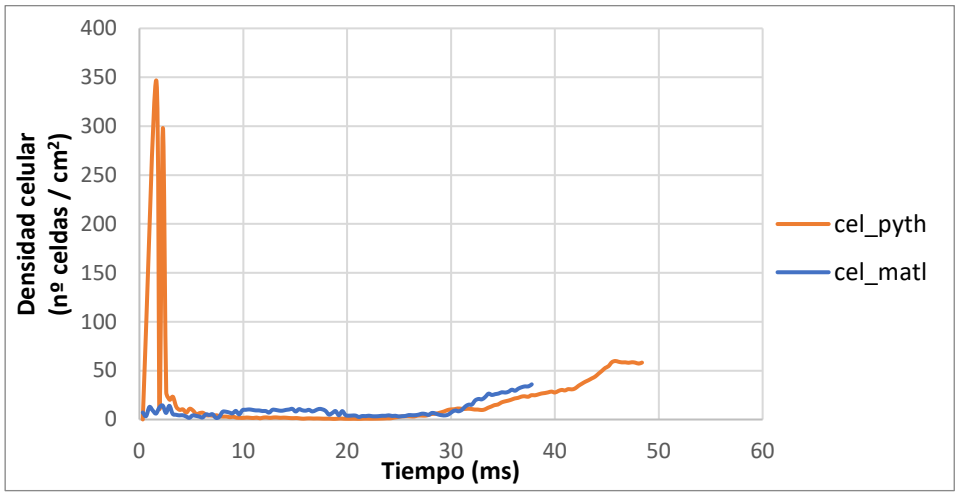


Figura 5.13: Comparación de la densidad celular obtenido por Matlab y Python para el experimento BC_E20.

El problema que presenta el programa de Python se produce en los primeros instantes en los que se desarrolla la combustión. Lo que sucede es que la llama al principio es muy pequeña como para que los contornos más grandes que encuentre el programa sean los correspondientes a esa llama, generalmente estos contornos corresponden con los de la cámara de combustión obteniendo así datos que no proporcionan información para el análisis de los resultados. Es decir, el programa tarda en converger hasta que los contornos correspondientes al frente de llama toman importancia como para que comenzar a representar datos que realmente corresponden a la parte que se desea analizar. Estos problemas de convergencia iniciales también los presenta el programa de Matlab con la diferencia de que generalmente son menos notorios.

5.2. Estudio de los resultados obtenidos con la modificación de la variable "modificación_area"

Otro de los análisis realizados para comprobar el correcto funcionamiento y la obtención de datos razonados del programa desarrollado, consiste en analizar uno de los parámetros de los que dependen el uso del programa, es la modificación, por parte del usuario, de la variable "modificacion_area". Como se ha explicado en punto del desarrollo del programa, esta variable se ha establecido antes del cálculo del número de contornos que se van a usar para realizar los cálculos para obtener el radio del frente y la densidad celular en cada fotograma del video.

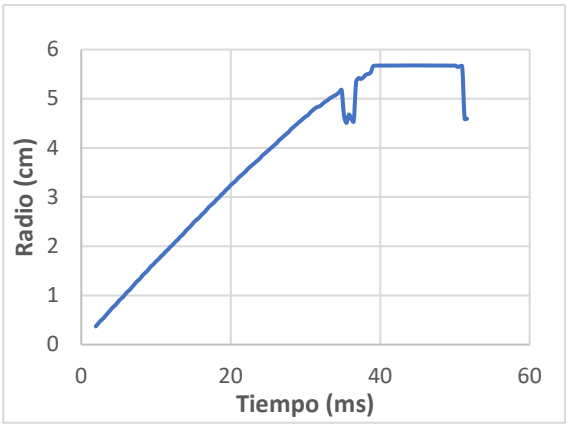
Los experimentos analizados en este apartado, tanto los del radio del frente de llama como los que analizan la densidad celular, son todos distintos. Esto permite analizar cómo se comportar la variable "modificacion_area" para distintos valores de los parámetros presión inicial y dosado (Fr).

Este es un dato que lo establece el usuario antes de ejecutar el programa, puesto que de él depende que se obtengan unos resultados más o menos exactos.

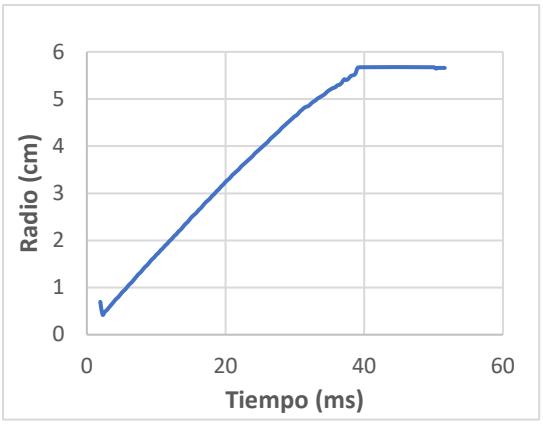
5.2.1. Resultados relacionados con el radio

En este punto del trabajo, se han analizado dos tipos de experimentos para comprobar qué valor de la variable "modificación_area" es mejor dependiendo de las condiciones observadas y establecidas previamente en el experimento. Para ambos experimentos se han obtenido los resultados establecidos como dato para la variable "modificación_area" los valores de 2, 3, 4, 5 y 6 contornos, este dato obtener el radio del frente de llama partiendo como base el número de contornos establecido.

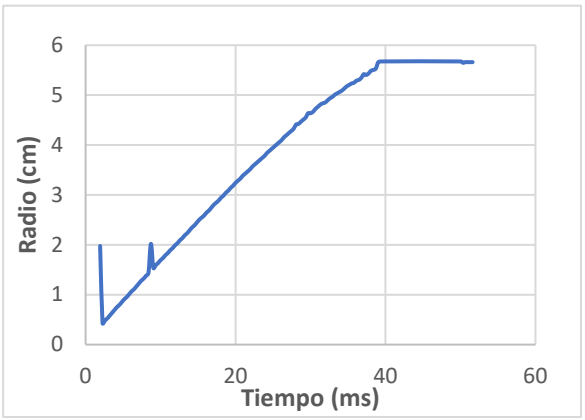
Uno de los experimentos analizados consiste en obtener los resultados para la aparición de una llama donde la celularidad aparece muy temprano en el frente. Las condiciones del experimento BC_E15 analizado son una presión inicial de 2.61 bar con un dosado (Fr) de 1.23. Para este experimento, se han obtenido los siguientes resultados de la evolución temporal del radio del frente de llama:



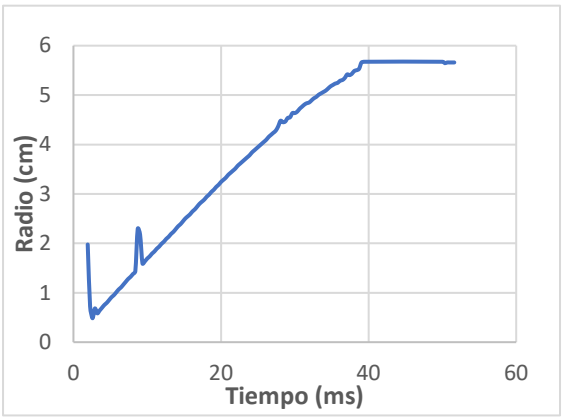
(a)



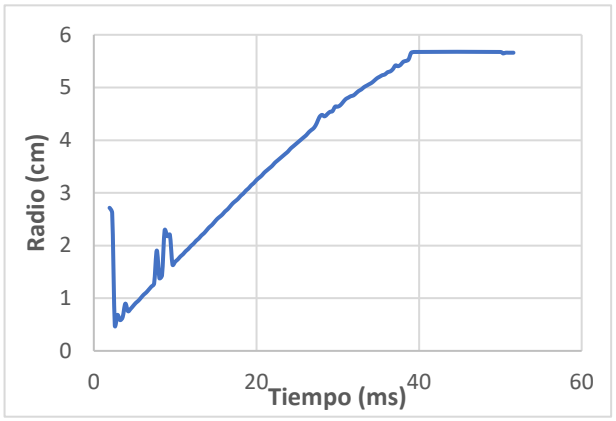
(b)



(c)



(d)



(e)

Figura 5.14: Obtención de la velocidad del frente de llama para el experimento BC_E15 (a) "modificacion_area" = 2 (b) "modificacion_area" = 3 (c) "modificacion_area" = 4 (d) "modificacion_area" = 5 (e) "modificacion_area" = 6

Una vez obtenidas todas las gráficas para el experimento BC_E15, se puede concluir que en la *Figura 5.14 (a)* los resultados muestran que la forma de calcular el radio estableciendo con el valor 2 la variable "modificación_area" genera unas desviaciones en la forma de captar el radio cuando el radio del frente de llama supera los 5 cm, al comienzo de la combustión el radio captado coincide a la perfección con el radio de la llama. Las desviaciones observadas en la progresión del radio

concluyen que capta dos contornos diferentes y más pequeños que los detectados en el radio del frente de llama, por lo que esos datos no serían válidos para obtener la progresión del frente no pudiendo utilizar los datos captados de los 5 cm a los 5.7 cm, esto se debe a que los contornos más grandes que está detectando el programa son más pequeños que los detectados en fotogramas anteriores y que no corresponden con los contornos reales del radio del frente de llama.

En la *Figura 5.14 (b)*, los resultados que se muestran en el gráfico captan una ligera desviación en el radio al comienzo de la combustión. Los datos obtenidos, posteriores a esa desviación, muestran la evolución correcta del avance del frente de llama. En comparación con la *Figura 5.14 (a)*, no existe subestimación de los datos del radio, esto es debido al aumentar el número de contornos con los que poder calcular de manera más aproxima el radio, el cálculo que realiza para el radio es más próximo al valor real del radio del frente por lo que no aparecen los picos alisando la curva en la zona de 5 cm a 5.7 cm.

En la *Figura 5.14 (c)*, la desviación que se muestra en la *Figura 5.14 (b)* es mayor, esto quiere decir, que, al realizar el cálculo con más contornos, el error que se comete aumenta. En este gráfico aparece otra desviación alrededor del valor del radio 1.5 cm.

En la *Figura 5.14 (d)*, en las zonas donde aparecen desviaciones en los gráficos anteriores hacen que la diferencia que hay crezca y de lugar a datos erróneos que impiden seguir el radio del frente de manera correcta. Además, aparecen ligeras desviaciones para valores del radio comprendidos entre 4 cm a los 5.7 cm, valor máximo para el cual se pueden recoger datos puesto que es el valor real de la cámara de combustión, esto indica que sigue el radio de manera correcta pero que da lugar a un pequeño sobredimensionamiento del mencionado.

En la *Figura 5.14 (e)*, todas las desviaciones captadas anteriormente magnifican el error que se comete por lo que aparecen numerosos picos de datos donde los contornos que se están detectando no son los correctos. El valor establecido para la variable “modificación_area” generaría unos datos con los que sería complicado trabajar para obtener de manera gráfica los datos para la velocidad del frente de llama puesto que para radios comprendidos entre 0.5 cm al 1.5 cm, radios al comienzo de la combustión, captan radios muy elevados para esos datos ya que al tener muchos contornos con los que trabajar para obtener el radio, el dato que se calcula no es real porque capta contornos importantes donde no se está produciendo la combustión o bien sobredimensiona en gran medida el radio captado.

Una vez analizados de manera individual todos los gráficos, se procede a superponer todos los gráficos para comprobar que las deviaciones captadas, tanto por subestimación del radio como por sobredimensionamientos del mismo, no afectan a la manera de captar el radio en aquellos puntos donde los gráficos aparecen de manera lisa.

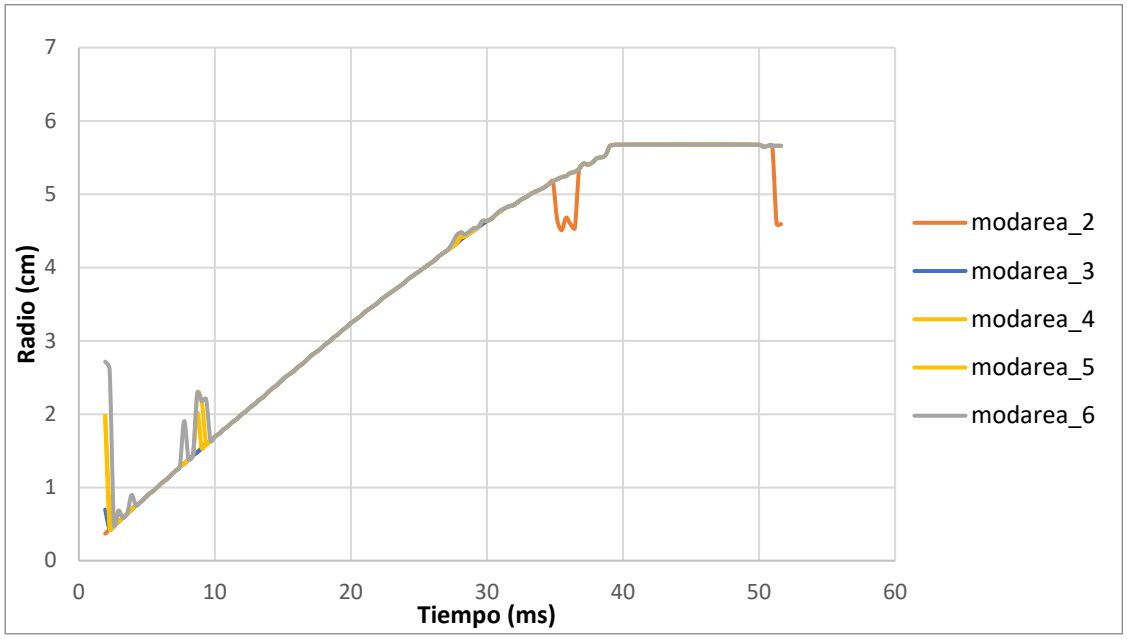


Figura 5.15: Obtención del gráfico conjunto donde se muestran los resultados obtenidos para la velocidad del frente de llama para los distintos valores de la variable "modificación_area" establecidos en el programa para el experimento BC_E15 para estableciendo como parámetros presión inicial = 2,61 bar y Fr = 1,23.

Observando la *Figura 5.15* donde se muestran todos los gráficos descritos anteriormente, se puede comprobar que, aunque se establezca un valor diferente de la variable "modificación_area" eso no modifica la pendiente de la recta, es decir, no modifica la velocidad de avance del frente de llama. La zona donde coinciden los cinco valores de la variable es entre los 1.5 cm a 5.1 cm, esto quiere decir, que en las zonas donde el frente de llama evoluciona a radios superiores, cuando pasa de un fotograma al siguiente, podríamos establecer el valor de la variable en cualquiera de los valores analizados.

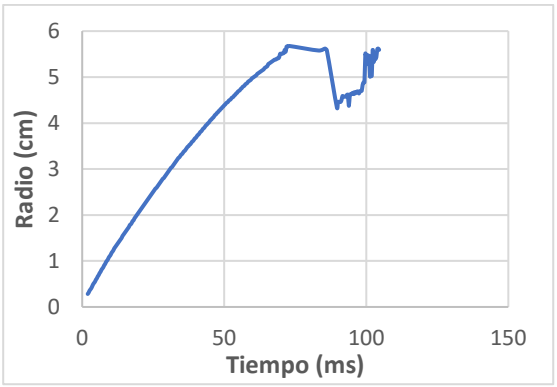
Si se pretendiera establecer valores diferentes a los analizados para la variable "modificación_area", es decir, si se pretendiera establecer su valor en 1 o en valores superiores a 6, los resultados que se van a obtener consisten en seguir realizando el cálculo del radio con más contornos que los que realmente se encuentran dentro del radio del frente de llama, por lo que los resultados obtenidos se sobredimensionarían todavía más en las zonas donde ya estaba esta condición o bien sobredimensionaría zonas donde anteriormente seguía el radio de manera correcta. Para el caso de captar solo un contorno, el programa no se compilaría y aparecería un mensaje de error informando que no se encuentran los contornos suficientes con los que poder realizar el cálculo puesto que no tiene el número mínimo de contornos, que, para el caso del radio, con o sin celularidad, es el valor de 2.

```
  File ~\anaconda3\Lib\site-
packages\spyder_kernels\py3compat.py:356 in
compat_exec
    exec(code, globals, locals)

  File e:\users\bilop\documents\universidad\tfg
mecánica\videos\videos_etanol2\celularidad_reduci
do.py:137
    radio = (diametro_mayor * diametro_menor) /
(diametro_mayor + diametro_menor)

ZeroDivisionError: float division by zero
```

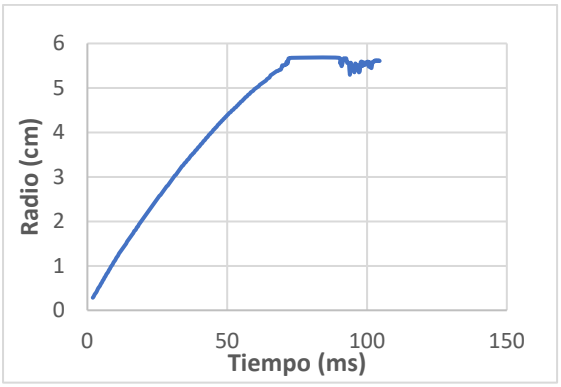
Figura 5.16: Error que se obtiene al establecer la variable "modificación_area" con un valor inferior a 2 contornos.

Para el experimento BC_E15 el valor más apropiado para establecer en la variable "modificación_area" es de 3, puesto que con este valor es con el que mejores resultados se han observado, *Figura 5.15*, en comparación con el resto de gráficos.

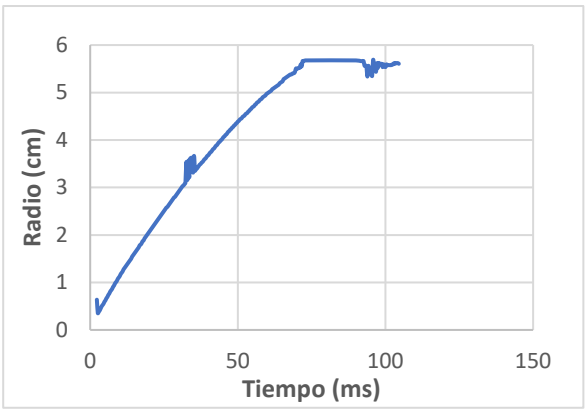
Uno de los experimentos analizados consiste en obtener los resultados para la aparición de una llama muy celular hasta el final de la combustión. Las condiciones del experimento analizado son una presión inicial de 1.92 bar con un dosado (Fr) de 1.4. Se ha obtenido los resultados para radio del frente de llama para analizar cómo afecta esta variable a la velocidad del frente de llama. Los resultados obtenidos para ese experimento son los siguientes:



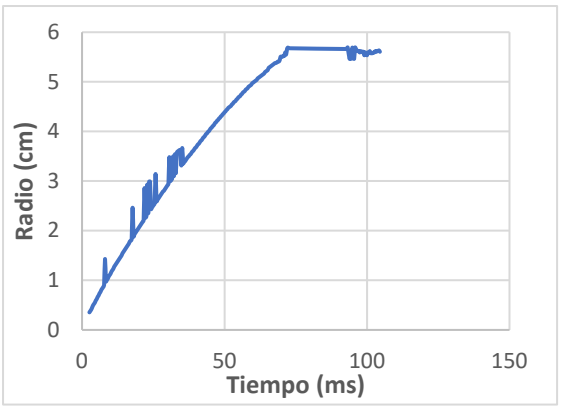
(a)



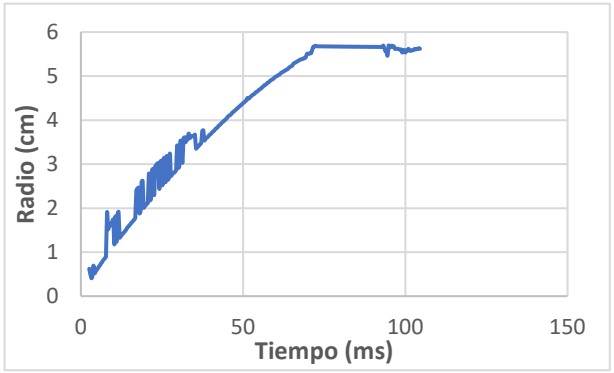
(b)



(c)



(d)



(e)

Figura 5.17: Obtención de la velocidad del frente de llama para el experimento BC_E28 (a) "modificacion_area" = 2 (b) "modificacion_area" = 3 (c) "modificacion_area" = 4 (d) "modificacion_area" = 5 (e) "modificacion_area" = 6

Una vez obtenidas todas las gráficas para el experimento BC_E28, se proceden analizar cada una de ellas de manera individual.

En la *Figuras 5.17 (a)*, los resultados generados al asignar a la variable "modificación_area" el valor 2 para que utilice para el análisis solo dicho número de contornos, muestran que el radio se obtiene de manera correcta desde el comienzo de la combustión desde la ignición hasta que adquiere el valor del radio de la cámara de combustión, 5,7 cm. Tras alcanzar el valor máximo, comienza a variar los contornos que capta el programa, produciendo una subestimación y dando lugar a desviaciones inferiores al valor que debería de corresponder, siendo en este caso el valor máximo. Esto se produce debido a que al final de combustión, cuando el frente de llama llega a las paredes de la cámara de combustión, no aparece frente de llama que captar puesto que no hay mezcla fresca que pueda seguir quemándose y ni la llama tiene más espacio para poder seguir expandiéndose lo que genera que se formen diversos contornos que el programa capta, y registra, pero no proporcionan información relevante para el cálculo de la velocidad del frente de llama en cada instante de tiempo.

En la *Figura 5.17 (b)*, los resultados que se muestran en el gráfico son muy similares a los expuestos en la *Figura 5.17 (a)*, lo que quiere decir que el programa sigue perfectamente el radio del frente de llama. Al igual que en la *Figura 5.17 (b)*, en la *Figura 5.17 (b)* al alcanzar las paredes de la cámara de combustión aparecen ligueras desviaciones siendo estas, cuando se le asigna el valor de 3 a la variable "modificación_area", menos pronunciadas que las registradas cuando se asigna el valor 2 a la variable. Sucede de manera similar que el caso anterior, las desviaciones producidas son debidas a que el frente de llama ha llegado a las paredes a la cámara de combustión provocando que se generen contornos de tamaños similares, cogido los 3 más grandes para el análisis, haciendo que el radio adquiera valores inferiores al valor de 5.7 cm del radio de la cámara de combustión, esta parte del gráfico debería ser plana puesto que al no haber frente de llama no tenemos velocidad del mismo.

En la *Figura 5.17 (c)*, comienzan aparecer desviaciones en el radio que detecta el programa debido a que utiliza más contornos para realizar el cálculo. Las desviaciones significativas que aparecen son al comienzo de la combustión, esto puede ser debido a que tiene que estabilizar el radio debido a que los contornos utilizados son similares entre ellos. El resto de desviaciones están comprendidas entre los 3 cm a los 4 cm siendo estas desviaciones puntuales, esto puede ser debido a que cercano al frente de llama aparare algún contorno más grande que obligue a realizar el cálculo del radio en otra posición. Las desviaciones desarrolladas se producen por sobredimensión del radio. Al llegar a la cámara de combustión, el programa detecta más contornos lo que permite ajustar mejor al valor real de la cámara de combustión aplanando la curva salvo en pequeños puntos donde se desvía el radio detectando uno menor al de la cámara de combustión.

En la *Figura 5.17 (d)*, las desviaciones que se producidas en los gráficos anteriores, se magnifican hasta valores más altos debido a que hay más contornos con los que el programa puede hacer el cálculo, lo que aumenta la probabilidad de que el radio se sobredimensione. Esto sucede al comienzo de la combustión con radios pequeños y al llegar a radios grandes el radio se estabiliza. Cuando llega al radio de la cámara de combustión, se aproxima mucho al valor real de la misma por lo que se estabiliza y amortigua la subestimación del radio aproximándose al valor del resto de datos.

En la *Figura 5.17 (e)*, para valores inferiores a 4 cm de radio del frente de llama, no se obtienen valores reales del frente de llama, por lo que no podríamos utilizar estos valores para realizar el cálculo adecuado de la velocidad del frente de llama en el rango comprendido desde los 0 cm a los 4 cm. Tras la zona de desvíos, aparece una zona donde sigue de manera adecuada el radio, por lo que esos valores podemos utilizarlos para calcular la velocidad del frente de llama entre los 4 cm a los 5.7 cm. Al llegar a las paredes de la cámara de combustión, el radio permanece prácticamente constante e igual a 5,7 cm coincidiendo con el valor real de la cámara de combustión.

Al igual que el análisis realizado para el experimento BC_E28, se procede a superponer todos los gráficos para comprobar las desviaciones que aparecen al asignar diferentes valores a la variable "modificación_area". Para este caso, también se analizan el sobredimensionamiento o la subestimación puesto que afecta de manera directa a los datos generados para el radio del frente de llama.

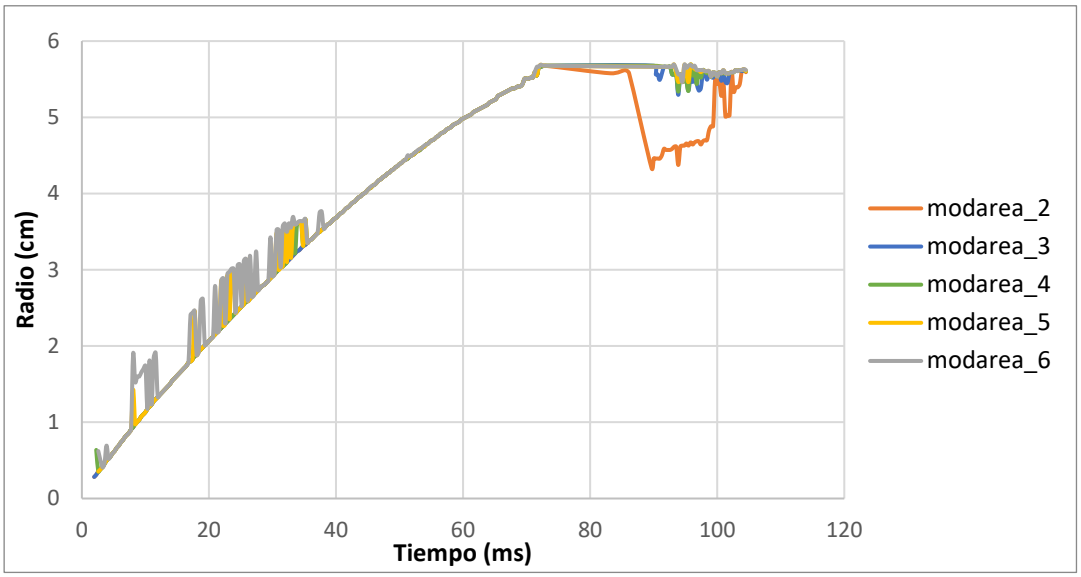


Figura 5.18: Obtención del gráfico conjunto donde se muestran los resultados obtenidos para la velocidad del frente de llama para los distintos valores de la variable "modificación_area" establecidos en el programa para el experimento BC_E28 para estableciendo como parámetros presión inicial = 1,92 bar y Fr = 1,4.

Tras analizar todos los gráficos de manera conjunta, se comprueba que a medida que se aumenta el valor asignado a la variable "modificación_area", es decir, se incrementa hasta llegar el valor de 6, el sobredimensionamiento del radio del frente para valores inferiores a 4 cm crece, no solo en cuando al número de puntos erróneos, sino también el valor calculado para ese dato, lo que hace inviable utilizar, para este experimento, valores superiores a 4. Por el contrario, cuando disminuye el valor asignado a la variable, los radios inferiores son correctos, lo que permite utilizarlos para calcular la velocidad del frente. Cuando se alcanzan el radio de la cámara de combustión, se produce una subestimación del radio, pero estos datos no son útiles para calcular la velocidad del frente.

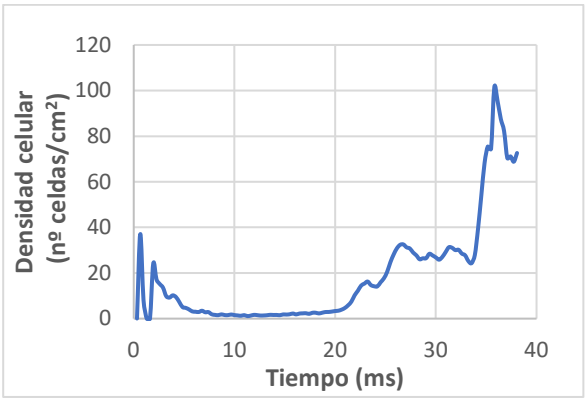
Al superponer las curvas, todas ellas coinciden, esto quiere decir, a pesar de que haya ciertos valores donde se sobredimensiona o subestima el radio la pendiente de la recta, coincide con el valor de la velocidad del frente de llama, no cambia, aunque se modifique el valor de la variable "modificación_area".

Para poder obtener una velocidad útil del frente de llama en el experimento BC_E28 analizado, es necesario asignar valores pequeños, como 2 o 3, a la variable "modificación_area".

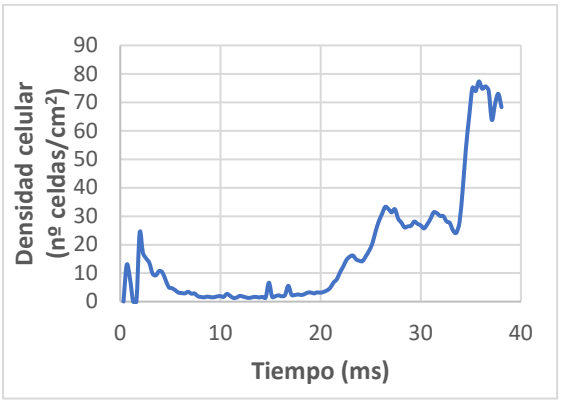
5.2.2. Resultados relacionados con la celularidad

Al igual que para el cálculo del radio del frente de llama, se han analizado dos tipos de experimentos donde se modifica el valor de la variable "modificación_area" para comprobar que parámetro es el mejor para obtener los resultados más óptimos para el ensayo analizado. Para comprobar únicamente la influencia de la variable, se modifica el parámetro para un experimento donde se mantienen las mismas condiciones de presión inicial y dosado en cada iteración. Para ambos experimentos se han analizado en este apartado se han asignado valores a la variable "modificación_area" correspondientes a 3, 4, 5 y 6 contornos para realizar el cálculo.

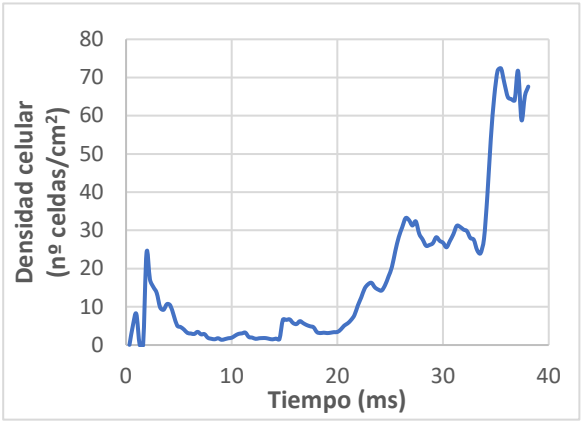
Uno de los experimentos analizados consiste en obtener los resultados para una llama laminar donde la celularidad aparece hacia el final de la combustión casi al tocar las paredes de la cámara de combustión. Las condiciones del experimento BC_E09.1 analizado son una presión inicial de 1.95 bar con un dosado (Fr) de 1.01. Se ha obtenido los resultados para la densidad celular frente al tiempo para comprobar cómo afecta la variable "modificación_area" a los parámetros encargados de recoger los datos correspondientes a la celularidad. Los resultados obtenidos para ese experimento son los siguientes:



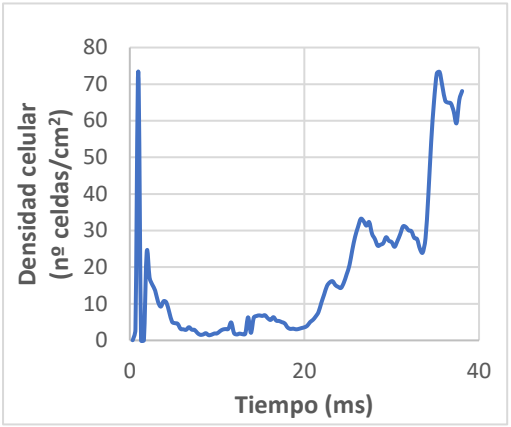
(a)



(b)



(c)



(d)

Figura 5.19: Obtención de la densidad celular para el experimento BC_E09.1
(a) "modificacion_area" = 3 (b) "modificacion_area" = 4 (c) "modificacion_area" = 5
(d) "modificacion_area" = 6

Una vez obtenidas todas las gráficas para el experimento BC_E09.1, se proceden analizar cada una de ellas de manera individual.

En la *Figuras 5.19 (a)*, los resultados generados al asignar a la variable "modificación_area" el valor 3 para que el programa analice solo esta cantidad de contornos para obtener los datos de la densidad celular, muestran que la gráfica es bastante plana al comienzo a excepción de los primeros datos donde se sobredimensiona el valor de la densidad celular debido a que como el programa detecta solo contornos le cuesta converger. Tras la zona de sobredimensionamiento, la curva se aplana lo que implica que en esa zona la llama es celular. Pasados los 20 s, se produce un cambio en la gráfica indicando que está comenzando a aparecer celularidad y el programa detecta más elementos dentro del fotograma. En esta zona los datos recogidos no tienen zonas de sobredimensionamiento, siendo estos los datos que se desvían en gran medida respecto de los de su alrededor. Antes de los 40 s, el valor de la densidad celular comienza a decaer, esto se debe a que para esos instantes de tiempo la combustión ya ha llegado a las paredes de la combustión y la detección de los parámetros comienza a no ser precisa y subestima los datos que detecta.

En la *Figura 5.19 (b)*, al comienzo de la combustión sucede lo mismo que ocurre en la *Figura 5.19 (a)* donde al no converger los resultados no podemos considerarlos como precisos para considerarlos dentro de estudios posteriores. En esta zona se sobredimensionan los mismos datos que en la *Figura 5.19 (b)* pero en mayor valor ya que el programa tiene más contornos para realizar los cálculos, por lo que se genera más imprecisión en los datos representados. Pasados los 20 s, comienza la zona donde si convergen los datos, para la *Figura 5.19 (a)* esta zona era completamente plana, para la *Figura 5.19 (b)* aparecen perturbaciones ya que son menos precisos que los de la figura anterior llegando incluso aparecer dos puntos donde se sobredimensiona el radio. Cuando aparece la celularidad aparecen zonas donde se subestima la densidad celular. En los últimos datos del gráfico donde la combustión ya está en las paredes de la cámara de combustión, los datos que en el gráfico anterior se subestimaban, se siguen subestimando, pero con un valor inferior ya que tenemos más contornos para obtener más exacto el valor.

En la *Figura 5.19 (c)*, sucede lo mismo que en los dos gráficos explicados anteriormente, para el comienzo de la combustión se sobredimensionan los datos, pero de manera más excesiva que en las otras dos gráficas anteriores puesto que el valor de la variable "modificacion_area" es de 5 contornos, lo que genera que para este tipo de llama muy laminar durante toda la combustión sean demasiados contornos. En la zona donde la celularidad no ha aparecido todavía, el programa detecta elementos que no pertenecen a la combustión generando que sobredimensionen los datos. Hacia el final de la combustión los datos que recoge

donde para los gráficos anteriores se subestimaban más los datos, para este valor de la variable, se subestiman menos datos ya que tiene más contornos con los que poder los que el programa puede trabajar ajustándolo a valores más próximos a los de los datos del antes de llegar a las paredes de la cámara de combustión.

En la *Figura 5.19 (d)*, se comprueba que en las zonas donde se sobredimensionaba la detección de contornos, se magnifican más los resultados, es decir, al comienzo de la combustión, los valores son más grandes que los recogidos en las gráficas anteriores. En las zonas donde todavía no se desarrolla la celularidad también aparecen datos sobredimensionados ya que se utilizan más contornos para realizar el cálculo con lo que el programa detecta elementos que no corresponden a la celularidad. Por último, en la zona donde la combustión llega a las paredes de la cámara, al tener más contornos con lo que poder detectar y calcular la densidad celular, se reducen los datos que se están subestimando y los que siguen apareciendo en la gráfica reducen su valor.

Una vez analizadas de manera individual todas las gráficas, se procede a superponer todas las gráficas para el experimento BC_E09.1 para comprobar las zonas donde se producen desviaciones y su comportamiento al modificar la variable "modificacion_area", analizando los sobredimensionamientos y las subestimaciones que se producen y cómo afectan a la densidad celular.

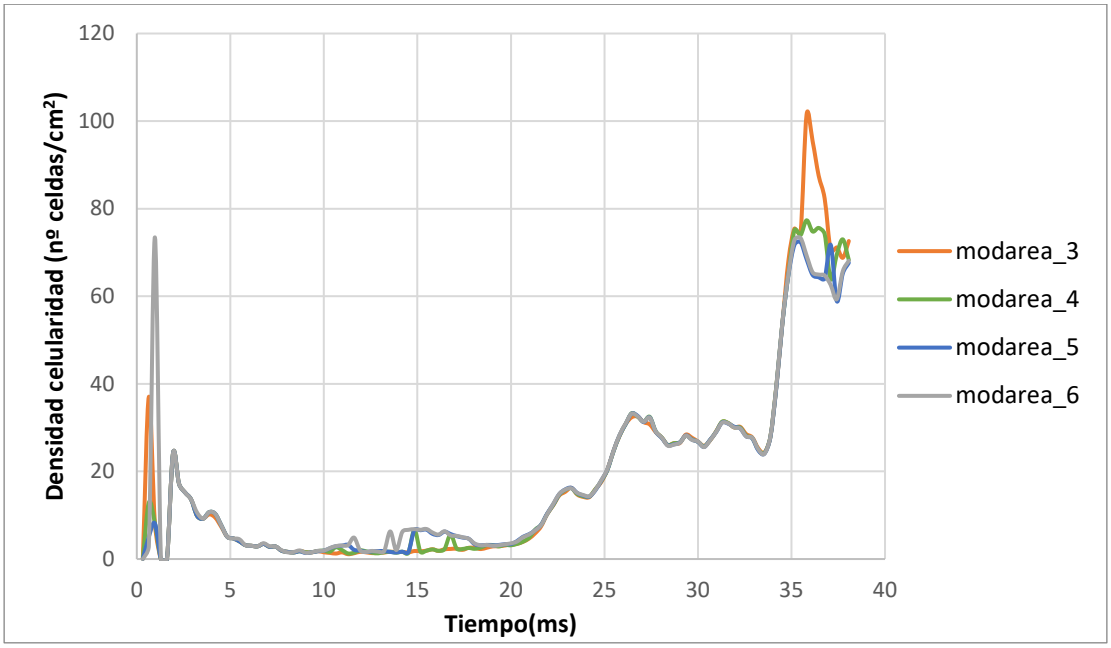


Figura 5.20: Obtención del gráfico donde se muestran los resultados para los distintos valores de la variable "modificación_area" establecidos en el programa para el experimento BC_E09.1 para estableciendo como parámetros presión inicial = 1,95 bar y Fr = 1,01.

Al observar la *Figura 5.20*, se comprueba que para cualquiera de los valores que se establezcan para la variable "modificacion_area", el programa sigue en las mismas

condiciones todo el video. A medida que se aumenta el valor de la variable "modificacion_area" disminuye el valor de la densidad celular para los datos en los primeros instantes de la combustión. Por el contrario, a medida que se aumenta el valor de la variable, se sobredimensionan los datos cuando entre los 10 s a 20 s. En los valores finales de la combustión, al aumentar la variable, los valores se comienzan a estabilizar más, conociendo que estos datos son ya referentes a lo que sucede cuando el frente de llama está en las paredes de la cámara de combustión, para valores bajos de la variable, 3 contornos, estos datos aparecen sobredimensionados mientras que, para los valores de 5 contornos y 6 contornos, se subestiman los datos.

Con respecto a la aparición de la celularidad para este experimento, comienza a tener notoriedad a partir de los 20 s donde la densidad celular comienza a crecer de manera rápida y lo hace hasta que la llama llega a las paredes de la cámara de combustión.

Para valores inferiores de 3 contornos para la variable "modificacion_area" el programa no dispone de contornos suficientes como para realizar los cálculos y muestra un error de ejecución en la consola, como se observa en la *Figura 5.16*. No es recomendable tomar más de 6 contornos para el valor de la variable ya que el programa dispone de numerosos contornos con los que realizar los cálculos y esto permite que se puedan sobredimensionar o subestimar los datos en partes de la combustión captando elementos que no tienen relevancia para los estudios.

Para poder estudiar el experimento BC_E09.1 analizado es recomendable asignar el valor 4 a la variable "modificacion_area" puesto que con ella es con los que mejores resultados se han obtenido.

Otro de los experimentos analizados consiste en obtener los resultados para una llama donde la celularidad aparece hacia la mitad de la combustión, pero desde la ignición ya aparecen grietas sobre el frente de llama. Las condiciones del experimento BC_E17.1 analizado son una presión inicial de 2.62 bar con un dosado (Fr) de 1.24. Se ha obtenido los resultados para la densidad celular frente al tiempo para comprobar cómo afecta la variable "modificación_area" a los parámetros encargados de recoger los datos correspondientes a la celularidad. Los resultados obtenidos para ese experimento son los siguientes:

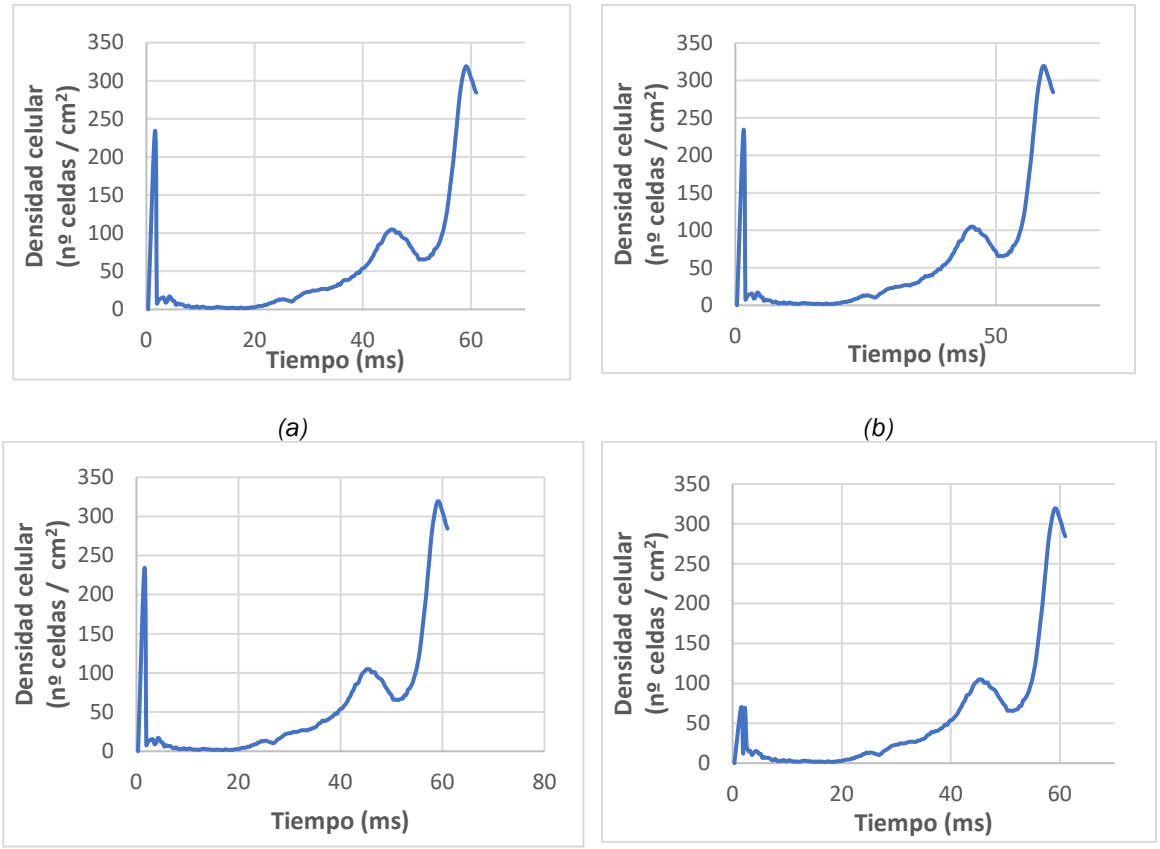


Figura 5.21: Obtención de la densidad celular para el experimento BC_E17.1
(a) "modificacion_area" = 3 (b) "modificacion_area" = 4 (c) "modificacion_area" = 5
(d) "modificacion_area" = 6

Una vez obtenidas todas las gráficas para el experimento BC_E17.1, se proceden analizar cada una de ellas de manera individual.

Con respecto a los análisis realizados en los puntos anteriores, los datos recogidos en las *Figuras 5.21 (a), (b) y (c)* son muy similares. Al comienzo de estos gráficos se sobredimensiona uno de los datos, donde el valor de la densidad celular coincide en el mismo valor para los 3 gráficos, esto se debe a que al comienzo de la combustión el programa tarda en converger hasta que detecta el comienzo del frente de llama de la combustión. El resto de la combustión transcurre sin zonas donde se sobredimensione o se subestime datos que influyan en el cálculo de la densidad celular.

En la *Figura 5.21 (d)*, la zona donde para los otros gráficos se sobredimensiona uno de los valores de la densidad celular, sigue ocurriendo, pero el valor que adquiere es mucho menor que para los gráficos anteriores desarrollados. Además, aparece un segundo punto, cercano al ya explicado, donde se sobredimensiona el valor para de la densidad celular, adquiriendo un valor similar al anterior. La combustión se desarrolla adquiriendo valores similares a los de las *Figuras 5.21 (a), (b) y (c)*.

Una vez analizadas de manera individual todas las gráficas, se procede a superponer todas las gráficas para el experimento BC_E17.1 para comprobar las zonas donde se producen desviaciones y su comportamiento al modificar la variable "modificacion_area", analizando los sobredimensionamientos y las subestimaciones que se producen y cómo afectan a la densidad celular.

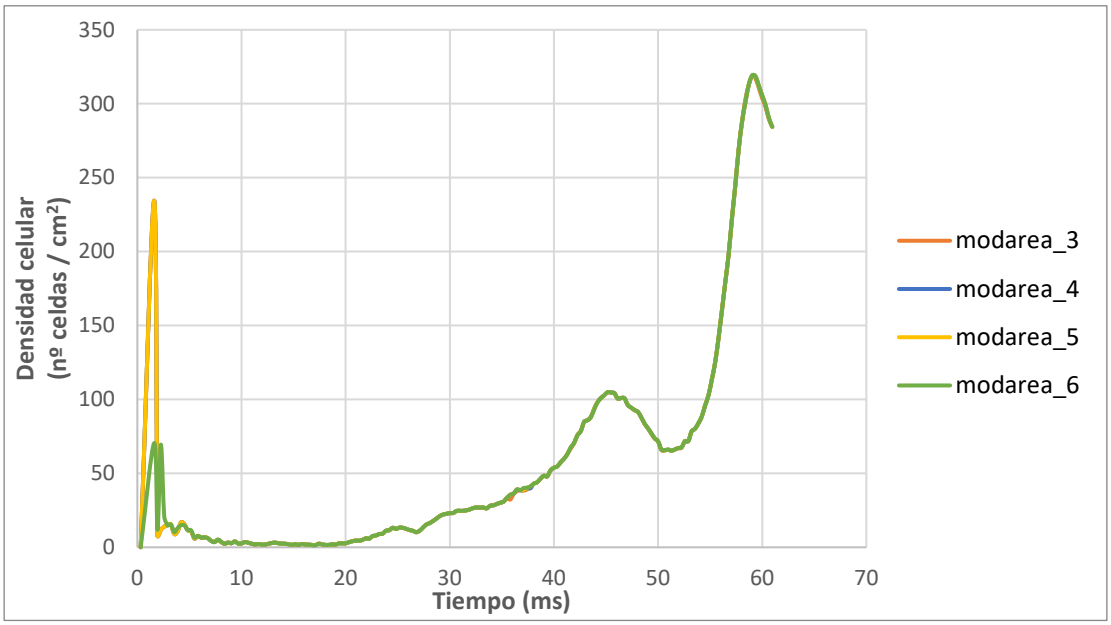


Figura 5.22: Obtención del gráfico donde se muestran los resultados para los distintos valores de la variable "modificación_area" establecidos en el programa para el experimento BC_E17.1 para estableciendo como parámetros presión inicial = 2,62 bar y Fr = 1,24.

Al observar la *Figura 5.22*, se comprueba que cuando se asigna el valor de 3, 4 o 5 a la variable "modificacion_area", para este experimento, los gráficos coinciden en todos los valores que adquiere la densidad celular para cada instante de tiempo lo que indica que al ser un experimento donde tiene celularidad desde el comienzo, no influye la cantidad de contornos que se asignan para realizar el cálculo. La única diferencia que se observa en todos los gráficos sucede cuando se asigna a la variable "modificacion_area" el valor de 6 contornos ya que, el primer dato donde no ha convergido el programa todavía tiene un valor inferior al presentado en los otros gráficos. El resto del gráfico coincide e valores con el resto de gráficos planteados hasta el final de la combustión.

Con respecto a la aparición de la celularidad para este experimento, comienza a tener notoriedad a partir de los 20 s donde la densidad celular comienza a crecer de forma progresiva hasta que la densidad celular engloba todo el frente de llama. Al llegar a las paredes de la cámara de combustión, los datos representados indican que hay numerosos contornos, pero estos datos no son relevantes para analizar lo que sucede dentro de la combustión.

Para valores inferiores de 3 contornos para la variable "modificacion_area" el programa no dispone de contornos suficientes como para realizar los cálculos y

muestra un error de ejecución en la consola como se muestra en la *Figura 5.42*. No es recomendable tomar más de 6 contornos para el valor de la variable ya que el programa dispone de numerosos contornos con los que realizar los cálculos y esto permite que se puedan sobredimensionar o subestimar los datos en partes de la combustión captando elementos que no tienen relevancia para los estudios.

Para poder estudiar el experimento BC_E17.1 analizado es recomendable asignar el valor 6 a la variable "modificacion_area" puesto que con ella es donde se han reducido el valor de los primeros datos en la combustión.

Como conclusión general tras analizar el comportamiento de la variable "modificacion_area" para los dos tipos de celularidad que se ha estudiado, para llamas donde la celularidad sea muy laminar y para llamas donde la celularidad aparezca muy pronto en la combustión, la variable se debe establecer en valores bajos para llamas laminares, entre los 3 y 4 contornos de estudio, ya que durante el proceso van aparecer más grietas y la celularidad se desarrolla llegando a las paredes de la cámara de combustión siendo la densidad celular no tan elevada como en las llamas más celulares. Para llamas donde la celularidad se desarrolla más temprano, es recomendable usar un mayor número de contornos, entre los 5 y 6 contornos, permitiendo una mayor definición del radio para estabilizarlo y permitiendo detectar mejor la evolución de la densidad celular ya que sobre el frente de llama se dividen en más células en cada instante.

5.3. Estudio de la influencia del dosado

En este apartado se realiza un estudio para comprobar los resultados obtenidos al ejecutar el programa. Para poder comparar los resultados obtenidos con otros, se deben de mantener una serie de parámetros iguales en todos ellos, siendo estos la presión inicial del experimento y la variable "modificación_area" que establece el número de áreas de contornos va a tomar para analizar el video completo.

Una vez establecidos todos los parámetros iguales en todos los experimentos, se obtienen los resultados que permiten comparar como influye el dosado en la velocidad del frente de llama, afectado en ello a la evolución del radio del frente de llama, a la aparición de la celularidad comprobando en que punto aparece en cada experimento y cómo evolucionan por tanto la densidad celular frente a la evolución del radio.

Para obtener un análisis adecuado de los experimentos, se han seleccionados tres presiones iniciales diferentes. Dentro de esas presiones inciales se han seleccionado experimentos que tengan diferentes dosados, lo que permite hacer una comparativa posterior para comprobar cómo afecta en el radio del frente de llama, en la densidad celular que se obtiene para cada experimento y en la evolución de la densidad celular para cada valor del radio del frente de llama.

El primero de los estudios consiste en analizar el comportamiento del radio del frente de llama. Se busca analizar cómo afecta la evolución del radio para distintos dosados y manteniendo constante la presión inicial y la variable “modificacion_area” para todos los experimentos analizados. Se han seleccionado varios experimentos, con diferentes dosados, dentro de tres presiones iniciales distintas.

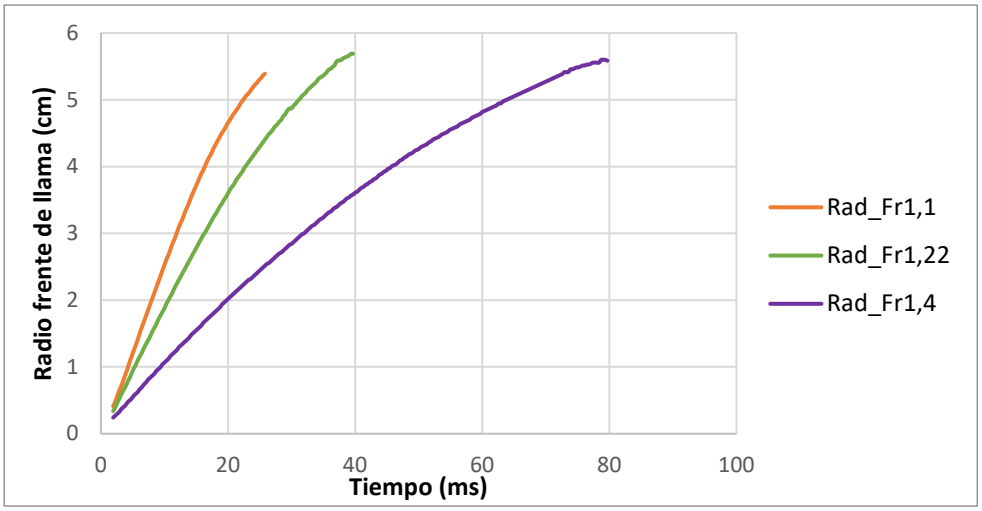


Figura 5.23: Gráfico comparativo del radio del frente de llama frente al tiempo para distintos valores del dosado estableciendo como parámetros fijos una presión inicial de 1,5 bar y variable “modificacion_area” = 2.

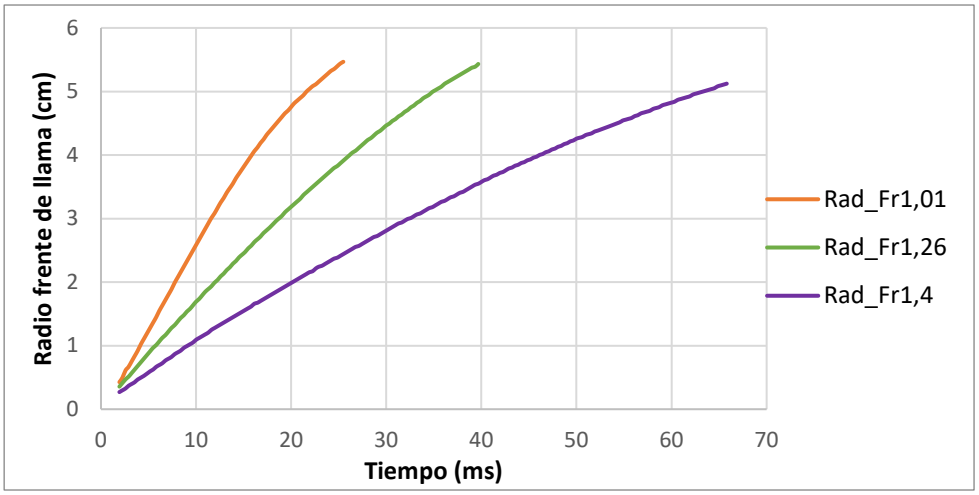


Figura 5.24: Gráfico comparativo del radio del frente de llama frente al tiempo para distintos valores del dosado estableciendo como parámetros fijos una presión inicial de 1,95 bar y variable “modificacion_area” = 3.

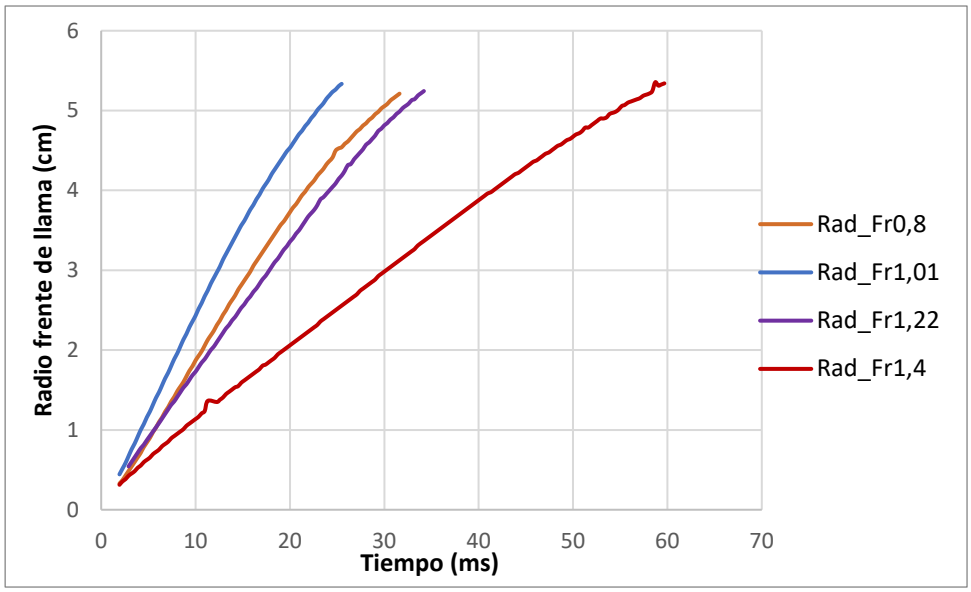


Figura 5.25: Gráfico comparativo del radio del frente de llama frente al tiempo para distintos valores del dosado estableciendo como parámetros fijos una presión inicial de 3 bar y variable "modificacion_area"= 4.

Observando las figuras anteriores, se pueden llegar a una serie de conclusiones referentes al comportamiento del radio del frente de llama bajo distintos dosados.

Para dosados próximos al estequiométrico, la velocidad del frente de llama, obtenida en función del radio del frente y el tiempo en que se produce la combustión, crece de rápidamente hasta las paredes de la cámara de combustión puesto que el frente de llama evoluciona de manera rápida, la mezcla fresca disminuye rápidamente y aumentan los productos de la combustión. Para presiones iniciales bajas, es decir, para 1,5 bar y 1,95 bar, la pendiente de las rectas que dan lugar a la velocidad del frente es más pronunciada, esto quiere decir, que pasa dosados cercanos al estequiométrico y presiones bajas, la velocidad del frente de llama es más rápida y se ejecuta la combustión en un menor tiempo.

Para dosados pobres, es decir, dosados inferiores al estequiométrico, como el que se observa en la *Figura 5.25* correspondiente a 0,8 o para dosados ricos, es decir, dosados superiores al estequiométrico, como pueden ser para 1,22 o 1,26, la tendencia observada para la velocidad del frente de llama es que son más lentos que los dosados próximos al estequiométrico independientemente de la presión inicial que tengan. A medida que se aumenta la presión inicial provoca que la reacción se desarrolle de manera más rápida en periodos de tiempo más cortos.

Para dosados muy ricos con valores de 1,4, al aumentar la presión inicial de la reacción, la velocidad a la que se produce la combustión se desarrolla de manera más rápida que para presiones más bajas. Para una presión inicial de 1,5 bar la reacción se desarrolla en 80 ms, para una presión de 1,95 bar el tiempo pasa a ser de 65 ms mientras que para una presión de 3 bar el tiempo es de 60 ms. Aun

disminuyendo el tiempo en que se produce la reacción, no se aproxima a los valores obtenidos cuando el dosado es próximo al estequiométrico donde la reacción se desarrolla entre los 23 a 26 ms.

Una vez analizado cómo se comporta la velocidad del frente de llama cuando se analiza para diferentes dosado, se procede a realizar un estudio similar donde se analiza el comportamiento de la densidad celular frente al tiempo. Se busca analizar cómo se desarrolla la celularidad para diferentes dosados, mantenido constante la presión inicial y el valor de la variable "modificacion_area" para cada análisis.

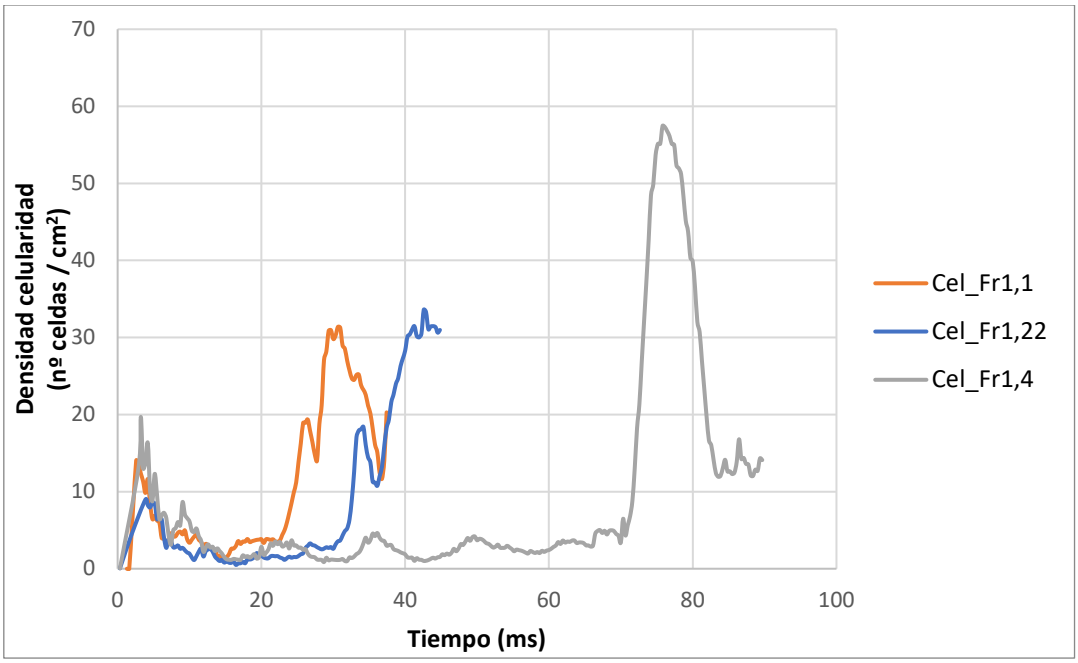


Figura 5.26: Gráfico comparativo de la densidad celular frente al tiempo para distintos valores del dosado estableciendo como parámetros fijos una presión inicial de 1,5 bar y variable "modificacion_area" = 2.

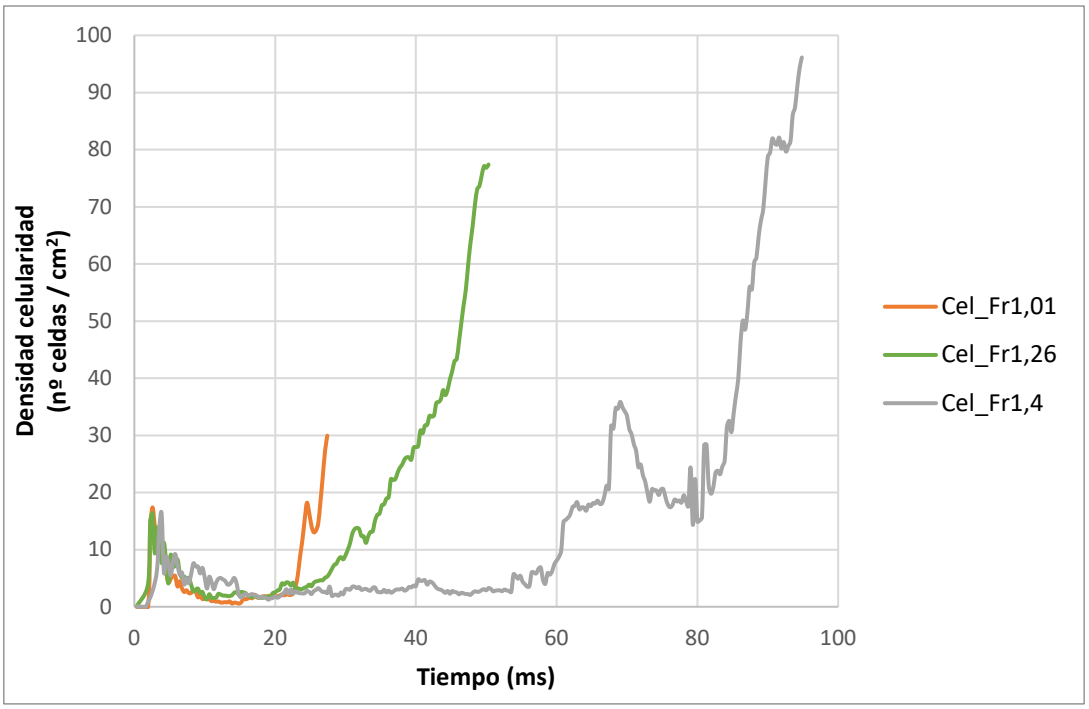


Figura 5.27: Gráfico comparativo de la densidad celular frente al tiempo para distintos valores del dosado estableciendo como parámetros fijos una presión inicial de 1,95 bar y variable "modificacion_area" = 3.

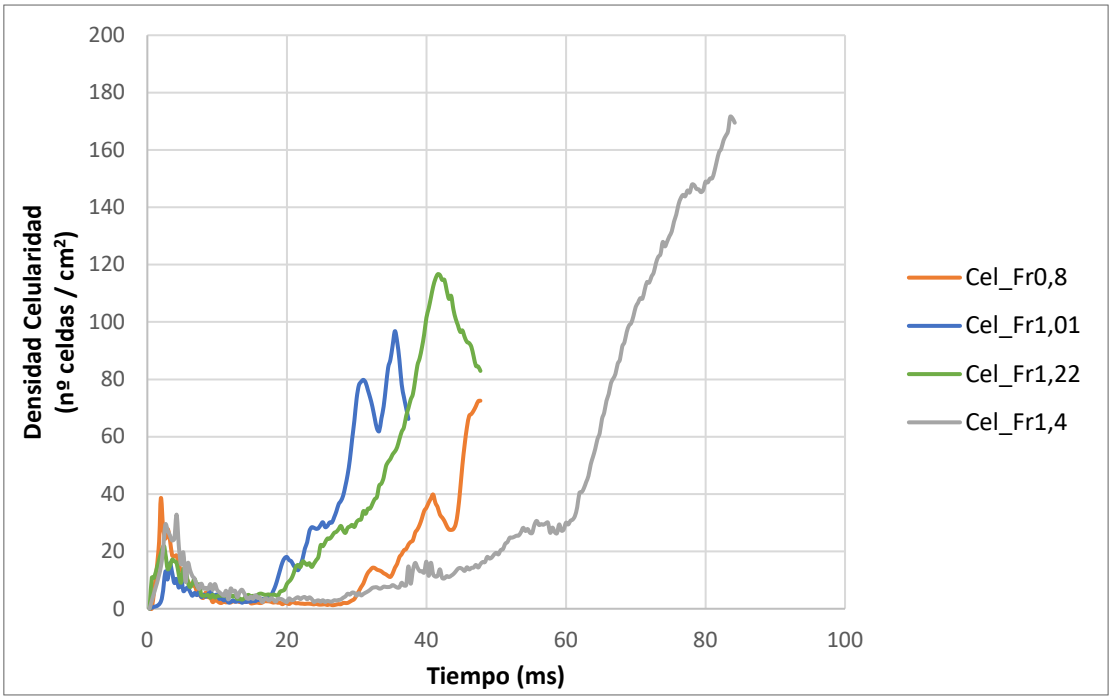


Figura 5.28: Gráfico comparativo de la densidad celular frente al tiempo para distintos valores del dosado estableciendo como parámetros fijos una presión inicial de 3 bar y variable "modificacion_area"= 4.

Con las gráficas representadas, se puede observar el comportamiento del parámetro de densidad celular bajo distintos valores del dosado.

Para presiones iniciales altas, para 3 bar, la celularidad se desarrolla más lentamente para cualquier dosado frente a que para bajas presiones iniciales, para 1,5 bar y 1,95 bar. Esto se debe a que sobre el frente de llama aparecen grietas que retrasan el crecimiento del radio y retrasando la aparición de la celularidad. Cuando las grietas alcanzan tamaños considerables, rompen la tensión del frente de llama y aparece la celularidad. Se puede considerar a este tipo de celularidad como progresiva.

Para dosados próximos al estequiométrico, la combustión es más rápida por lo que la celularidad crece antes. Para presiones iniciales bajas, 1,5 bar, cuando se desarrolla la celularidad lo hace prácticamente de manera instantánea, es decir, que para los primeros instantes de la combustión las llamas son muy laminares y cuando el radio está próximo a las paredes de la cámara de combustión aparece la celularidad repartida en todo el frente, sin grietas previas. Al aumentar la presión inicial, la aparición de la celularidad no se desarrolla de forma espontánea, sino que con el crecimiento del frente de llama aparecen grietas que van aumentando con el frente hasta que aparece la celularidad.

También se observa que a medida que se aumenta el valor de la presión inicial, cuando aparece la celularidad en el frente de llama se registra un mayor aumento de densidad celular, ocurre para todos los dosados elegidos para el estudio en esa presión.

Para dosados muy altos, 1,4, las combustiones son muy lentas, se observa para las tres presiones iniciales elegidas, y desarrollan la celularidad instantes antes de que el frente de llama llegue a las paredes de la cámara de combustión. Tienen un comportamiento distinto al resto de dosados analizados y puede ser debido a que las reacciones y velocidades de reacción que se producen dentro de la cámara de combustión son distintas y por tanto los inquemados reaccionan de manera diferente.

Es importante saber que al comienzo de cualquiera de las combustiones planteadas en los *Figura 5.26*, *Figura 5.27* y *Figura 5.28*, aparecen unos picos de sobredimensionamiento de la densidad celular, esto se debe a que, al comienzo del análisis del video, cuando todavía no se ha producido la ignición, el programa detecta contornos pertenecientes a la cámara de combustión. Al desarrollarse después el frente de llama, el programa tarda en converger hasta que encuentra los contornos más grandes que son los correspondientes al frente de llama.

El último de los análisis que se realiza consiste en comparar los resultados obtenidos para la densidad celular frente al radio del frente de llama, buscando así observar para qué radio aparece la celularidad y cómo es el crecimiento de esta para instantes posteriores. Al igual que en los análisis anteriores, se busca estudiar cómo son los resultados para distintos dosados, mantenido constante la presión inicial del experimento y el mismo valor de la variable "modificación_area" asignada en el programa desarrollado.

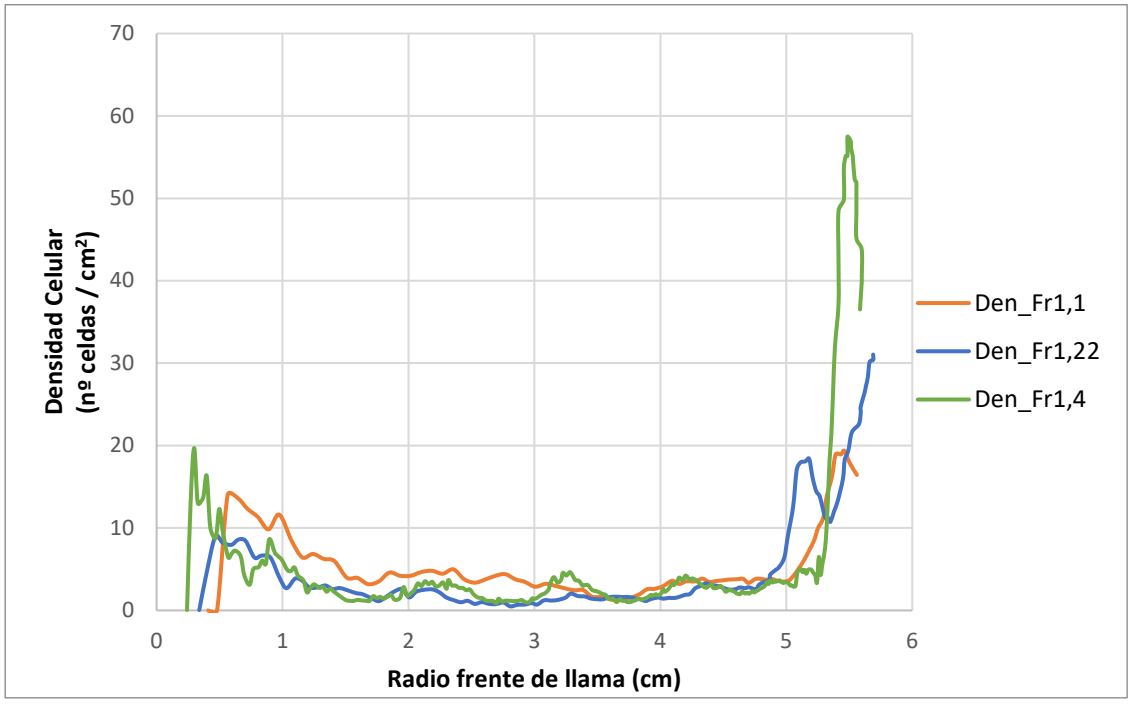


Figura 5.29: Gráfico comparativo de la densidad celular frente al radio del frente de llama para distintos valores del dosado estableciendo como parámetros fijos una presión inicial de 1,5 bar y variable "modificacion_area" = 2.

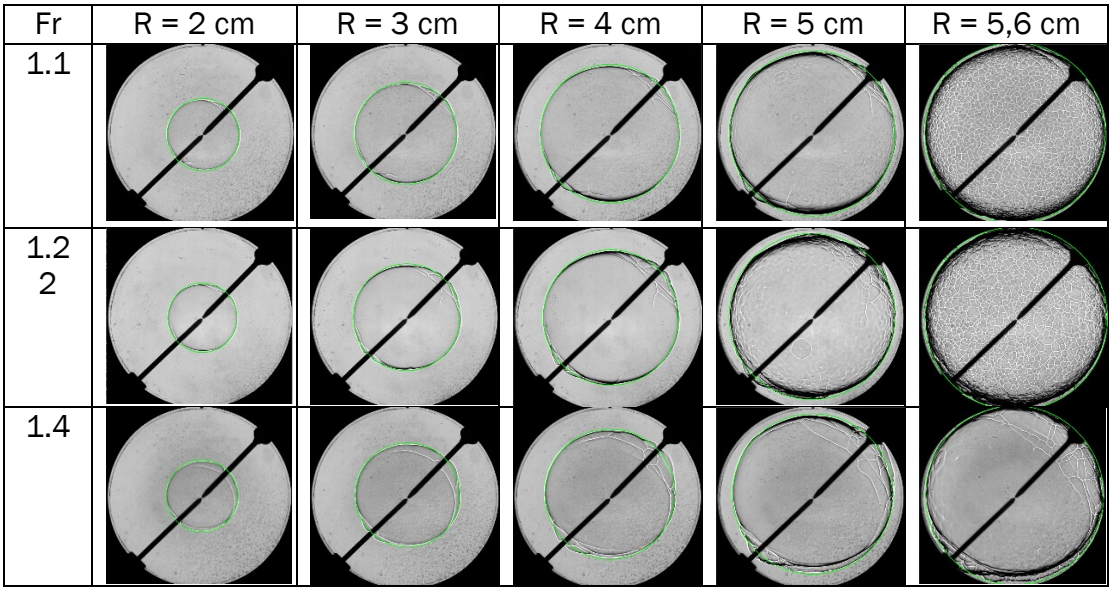


Figura 5.30: Imágenes obtenidas por el programa de Python para distintos radios bajo para distintos dosados estableciendo como parámetros fijos una presión inicial de 1,5 bar y variable "modificacion_area" = 2.

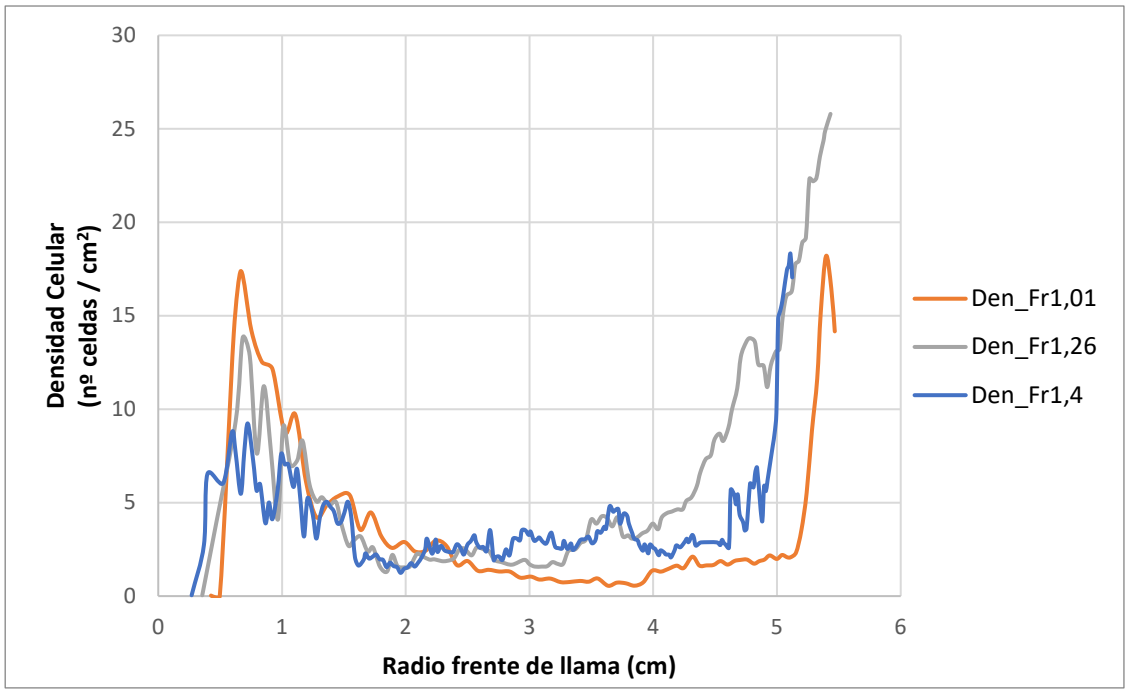


Figura 5.31: Gráfico comparativo de la densidad celular frente al radio del frente de llama para distintos valores del dosado estableciendo como parámetros fijos una presión inicial de 1,95 bar y variable "modificacion_area" = 3.

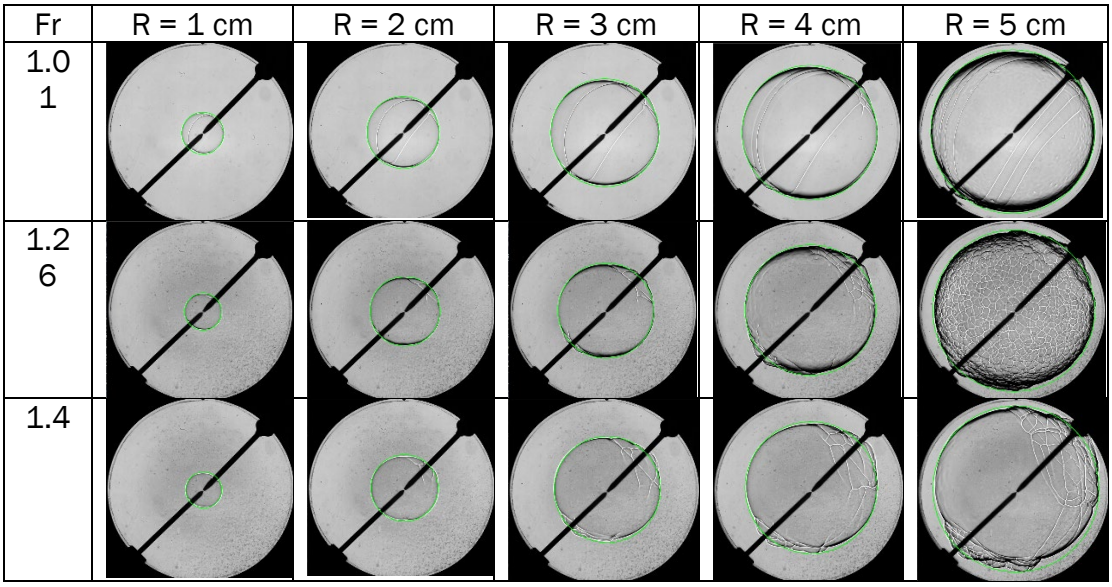


Figura 5.32: Imágenes obtenidas por el programa de Python para distintos radios bajo para distintos dosados estableciendo como parámetros fijos una presión inicial de 1,95 bar y variable "modificacion_area" = 3.

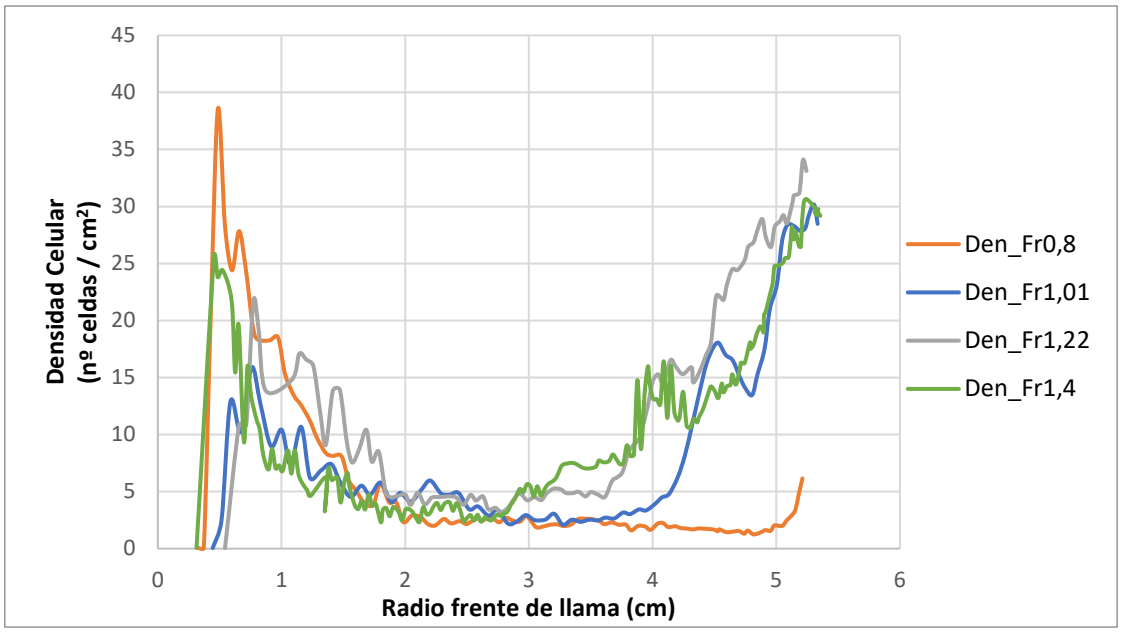


Figura 5.33: Gráfico comparativo de la densidad celular frente al radio del frente de llama para distintos valores del dosado estableciendo como parámetros fijos una presión inicial de 3 bar y variable "modificacion_area"= 4.

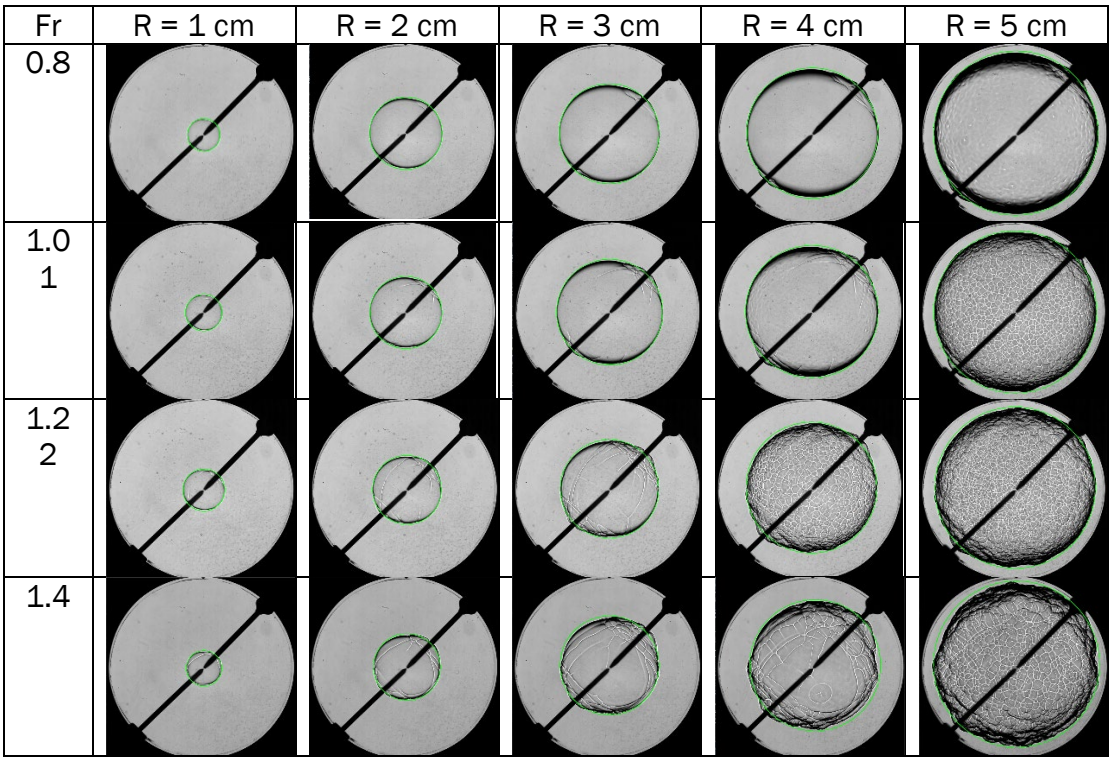


Figura 5.34: Imágenes obtenidas por el programa de Python para distintos radios bajo para distintos dosados estableciendo como parámetros fijos una presión inicial de 1,95 bar y variable "modificacion_area" = 4.

Con las gráficas representadas, se puede observar el comportamiento del parámetro de densidad celular frente al crecimiento del radio del frente de llama para estudiar para qué valor del radio se desarrolla la celularidad para cada dosado.

Para presiones iniciales bajas, las llamas son muy laminares hasta radios próximos a 5 cm, cuando se desarrolla la celularidad lo hace de manera rápida ya que se obtienen valores de densidad celular altos. Para presiones bajas y altos dosados, 1,4, la celularidad se aparece prácticamente cuando se encuentra próximo a las paredes de la cámara de combustión.

A la que va aumentando la presión inicial, para dosados cercanos al estequiométrico, la celularidad tarda más en desarrollarse llegando aparecer hasta valores altos del radio del frente de llama creciendo la celularidad de manera espontánea. Para valores con dosados ricos o pobres pero que no son próximos al estequiométrico, la celularidad se adelanta y aparece para radios a partir de los 3 cm. Los dosados muy ricos (1,4) no favorecen la aparición de la celularidad ya que se desarrolla prácticamente al final de la combustión, si se llega a desarrollar.

Para bajas presiones iniciales, las llamas son prácticamente laminares por lo que la densidad celular no adquiere valores muy elevados, independientemente del dosado, hasta la aparición de la celularidad. Al aumentar la presión inicial, al principio de las combustiones comienzan a desarrollarse grietas que afectan a los valores de densidad celular generando que la celularidad se desarrolle de una forma más progresiva. Para valores altos de presión inicial, las grietas que se forman desde un comienzo son grandes, lo que afecta al frente de llama y distorsiona el comienzo exacto de donde se desarrolla toda la celularidad.

Al igual que ocurre con las gráficas donde se compara la densidad celular frente al tiempo, al comienzo del análisis realizado por el programa desarrollado con Python, el programa detecta contornos pertenecientes a la cámara de combustión, tras la ignición el programa comienza a detectar otros contornos correspondientes al frente de llama y empieza a converger obteniendo los resultados que se muestran en las *Figuras 5.29*, *Figura 5.31* y *Figura 5.33*.

6. CONCLUSIONES

En el presente Trabajo de Fin de Grado se ha desarrollado un programa utilizando el lenguaje de programación de Python para la detección de contornos en vídeos deprocesos de combustión obtenidos con la técnica Schlieren. Dicho programa se ha desarrollado sobre la base de un programa similar desarrollado en Matlab y ha corregido errores importantes del mismo, además, el cambio de lenguaje de programación a Python permite que el programa sea más universal y versátil que su predecesor. La base de ambos programas es la misma: ambos buscan obtener información del fotograma con la que poder calcular el radio del frente de llama y la celularidad que hay en el interior de dicho radio.

Para poder comenzar a desarrollar adecuadamente este proyecto, ha sido necesario comprender y estudiar los fenómenos físicos que generan que el radio evoluciones de una manera o de otra y de que aparezca la celularidad en un instante de la evolución o en otro, además de identificar el tipo de celularidad que se está generando. Una vez definidos todos los conceptos, se analizó a fondo cómo capta y obtiene el programa de Matlab los parámetros partiendo del fotograma. Por último, se han estudiado las funciones de la librería de Python denominada OpenCV ya que esta ha permitido implementar los algoritmos y funciones necesarias para la detección de los contornos que se analizan en el video de cada combustión.

Posteriormente, se procedió a realizar un esquema de la estructura que debe de tener el programa. Para este programa han sido necesarias utilizar librerías para utilizar funciones que permitan calcular áreas máximas o representar los datos en gráficas, pero todo el proceso de análisis del fotograma ha sido posible gracias a los algoritmos que dispone la librería OpenCV. Para calcular las variables que almacenan los datos que se obtienen del análisis de las imágenes, se han utilizado las mismas fórmulas que en el programa de Matlab en cuanto a comparación de los datos con los parámetros de calibración del radio de la cámara de combustión y ocurre de una manera muy similar con los datos para obtener la variable de la densidad celular que hay en el frente de llama.

Sin embargo, la forma que tienen ambos programas de detectar y obtener esos parámetros es diferente. El programa de Matlab utiliza el método Ransac donde selecciona 3 puntos aleatorios y de ellos genera una circunferencia. Una vez definida la circunferencia hace una resta de fotogramas y la diferencia entre ambos es la región donde el programa debe seleccionar los siguientes tres puntos aleatorios, repitiendo este proceso hasta el final de la combustión. Los problemas aparecen cuando la primera circunferencia generada a partir de los 3 puntos aleatorios no se genera con puntos cercanos al frente de llama, por lo que los datos que se generan se sobredimensionan o se subestiman y seguidamente se realiza la resta de fotogramas, por lo que generalmente en todos los videos analizados usando este programa el radio calculado no coincide con el frente de llama. Ocurre de una manera

muy similar con la detección que emplea para calcular la celularidad en el interior del radio calculado.

El programa desarrollado en Python, resuelve estos problemas analizando individualmente cada fotograma del cual detecta los contornos para cada uno de ellos y realiza el cálculo del radio del frente de llama en base al número de contornos seleccionados y detecta la celularidad empleando una función encargada de adelgazar los pixeles detectados sin modificar la información de contienen. Con esto la detección en cada fotograma es independiente de la información obtenida en el fotograma anterior y no influye en el cálculo de los parámetros del fotograma siguiente.

Para poder ejecutar todo el programa de la manera más adecuada, ha sido necesario estudiar a fondo todas funciones de interés que componen la librería OpenCV para que calcule de manera oportuna los parámetros buscados para analizar los datos posteriormente. Estudiando cómo funcionan las funciones encargadas de detectar los contornos, el tratamiento previo que necesita el fotograma para detectar adecuadamente los contornos y el método de transformada de adelgazamiento que se emplea para simplificar los contornos del frente de llama sin reducir ni perder la información necesaria para el estudio posterior.

Otro de los problemas que surgía al emplear el programa de Matlab, es que el formato de video que se utilizaba, ".avi", no era el formato de video que se guardaba cuando en la parte experimental por la cámara de alta velocidad, que es un formato ".cine". El formato ".avi" se obtiene al comprimir el ".cine", por lo que se pierde información. El programa desarrollado en Python permite analizar cualquier tipo de extensión lo que favorece que se use la extensión con la que se captan los videos en la fase experimental que es ".cine", es decir, se analiza toda la información de capta la cámara de alta velocidad.

El programa de Matlab una vez capturados todos los datos, generaba un archivo en el formato ".txt" que necesitaba ser tratado para poder traducir los datos al un formato de Excel con el que poder analizar y comparar resultados. El programa de Python directamente genera el formato Excel distinguiendo en su interior que datos corresponden al "TIEMPO", los datos que corresponden al "RADIO DEL FRENTE DE LLAMA" y la columna que almacena los datos de "DENSIDAD CELULAR" referentes a la densidad celular que detecta en el frente de llama.

Una conclusión referente a los estudios realizados son las siguientes:

- Respecto a la comparativa a la forma que tiene Matlab de obtener sus radios y densidad celular, el radio puede estar sobredimensionado o subestimado respecto al calculado por Python, mientras que la densidad celular que captura Matlab siempre está sobredimensionada respecto de la captada por Python. En Python también aparecen una serie de consideraciones con la que

se debe de contar para representar posteriormente los datos, y sucede que en ciertos videos si no se selecciona el mejor valor para la "modificacion_area", aparecen picos donde se sobredimensionan los datos. Además, el programa para el comienzo de la combustión tarda unos milisegundos hasta converger en los datos que hagan referencia al frente de llama.

- Para los estudios realizados asignando valores diferentes a la variable "modificacion_area" es importante saber que para llamas laminares es recomendable tomar valores entre los dos y tres contornos para que el programa realice los cálculos mientras que para llamas muy celulares o se sabe que la celularidad aparece sobre la mitad de la combustión, es recomendable tomar más contornos entre los cuatro y seis contornos. Cabe matizar que es importante que el programa siempre tenga un valor de mínimo dos contornos asignados en la variable ya que sino aparece un mensaje de error en la consola indicando que no puede realizar los cálculos debido a que no tiene un número suficiente de contornos. Además, tampoco es recomendable tomar más de seis contornos para realizar los estudios porque esos contornos pueden pertenecer a elementos demasiado pequeños que no tengan relevancia en el proceso de combustión y alterar el cálculo de los parámetros.
- Por último, respecto a la influencia del dosado Se concluye que los dosados próximos al estequiométrico tienen una mayor velocidad de avance del frente de llama, mientras que para dosados ricos o pobres, la evolución del frente de llama es más lenta.

7. REFERENCIAS

[1] M. González Rodríguez, «Visualización de procesos de combustión con cámara de alta velocidad», Universidad de Valladolid, Valladolid, 2018.

[2] J. M. Desantes Fernández y M. Lapuerta Amigó, *Fundamentos de combustión*. Valencia: Universidad de Valencia, 1991.

[3] M. Reyes Serrano, *Apuntes de Máquinas Térmicas*. Valladolid: Universidad de Valladolid, 2022.

[4] M. Reyes Serrano, «Apuntes de Motores de Combustión Interna Alternativos», Universidad de Valladolid, Valladolid, 2023.

[5] R. Sastre Zamora, «Estudio y caracterización del origen y la naturaleza de las inestabilidades generadas en procesos de combustión de mezclas hidrógeno/metano», Universidad de Valladolid, Valladolid, 2021.

[6] A. L. Camaño Camaño, «Estudio experimental y modelado CFD del proceso de combustión de mezclas combustibles de hidrógeno, metano y gas de síntesis en una bomba cilíndrica con acceso óptico», Universidad de Valladolid, Valladolid, 2021.

[7] M. Gómez Carnero, «Estudio y simulación de cámaras de combustión mediante CFD», Universidad de Valladolid, Valladolid, 2023.

[8] «La fotografía Schlieren: El arte de visualizar lo invisible». Accedido: 6 de mayo de 2024. [En línea]. Disponible en: https://noticiasdelaciencia.com/art/47286/la-fotografia-schlieren-el-arte-de-visualizar-lo-invisible

[9] C. Carvalho, «¿Qué es Python? Historia, sintaxis y una guía para iniciarse en el lenguaje». Accedido: 20 de enero de 2024. [En línea]. Disponible en: https://www.aluracursos.com/blog/que-es-python-historia-guia-para-iniciar

[10] «Python: qué es, para qué sirve y cómo se programa». Accedido: 19 de enero de 2024. [En línea]. Disponible en: https://www.cursosaula21.com/que-es-python/

[11] Hugo Rodríguez, «¿Qué es OpenCV y para qué sirve?» Accedido: 20 de enero de 2024. [En línea]. Disponible en: https://www.crehana.com/blog/transformacion-digital/que-es-opencv/

[12] «Procesamiento de imágenes con OpenCV en Python». Accedido: 20 de enero de 2024. [En línea]. Disponible en: https://imaginaformacion.com/tutoriales/opencv-en-python

[13] «OpenCV una biblioteca para el reconocimiento de objetos en imágenes y cámaras». Accedido: 15 de mayo de 2024. [En línea]. Disponible en: https://blog.desdelinux.net/opencv-una-biblioteca-para-el-reconocimiento-de-objetos-en-imagenes-y-camaras/

[14] «Comenzando con OpenCV». Accedido: 20 de abril de 2024. [En línea]. Disponible en: https://learnopencv.com/getting-started-with-opencv/

[15] «Automatiza las cosas aburridas con Python». Accedido: 15 de marzo de 2024. [En línea]. Disponible en: https://automatetheboringstuff.com/

[16] «NumPy: La biblioteca de Python más utilizada en Data Science». Accedido: 15 de abril de 2023. [En línea]. Disponible en: https://datascientest.com/es/numpy-la-biblioteca-python

[17] «Introducción a la librería Matplotlib de Python». Accedido: 10 de abril de 2024. [En línea]. Disponible en: https://programacion.net/articulo/introduccion_a_la_libreria_matplotlib_de_python_1599

[18] «La librería Pandas». Accedido: 25 de mayo de 2023. [En línea]. Disponible en: https://aprendeconalf.es/docencia/python/manual/pandas/

[19] «Interfaces misceláneas del sistema operativo». Accedido: 20 de mayo de 2023. [En línea]. Disponible en: https://docs.python.org/es/3.10/library/os.html

[20] «Procesamiento de imágenes extendido». Accedido: 9 de junio de 2023. [En línea]. Disponible en: https://docs.opencv.org/4.x/df/d2d/group__ximgproc.html

[21] «Características del contorno». Accedido: 1 de mayo de 2023. [En línea]. Disponible en: https://docs.opencv.org/4.x/dd/d49/tutorial_py_contour_features.html