Trabajo Fin de Máster
Máster en Ingeniería Industrial (MII)

Desarrollo de un código para el procesado de vídeos obtenidos en la producción de microfibras mediante un dispositivo de electro-flow focusing

Autor: Fernando Carrión Ronda
Tutor: Elena de Castro Hernández

Dpto. Ingeniería Aeroespacial y Mecánica de Fluidos
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2019

Proyecto Fin de Máster
Máster en Ingeniería Industrial

Desarrollo de un código para el procesado de vídeos obtenidos en la producción de microfibras mediante un dispositivo de electro-flow focusing

Autor:
Fernando Carrión Ronda

Tutor:
Elena de Castro Hernández
Profesor ayudande Doctor

Dpto. de Ingeniería Aeroespacial y Mecánica de Fluidos
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2019

Proyecto Fin de Carrera: Escuela Técnica Superior de Ingeniería de Sevilla

Autor: Fernando Carrión Ronda

Tutor: Elena de Castro Hernández

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2019

El Secretario del Tribunal

Índice general

1. Agradecimientos **3**

2. Motivación **5**

3. Descripción del código para el procesado de vídeos generados en microfluídica **13**

3.1. Introducción 15
3.2. Descripción de las variables principales 17
3.3. Preprocesaso 19
3.3.1. Selección de nuevo vídeo o imagen 19
3.3.2. Ángulo de rotación 19
3.3.3. Región de ineterés 21
3.3.4. Resolución y velocidad de adquisición del vídeo . . . 22
3.3.5. Tamaño de la burbuja 22
3.4. Análisis de los fotogramas 24
3.4.1. Análisis realizado para las burbujas 24
3.4.2. Análisis realizado para el chorro 28
3.5. Postproceso 33
3.5.1. Tamaño de burbuja 33
3.5.2. Frecuencia de generación de burbujas 34
3.5.3. Velocidad de la burbuja, caudal interior y exterior . . 34
3.5.4. Longitud y diámetro del chorro 36
3.5.5. Visualización del vídeo 36
3.5.6. Elegir roturas 37
3.5.7. Elegir instantes de tiempo 37
3.5.8. Salvar resultados 38
3.6. Representación de las fronteras extraídas 40

4. Breve manual de uso **43**

5. Evolución de la longitud del chorro al variar las condiciones experimentales **59**
5.1. Introducción y descripción de las condiciones del análisis . . 61
5.2. Análisis de la influencia de la frecuencia 62
5.3. Análisis de la influencia del voltaje 64
5.4. Análisis de la influencia de la conductividad 66

6. Conclusiones **69**

A. Bibliografía **75**

B. Código empleado **79**

Capítulo 1

Agradecimientos

En primer lugar, agradecer a Elena de Castro Hernández por darme la oportunidad de realizar este proyecto. Agradecer también a Win Van Hoece por aportar fragmentos de código que han servido de apoyo para el cálculo del tamaño de las burbujas, y a Pilar Fernández Pisón por aportar fragmentos de código que han servido de apoyo para el cálculo de propiedades del chorro. Gracias también a Antonio Silas por la colaboración en la verificación del funcionamiento del código.

En último lugar pero no por ello menos importante a toda mi familia.

Gracias.

Capítulo 2

Motivación

Además de en nuestra vida cotidiana en forma de lluvia, evaporación etc., las burbujas aparecen de forma muy frecuente en el ámbito industrial y profesional en prácticamente la totalidad de las ramas del conocimiento. Los **meteorólogos** estudian la formación de lluvia y granizo; los **ingenieros químicos y de materiales** en la producción y procesado de aerosoles; los **ingenieros mecánicos** en la combustión, electromecanizado y evaporación. En la **industria alimentaria** para múltiples aplicaciones como enmascarar olores o mejorar textura entre otros. En la **industria farmaceútica** también tiene una gran importancia, tanto para la producción del fármaco como para su transporte. También en la rama de la **medicina** encontramos, entre otras aplicaciones, la mejora del suministro de fármacos vía pulmonar.

Debido a las múltiples aplicaciones de la generación de burbujas, las investigaciones que tratan de avanzar en el conocimiento de dichos procesos están en auge en los últimos años. Hay distintos procesos para la generación artificial de burbujas, generalmente divididos en dos tipos: generación masiva y generación controlada.

Figura 2.1: Sistema de generación masiva de burbujas: Mezclador.

Los procesos de generación masiva son muy robustos, eficientes y son capaces de producir una gran cantidad de burbujas. El inconveniente es que

el control sobre el diámetro u otros parámetros de la burbuja es escaso, por lo que se complica la deducción o extracción de resultados concluyentes o el estudio de situaciones concretas. En las Figuras 2.1 y 2.2 se muestran ejemplos de sistemas para la generación masiva de burbujas.

Figura 2.2: Sistema de generación masiva de burbujas: Emulsificador de ultrasonido.

En el caso en el que se desee estudiar el comportamiento del fluido en unas condiciones concretas, hay que recurrir a métodos más precisos los cuales se engloban dentro del campo llamado **microfluídica**. Estos métodos se basan en el uso de tubos o canales con secciones pequeñas, consiguiendo de esa forma que el número de Reynolds sea bajo, el flujo laminar y por tanto controlable y reproducible. Se presentan a continuación los principales métodos actualmente más conocidos para la generación de chorros y burbujas:

1. **Flow-Focusing**: En este método un fluido es enfocado por otro fluido inmiscible a través de un orificio. En la Figura 2.3 se observa un ejemplo de sistema de generación de burbujas de agua utilizando como fluido de enfoque aceite.
   En la Figura 2.3 b) se observa el proceso de formación de las burbujas. El fluido exterior enfoca al interior a través del orificio. El fluido interior bloquea dicho orificio lo que conlleva un aumento de la presión aguas arriba. Este aumento de la presión acaba provocando la producción de una burbuja y el proceso continúa de nuevo.

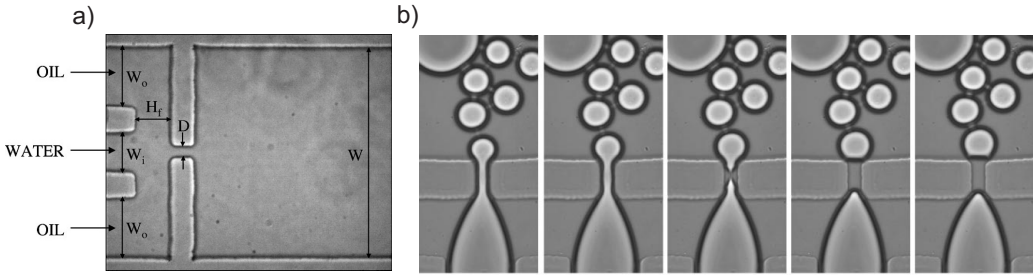

Figura 2.3: (a) Geometría Flow-Focusing implementada en un dispositivo generado por litografía plana. (b) Secuencia de imágenes mostrando la rotura de una burbuja en un dispositivo microfluídico Flow-Focusing.

2. **Coflujo**: En este método se hace circular dos fluidos inmiscibles paralelamente de forma que según la relacion entre las fuerzas de tensión superficial, inercia y de viscosidad, se producen distintos tipos de regímenes.

   En la Figura 2.4 podemos observar una forma de implementar esta configuración. Según los parámetros que se seleccionen seremos capaces de crear burbujas de menor tamaño que la sección del conducto de menor sección (régimen *narrowing*, Figura 2.5) o burbujas de mayor tamaño que el diámetro de menor sección (régimen *widening*, Figura 2.6). El primer caso es de gran interés ya que permite crear una burbuja de un diámetro más pequeño al del elemento que lo genera, pudiendo así generar burbujas pequeñas con elementos no necesariamente pequeños o poco robustos. Además de la diferencia ya comentada entre régimen *narrowing* y régimen *widening*, existe otra división adicional según si las burbujas se generan directamente en el final del conducto (*dripping*) o si se genera un chorro (*jetting*). En este proyecto no se entrará más en detalle sobre los distintos tipos existentes y en qué condiciones se dan, por estar fuera de los objetivos del mismo.

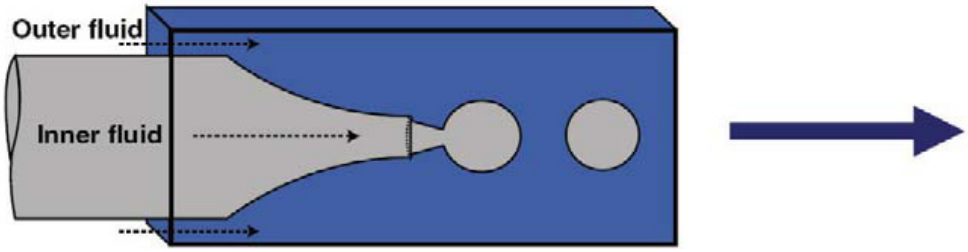

Figura 2.4: Esquema de la configuración de coflujo.

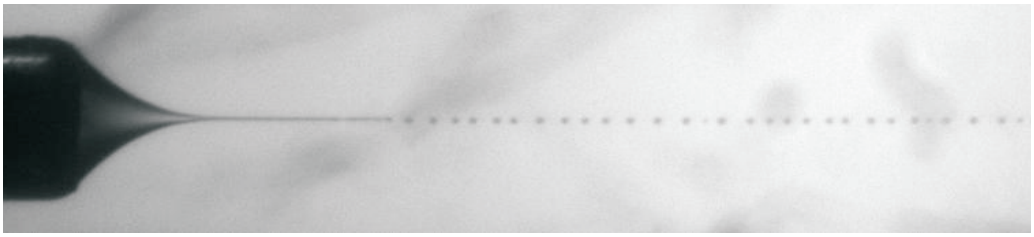

Figura 2.5: Configuración de coflujo operando en el régimen de narrowing.

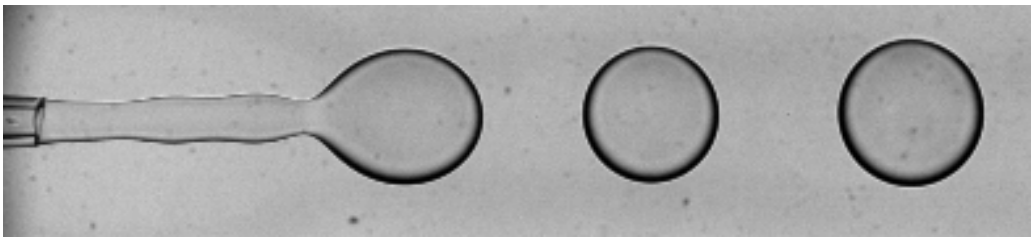

Figura 2.6: Configuración de coflujo operando en el régimen de widening.

Se han presentado algunos métodos de generación de burbujas, y las diferentes características que los definen y diferencian. No obstante, todos tienen un denominador común a raíz del cual surge el interés de este proyecto. Ese denominador común es la necesidad de un laborioso procesado del material resultante de los distintos vídeos o imágenes. Para obtener los resultados y deducciones necesarias de los distintos experimentos que se lleven a cabo es necesario realizar un control de los resultados obtenidos, como puede ser la distribución de tamaños de las burbujas, longitud y diámetro del chorro o frecuencia de generación de burbujas entre otros.

En este proyecto se continua el trabajo iniciado por un servidor en el Trabajo Fin de Grado "Desarrollo de un código para el procesado de vídeos en microfluídica", un programa que ofrece una alternativa al procesado manual de dicha información a través de una interfaz de usuario en Matlab, que permite al usuario procesar dicha información de una forma más rápida, precisa y exhaustiva. Respecto a la primera versión del programa, en esta se incluyen las siguientes mejoras o avances:

1. Se afina el cálculo de la frontera del chorro, incorporando algoritmos que mejoran el procesado de la imagen y tratan de solventar posibles problemas en la resolución y calidad de la imagen.

2. Incorpora un algoritmo para identificar y numerar los chorros entre una rotura y otra.

3. Permite visualizar el resultado del cálculo de la frontera del chorro y de la numeración de los mismos en la ya existente funcionalidad que permitía visualizar las fronteras de las gotas (botón "Watch Movie").

4. Se ha incorporado una funcionalidad para poder extraer en un bloc de notas las coordenadas de la frontera del chorro para cualquiera de las roturas y para cualquier instante comprendido entre una rotura y otra.

5. Se ha creado otro programa complementario para representar las fronteras utilizando el bloc de notas comentado en el punto anterior en gráficas diseñadas para facilitar la extracción de conclusiones.

En el Capítulo 3 se detalla la información que el programa ofrece como resultado así como el proceso seguido para obtenerla. En el Capítulo 4 se ofrece un breve manual del usuario con el objeto de permitir al lector

utilizar el programa de una forma rápida sin necesidad de entender el funcionamiento en su totalidad. En el Capítulo 5 se utiliza el programa en un caso práctico en el que se analiza la evolución de la longitud del chorro para distintas condiciones de frecuencia, voltaje y conductividad. En el Capítulo 6 se exponen las conclusiones obtenidas tras la realización de este proyecto así como las posibilidades de implementación de nuevos módulos que el programa ofrece dada su estructura. Por último, en los Apéndices se adjunta el código en su totalidad.

Capítulo 3

Descripción del código para el procesado de vídeos generados en microfluídica

3.1. Introducción

En las siguientes secciones del presente capítulo se van a desarrollar las distintas partes que componen el código del programa objeto de este proyecto. El programa ha sido implementado en una interfaz gráfica de usuario en Matlab para el procesado de vídeos de experimentos de microfluídica, para el análisis de burbujas y chorros en conductos o canales.

Los principales **resultados** que se pueden obtener a través del programa son los siguientes:

1. Índice de polidispersión y desviación estándar de tamaños.
2. Longitud y diámetro del chorro.
3. Frecuencia de generación de burbujas.
4. Velocidad de la burbuja.
5. Caudales interior y exterior.

Los **archivos** que se extraen del programa son:

1. Fichero con un resumen de todos los parámetros utilizados para el análisis y de los resultados extraídos.
2. Fichero con los diámetros medios de todas las burbujas analizadas en el vídeo.
3. Ficheros con las coordenadas de las fronteras de los chorros que se hayan seleccionado en las opciones "Choose breakupsz "Choose times".

Los **formatos** analizables por este programa son:

1. Imágenes en formato *.tif*.
2. Vídeos en formato *.tif*, tanto comprimidos como separados en fotogramas.
3. Vídeos en formato *.avi*.

Estos resultados se muestran por pantalla y además se almacenan en un archivo de texto en la carpeta donde se encuentra el vídeo que se está analizando. En la Figura 3.1 se muestra el menú interactivo con el que se van ejecutando las operaciones que el usuario decida. En el caso de una única

imagen el programa calcula longitud y diámetro del chorro, índice de polidispersión y desviación estándar, pero no calcula velocidad, frecuencia ni caudales ya que solo dispone de una sola imagen.

Figura 3.1: Menú interactivo del programa.

En la Sección 3.2 se describen las variables en las que se almacenan la información a lo largo del programa. A continuación, en la Sección 3.3 se describen las operaciones que se realizan a las imágenes antes de ser analizadas. En la Sección 3.4 se describe el proceso seguido para procesar el vídeo y almacenar la información necesaria para el cálculo de las distintos parámetros de interés anteriormente enumerados. Por útimo, en la Sección 3.5 se describen las operaciones realizadas para obtener dichos parámetros a partir de la información obtenida en el apartado de análisis.

3.2. Descripción de las variables principales

En el programa se utilizan una serie de variables como base para el cálculo de los resultados deseados. A lo largo de los bloques *Preprocessor* y *Analyze* se va recopilando información en dichas variables, de forma que en el apartado *Postprocessor* solo se maneja y opera con esa información obteniendo los resultados que se deseen, con la salvedad del apartado del cálculo de la longitud y diámetro del chorro que requiere un preprocesado concreto y particular.

A continuación, se describen las principales variables que utiliza el programa:

1. **Param**: En esta *estructura* se guardan los datos necesarios para realizar el análisis. Contiene la siguiente información:
   - Resolution: Resolución en μm por pixel.
   - Framerate: Fotogramas por segundo con el que fue grabado el vídeo.
   - Rotate: Contiene el ángulo que hay que girar los fotogramas para dejarlos en la posición adecuada para el análisis.
   - Object_area: Primera aproximación del área de la burbuja.
   - Frnum: Vector que contiene los números de los fotogramas que serán analizados.
   - Roi: Vector que contiene los cuatro vértices que determinan la region de interés para el análisis de las burbujas.
   - Roi_jet: Vector que contiene los cuatro vértices que determinan la región de interés para el análisis del chorro.
   - Ref_line: Vector que contiene los dos vértices de la línea a partir de la cual se comienza a medir la longitud del chorro.
2. **Results**: En esta *estructura* se guardan los resultados obtenidos tras realizar el análisis de todos los fotogramas. Contiene la siguiente información **para cada uno de los fotogramas analizados**:
   - Width, Height: Altura y anchura en píxeles del fotograma.
   - Framenumber: Número del fotograma.
   - nrBubblesCaptured: Número de burbujas detectadas en el fotograma.

- Bubbles: Estructura que contiene para cada una de las burbujas capturadas en el fotograma la siguiente información:

  Boundary: Conjunto de puntos que delimitan la frontera de la burbuja.

  Area: Área de la burbuja en pixeles cuadrados.

  Centroid: Punto con las coordenadas del centro de la burbuja.

  Eccentricity: Excentricidad de la burbuja.

  Orientation: Orientación de la burbuja.

  MajorAxisLength: Diámetro mayor en pixeles.

  MinorAxisLength: Diámetro menor en pixeles.

  Radius: Radio de la burbuja en pixeles.

  Identity: Identidad o número de la burbuja. La numeración de las burbujas se explica en el apartado 3.4.

- Idrange: Vector con los números que identifican a las burbujas capturadas en el fotograma correspondiente.
- TotalNrBubbles: Total de burbujas capturadas hasta el fotograma correspondiente incluyendo todos los anteriores analizados.
- Jet: Estructura que contiene, para cada fotograma, la siguiente información del chorro correspondiente:

  JetBoundary: Coordenadas de la frontera del chorro.

  JetId: Identificador del chorro. Los chorros se van numerando a medida que las roturas van ocurriendo. El funcionamiento de detalle se explica más adelante, en el apartado 3.4.

  JetLength: Longitud del chorro.

  JetDiam: Diámetro del chorro.

3. **Sdir, Filename**: Cadena de caracteres que contiene la dirección de la carpeta en la que se encuentra el vídeo a analizar y el nombre de dicho vídeo respectivamente.

4. **Format_index**: En esta variable se almacena un número entre uno y cuatro en función del formato del vídeo que se va a analizar.

Como se expone en el Capítulo 6 el hecho de tener la información almacenada de una forma tan ordenada proporciona la posibilidad de incorporar nuevos módulos al programa, si el usuario así lo desea, de una forma considerablemente sencilla.

3.3. Preprocesaso

En este apartado se encuentra el conjunto de operaciones que el usuario lleva a cabo para contextualizar el análisis que va a realizar y dar información relevante sobre el vídeo, necesaria para su posterior análisis.

3.3.1. Selección de nuevo vídeo o imagen

Esta operación se realiza siempre que abramos el programa antes de aparecer el menú y en caso de que seleccionemos la opción cambiar vídeo o imagen. Los objetivos y resultados de este apartado son los siguientes:

- **Obtener dirección y formato del vídeo**: Para ello se utiliza la función *uigetfile*, a través de la cual se almacena en dos variables separadas (*sdir* y *filename*) la dirección y el nombre del archivo a analizar, y en otra variable (*format_index*) un número que identifica el formato del vídeo o imagen a analizar. La numeración utilizada es la siguiente:

  Format_index vale uno en caso de que el formato sea *.avi*.

  Format_index vale dos en caso de que el formato sea *.tif* comprimido.

  Format_index vale tres en caso de que el formato sea *.tif* descomprimido.

  Format_index vale cuatro en caso de que el formato sea una imagen *.tif*.

  De esta forma, siempre que se seleccione una imagen o fotograma a lo largo del programa se utilizará la variable *format_index* para identificar la forma en la que la imagen debe ser leída.

- **Carga de parámetros**: En caso de que en la carpeta en la que se encuentra el vídeo haya guardada una variable de parámetros, esta se cargará automáticamente de forma que no será necesario empezar el preproceso desde el principio.

3.3.2. Ángulo de rotación

Para el correcto funcionamiento del programa es necesario que las burbujas o burbujas viajen desde el margen izquierdo de la pantalla hacia el

derecho, en sentido positivo según el criterio de signos convencional y en el eje horizontal (ver Figura 3.2) . Basta con girar la imagen en los casos en que sea necesario hasta colocarla como se muestra en la Figura 3.2. En el caso de la Figura 3.3 bastaría con girar la imagen 90°C en el sentido de las agujas del reloj. El motivo por el que deben disponerse las imágenes de esta forma se detalla en la sección 3.4. El ángulo a girar se guardará en *param.rotationangle* y cada vez que la imagen se muestre o se procese a partir de ahora se hará con dicho giro.

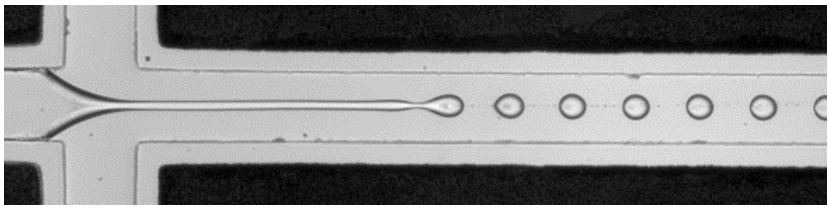

Figura 3.2: Imagen que muestra un fotograma de un vídeo colocado con la orientación adecuada.

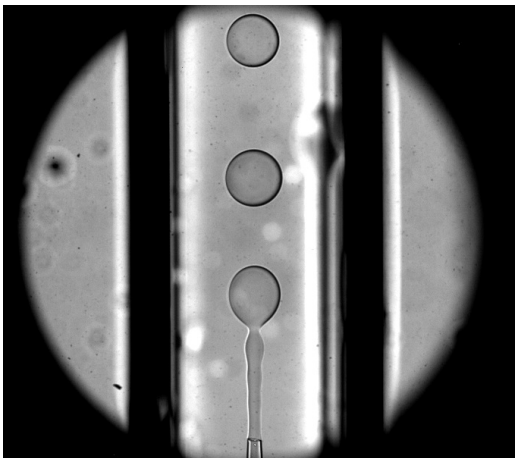

Figura 3.3: Imagen que muestra un fotograma de un vídeo con la orientación incorrecta.

3.3.3. Región de ineterés

En este apartado se selecciona la región para el análisis del chorro y para el análisis de las burbujas. También se selecciona la línea de referencia, línea a partir de la cual se comienza a medir la longitud del chorro. Para los dos primeros casos se muestra una imagen en el menú y se piden dos puntos correspondientes a dos vértices opuestos de un rectángulo, sin importar el orden en que se den. El objetivo es tener una región más reducida en la que encontrar las burbujas y el chorro respectivamente y quitar la parte de la imagen en la que se encuentren los márgenes de la imagen o la línea que delimita el conducto. De nuevo, los puntos que delimitan la región de interés serán almacenados en *param.roi* y *param.roi_jet* respectivamente. En la Figura 3.4 y 3.5 se observa un ejemplo de región de interés para el análisis de las burbujas y para el análisis del chorro.

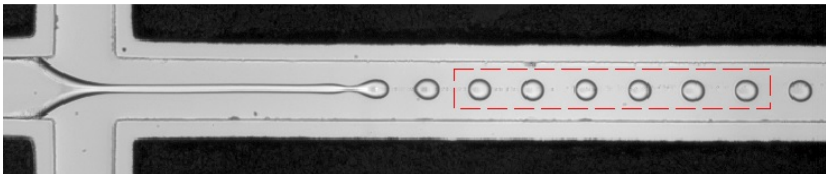

Figura 3.4: Imagen en la que se muestra un fotograma con la región de intereés para el análisis de las burbujas recuadrada en rojo.

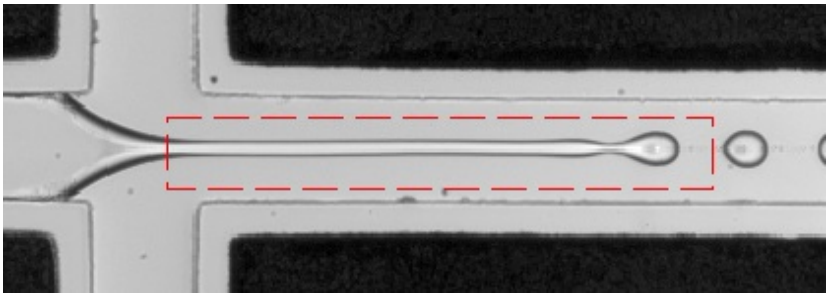

Figura 3.5: Imagen en la que se muestra un fotograma con la región de intereés para el análisis del chorro recuadrada en rojo.

Por último, se piden por pantalla dos puntos para definir la línea que se usará como referencia para medir la longitud del chorro. Vemos un ejemplo de esta línea en la Figura 3.6.

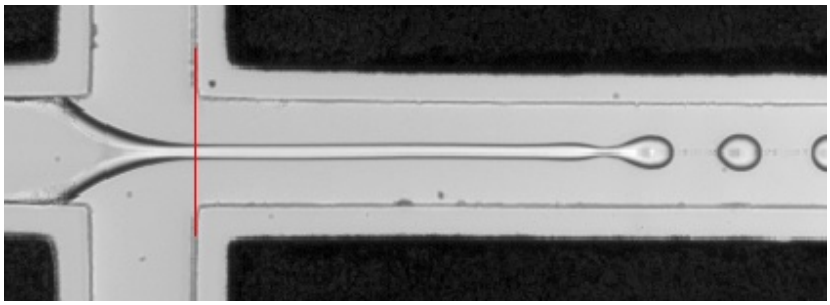

Figura 3.6: Imagen en la que se muestra un fotograma con la región de intereés para el análisis del chorro recuadrada en rojo.

3.3.4. Resolución y velocidad de adquisición del vídeo

Esta acción permite simplemente al usuario cambiar la resolución y los fotogramas por segundo(fps) del vídeo y almacenarlos en *param.resolution* y *param.framerate*. Estos dos datos son necesarios para posteriormente calcular todos los resultados como se verá más adelante.

3.3.5. Tamaño de la burbuja

En este último apartado del *preprocessor* se pretende reconocer una burbuja o burbuja para reducir el rango de búsqueda en el análisis. De esta forma, como veremos en la siguiente sección, para considerar un elemento como burbuja o burbuja se le exigirá que su área esté entre la mitad y el doble del área reconocida en este apartado.

Para ello, en primer lugar se muestra el primer fotograma y se piden por pantalla dos puntos. Estos dos puntos deben ser los vértices de un rectángulo en cuyo interior se encuentre una sola burbuja. De esta manera, tenemos un espacio muy reducido en el que buscar la burbuja. El siguiente paso se realiza a través de la función **get_objects**, cuyo código se puede encontrar en el Apéndice **??**. A continuación se explica la función:

1. **Sintaxis**: S = get_objects(Imagen, Áreas, Eccentricidad,Invertir)
2. **Descripción**:
    - Imagen: Imagen en la que se encuentra el objeto. En nuestro caso es la región delimitada por los puntos que se piden en el primer paso tras leer la imagen.

- Áreas: Vector formado por los límites entre los cuales tiene que estar el área de la burbuja para considerarse como tal. En nuestro caso, como aún no tenemos ningún dato al respecto, los límites serán que el área de la burbuja debe encontrarse entre el total del área de la región de la imagen que se ha seleccionado y la mitad de dicho área.
- Excentricidad: Límite de excentricidad de la burbuja.
- Invertir: En caso de que los píxeles que delimiten la frontera sean negros el valor de invertir debe ser uno. En caso contrario debe ser 0. Si no se introduce este argumento, por defecto se presupone que la frontera que delimita la burbuja es negra.
- S: Estructura que contiene información acerca de la burbuja que se ha encontrado. Contiene el área, centro, excentricidad,radio y frontera de la burbuja.

De esta forma, una vez se ha utilizado esta función ya se tiene el área de una burbuja, que se utiliza como criterio de aceptación de las demás en la etapa de análisis. Para ello almacenamos dicho valor en la estructura *param*, en el apartado *object_area*. El funcionamiento en detalle de esta subfunción es similar al funcionamiento de la subfunción utilizada en el apartado de análisis, por lo que será descrito en dicha Sección (3.4).

3.4. Análisis de los fotogramas

En este apartado se analizan todos los fotogramas almacenando información relevante acerca de las burbujas y del chorro que será procesada posteriormente. Debido a que el análisis y procesado de la imagen que se hace en el caso del chorro y en el caso de las burbujas es muy diferente, se va a a separar este apartado en dos subapartados, pese a que en el programa se realice todo desde un mismo módulo.

3.4.1. Análisis realizado para las burbujas

A continuación se explica el significado de dos variables que son claves para este subapartado:

- Maxxpos: Esta variable guarda la posición del centro de la burbuja detectada más cercana al borde derecho de la imagen, es decir, la que desaparecerá de la reginón de interés en los siguientes fotogramas.
- Idstart: Esta variable será la que sostenga la numeración de las burbujas. Su valor inicial es uno, y se irá actualizando cuando vayan saliendo burbujas de la región de interés para que la numeración sea la correcta. Es el id(Se entiende a partir de ahora por id el número que identifica a cada burbuja) a partir del cual hay que empezar a contar las burbujas.

Una vez inicializadas y declaradas globales dichas variables y la estructura *results* en la que se almacenan los resultados, comienza el bucle que recorre todos los fotogramas. En primer lugar se lee el fotograma , se le aplica la rotación que el usuario haya seleccionado en el *preprocessor* y se reduce la imagen a la región de interés. A continuación, se procesa el fotograma con la función ***Capture Bubbbles*** cuyo código podemos encontrar en el Apéndice 2. A continuación se explica el funcionamiento de dicha función:

1. **Sintaxis**: [width height nrBubblesCaptured idrange data] = captureBubbles(im, area, ecc, imbg)
2. **Descripción**:
    - Widht, height: Dimensiones de la imagen en píxeles.
    - NrBubblesCaptured: Número de burbujas encontradas en el fotograma.

- Idrange: Vector que contiene los id de las burbujas capturadas.
- Data: Estructura con información de cada burbuja encontrada en el fotograma correspondiente. Para cada burbuja incluye su área, puntos que delimitan la frontera de la burbuja correspondiente en el fotograma actual, centro, excentricidad, orientación, medida del eje más largo y del eje más corto, radio e id que la identifica.

- Im: Imagen una vez rotada y reducida a su región de interés.
- Area: Vector que contiene los límites entre los que se tiene que encontrar el área de una burbuja para considerarla como tal. Para ello se utiliza la primera aproximación obtenida en el apartado 7.
- Ecc: Excentricidad máxima para una burbuja, tomada por defecto como 0,6.
- Imbg: Imagen de fondo.

3. **Funcionamiento**: Su objetivo, además de obtener como parámetros de salida la información descrita, es numerar las burbujas. Se comienza **procesando la imagen**. Primero la convierte a binario, de forma que los píxeles con menos intensidad (negros), que son los que delimitan la frontera, quedarán como píxeles blancos y el resto negros. A continuación se borran los elementos conectados al borde de la imagen, de forma que si alguna burbuja no entra por completo en la región de interés no será tenida en cuenta, para evitar falsear el valor de los radios. A continuación borra pequeños elementos que hayan sido interpretados como píxeles negros que no estén conectados a nada, que podrán ser pequeñas manchas de la imagen, y rellena los agujeros, de forma que se rellena la frontera de las burbujas de blanco por completo. El proceso completo se ilustra en la Figura 3.7.

   Una vez procesada la imagen, se identifican los elementos que se encuentran en la imagen y se almacena su frontera en dos variables a partir de la función *bwboundaries* de Matlab. A continuación, se obtienen las propiedades de los distintos elementos detectados a través de la función *regionprops* y se cuentan los elementos que se han detectado que cumplen los requisitos comentados que tienen que cumplir para ser burbujas. De esta forma ya tenemos detectadas cuántas burbujas hay en el fotograma y sus propiedades, lo único que queda es **numerarlas** de forma que cuando pasen los fotogramas seamos capaces de

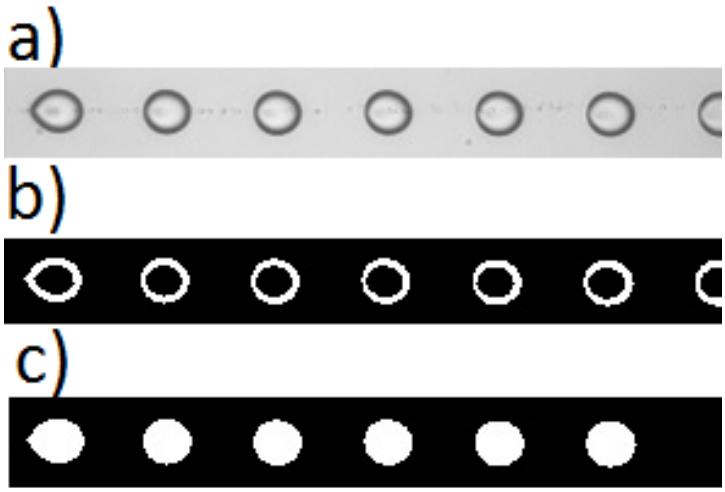

Figura 3.7: Imagen que muestra los pasos llevados a cabo por la función Capture Bubbles al procesar los fotogramas.(a) Imagen de la región de interés tal y como la recibe la función; (b) Imagen tras pasar a binario; (c) Imagen tras borrar elementos conectados al borde, rellenar agujeros o huecos y borrar pequeños elementos.

identificar las distintas burbujas de nuevo sin confundirlas con otras que aparezcan o desaparezcan.

Para ello se utilizan las variables *idstart* y *maxxpos*. Cuando se analice el fotograma número n, se tienen almacenadas las variables *idstart* y *maxxpos* del fotograma anterior. Al comienzo del bucle, se compara la posición de la burbuja más próxima a salir de la región de interés con la posición de la burbuja más próxima a salir de la región de interés en el fotograma $n-1$(almacenado en *maxxpos*). En el caso de que la burbuja más próxima a salir en el fotograma anterior al actual $(n-1)$ estuviera más cercana a salir que la burbuja más proxima a salir en el fotograma actual (n) significa que hay una burbuja que ha salido de la región de interés ya que las burbujas nunca viajan en sentido contrario al flujo, y entonces se actualiza la variable *idstart* y se empieza a sumar desde un número de id más($idstart = idstart+1$). Resumiendo, si en el fotograma actual la burbuja más cercana a salir está menos próxima a salir que en el fotograma anterior, empezamos a sumar desde un número de id más, ya que ese caso solo es posible cuando una burbuja ha salido de la región de interés (porque las burbujas no

viajan contrarias al flujo). En la Figura 3.8 se muestra el instante en que una burbuja desaparece y cómo evoluciona *maxxpos* en este caso.

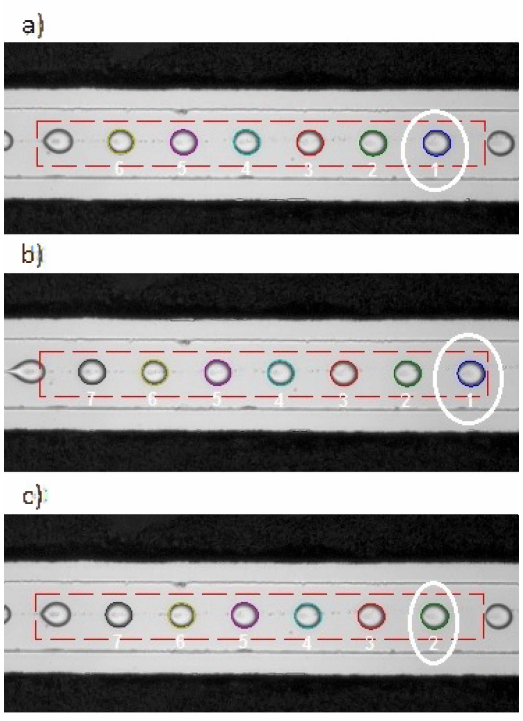

Figura 3.8: Ilustración que muestra el centro de la burbuja más cercana a salir de la región de interés(en blanco) en el momento en que va a desaparecer una burbuja. Nótese cómo en la transición de (b) a (c), momento en que desaparece una burbuja, maxxpos se retrasa.

El programa numerará de forma incorrecta en el caso de que la burbuja viaje entre dos fotogramas analizados una distancia igual o mayor a la distancia que separa dos burbujas. Este es un problema clásico de digitalización conocido como alyasing, y la solución sería bien bajar el incremento que se ha cogido entre fotogramas que se analizan o bien, en caso de que se hayan analizado todos los fotogramas, habría que repetir el experimento con más fotogramas por segundo para evitar este problema ya que incluso a mano sería complicado analizar estos experimentos.

Una vez está definido desde qué número de id hay que empezar a contar,

ya se pueden numerar el resto de burbujas y asignar a cada burbuja todas la propiedades.

3.4.2. Análisis realizado para el chorro

Para el análisis del chorro, se ha partido del mismo concepto de funcionamiento del código de las burbujas y se ha adaptado a las diferencias existentes:

1. En primer lugar, la identificación del chorro no obedece a la misma regla que en las burbujas. Para el chorro, el criterio para cambiar de número de identificación es lo que se ha denominado en el código criterio de rotura. En el momento en el que entre un fotograma y el siguiente la longitud del chorro disminuya en una cantidad igual o superior al diámetro de la burbuja, se considera que la rotura ha tenido lugar y se identifica el chorro como uno nuevo.

2. En segundo lugar, el procesado de la imagen requiere de dos pasos adicionales para adecuar la imagen para el análisis. Por una parte, cuando el chorro se va menguando antes de soltar la burbuja, la imagen se difumina y es posible que al pasar a binario se omitan algunos de los píxeles de la frontera. Por otra parte, en algunos chorros aparece un hilo remanente que une el chorro y la burbuja incluso una vez después de que se produzca la rotura, y este debe ser eliminado para no falsear la longitud.

Una vez enumeradas y comprendidas las diferencias existentes entre el procesado del vídeo para el caso de las burbujas y para el caso del chorro, pasamos a explicar con detalle los pasos del análisis de los fotogramas para extraer la información necesaria del chorro.

1. El primer paso del análisis es el **procesado de la imagen** contenida en la región de interés definida para el chorro durante el pre-proceso. Se comienza pasando a binario la imagen, luego se eliminan pequeños elementos que no son representativos respecto del resto de la imagen y se rellena el borde izquierdo de la imagen de píxeles blancos. Este último paso es necesario para el algoritmo que calcula la frontera del chorro, del cual se hablará más adelante.

   A continuación, se ejecuta un algoritmo que completa la frontera del chorro en caso de que haya píxeles que se hayan perdido al pasar a

binario. El funcionamiento, de forma resumida, es el siguiente. En primer lugar, calcula la línea media como la media de las coordenadas 'y' de los píxeles superiores e inferiores para cada coordenada 'x' a lo largo de toda la región de interés. Una vez calculada, se define como criterio de parada para el algoritmo que completa la frontera la coordenada 'x' del primer píxel blanco a lo largo de la línea media (que debe corresponder con el vértice del chorro). El objetivo de este criterio de parada es evitar completar como si fuera parte de la frontera del chorro el hilo remanente que se ha comentado en la introducción de este subapartado. Una vez definido el criterio de parada, el algoritmo recorre la imagen desde el límite izquierdo hasta la punta del chorro y, para cada columna de píxeles, asegura que hay dos líneas de frontera que definen el chorro. Si encuentra alguna columna en la que esto no se cumple, se coloca un píxel en esa columna a la misma altura que cualquier píxel de la frontera del chorro de la columna anterior. En la Figura 3.9 se ilustra el proceso seguido descrit para completar la frontera.

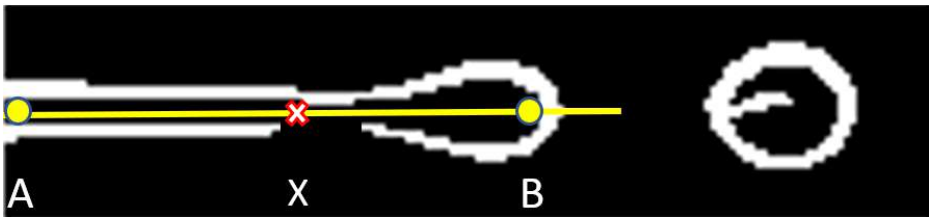

Figura 3.9: Imagen que ilustra el proceso seguido por el algoritmo que completa la frontera del chorro en caso de que haya píxeles que se pierdan al pasar a binario. El punto 'A' representa el inicio del algoritmo, el punto 'X' representa una coordenada horizontal de la frontera que será completado. El punto 'B' supone el final del algoritmo, punto en el que la línea media (representada en amarillo) corta la frontera del chorro.

Tras este paso, se rellena el chorro de píxeles blancos y se ejecuta el algoritmo que elimina el hilo remanente en caso de que exista. Esto se hace eliminando todo hilo con un diámetro menor de un 5 % del diámetro de la burbuja tomada como referencia en el pre-proceso. Con este último paso hemos completado el procesado de la imagen. En la Figura 3.10 se ilustran los pasos seguidos en el procesado de la imagen

para el análisis del chorro.

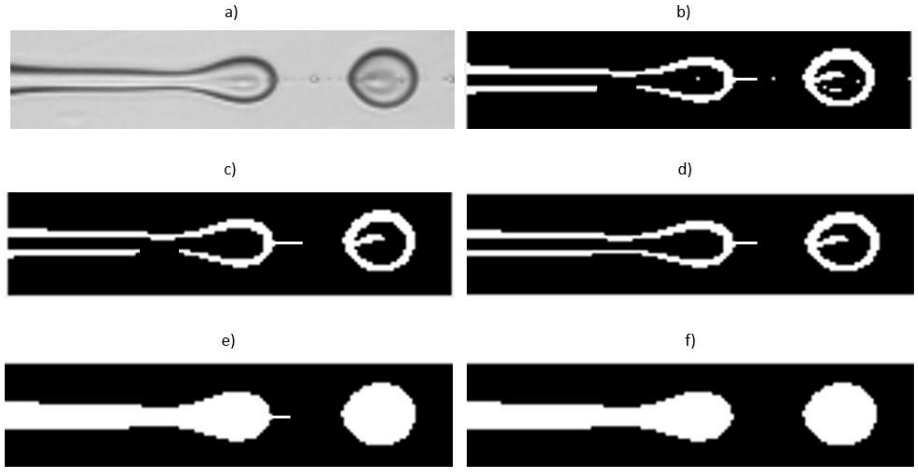

Figura 3.10: Imagen que muestra los pasos seguidos en el procesado de la imagen para el análisis del chorro. (a) Región de interés antes de ser procesada. (b) Imagen pasada a binario. (c) Imagen en binario tras eliminar los pequeños elementos no conectados con el chorro. (d) Imagen en binario con las partes del chorro incompletas ya completadas con píxeles. (e) Imagen en binario con el chorro rellenado de píxeles. (f) Imagen en binario con el hilo remanente eliminado.

2. El siguiente paso es el **trazado de la frontera**. Se hace con la función de Matlab *bwtraceboundary*. Esta función requiere empezar a buscar la frontera a partir de un punto que pertenezca a la misma. Es ese el motivo por el que se rellenó el margen izquierdo de blanco, para poder empezar desde la esquina superior izquierda de la imagen y que este punto pertenezca a la frontera. Además hay que indicar el sentido en que buscar la frontera(sentido de las agujas del reloj o el contrario a las agujas del reloj). Como hemos empezado por la parte de arriba el sentido que debe seguir es el de las agujas del reloj para ir recorriendo la frontera del chorro. En la Figura 3.11 vemos un ejemplo de frontera ya calculada superpuesta en azul a la imagen con la que se comienza el análisis del chorro.

3. A continuación, **calculamos la longitud del chorro**. Se calcula como la diferencia entre la coordenada abscisa del punto más a a la derecha

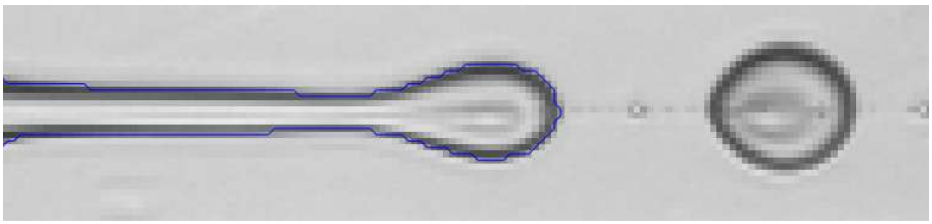

Figura 3.11: Imagen que muestra la frontera calculada en azul superpuesta a la imagen con la que se comienza el análisis del chorro.

de la frontera del chorro menos la coordenada abscisa de la línea de referencia multiplicado por la resolución.

4. Pasamos ahora al **cálculo del diámetro del chorro**: Para el cálculo del diámetro se busca en todas las verticales entre la línea de referencia y el final del chorro los píxeles más altos y más bajos con valor uno. Dicho así parece complicado, pero basta con buscar a lo largo de las columnas de la matriz de la imagen en binario los **unos de las filas más altas y mas bajas** y se almacena la diferencia entre sus posiciones en la variable *diam*. En la Figura 3.12 se ilustra un esquema del proceso seguido.

   Para cada columna se calcula el diámetro como la **diferencia entre las posiciones de los píxeles blancos de las filas superior e inferior** comentados anteriormente. De esta forma se tiene el diámetro para cada columna en píxeles. Para cada fotograma tendremos tantos valores de diámetros como columnas de píxeles haya entre la línea de referencia y el final del chorro. En un primer momento se pensó almacenar para cada fotograma la media del valor de todas las columnas, y luego calcular el diámetro medio del chorro como la media a lo largo de todos los fotogramas. Sin embargo se observó que de esta forma el valor del diámetro se sobredimensiona en muchos fotogramas debido a la burbuja que queda adherida al chorro antes de separarse del mismo. Por ello se decidió **para cada fotograma guardar la moda** de entre los valores de las columnas, y luego hacer la media de estos valores entre todos los fotogramas. Así se consigue un valor muy constante del chorro y sin sobredimensionar debido a la burbuja. En resumen:

Diam : Para cada fotograma, vector que contiene las distancias en píxeles entre los punto más altos y más bajos de la frontera en todas

las columnas. Para buscarlo se utiliza la función de Matlab *find*, que busca unos a lo largo de un vector y almacena el índice en el que se encuentran. Se guarda el máximo y mínimo y la diferencia entre ambos es la distancia almacenada en la variable *diam*. Tiene tantas componentes como columnas de píxeles haya entre la línea de referencia y el punto más alejado de la frontera.

Jet_diam : Vector que contiene la moda del vector *diam* para cada fotograma. Tiene tantas componentes como fotogramas se analicen.

Figura 3.12: Imagen que muestra el proceso seguido para el cálculo del diámetro del chorro. Se almacena la distancia entre los puntos de corte (cruces) de las líneas verticales (amarillas) con la frontera del chorro.

5. Por último, pasamos a **identificar el chorro**. Para hacerlo, comparamos la longitud del chorro actual con la del fotograma anterior, y si el actual es más corto que el anterior con una diferencia de al menos el diámetro de la burbuja, se suma en 1 el número de identificación del chorro. En caso contrario, se mantiene el mismo número de identificación.

Una vez realizadas estas operaciones para cada fotograma tendremos la variable *results* con toda la información que se introdujo en la Sección 3.2, y se está en disposición de hacer el post-procesado.

3.5. Postproceso

3.5.1. Tamaño de burbuja

En este apartado se muestra por pantalla el radio medio, el índice de polidispersión (PDI), la desviación estándar (SD), el número de burbujas capturadas. Además se muestran la distribución de áreas y radios en dos gráfica. Para ello, se cargan los radios y áreas a través de la función auxiliar *bubbleInfo*, cuyo código completo puede encontrarse en el Apéndice **??**, y cuyo funcionamiento es el siguiente:

1. **Sintaxis**: y = bubbleInfo(data, fieldname, id)
2. **Descripción**:
   - Data: Estructura a lo largo de la cual queremos buscar una propiedad determinada (Radio,Área,...)
   - Fieldname: Nombre de la propiedad que se desea buscar como cadena de caracteres.
   - Id: Vector con los id de las burbujas de las cuales se quiere conocer la información indicada.
   - Y: Matriz de celdas en la cual se almacena la información indicada de las burbuja con las id indicadas a lo largo de todos los fotogramas.

Si se utiliza la función de la forma

y = bubbleInfo(results,'Radius',1:totalNrBubbles)

obtendremos una matriz de celdas y que contiene todos los radios de todas las burbujas a lo largo de todos los fotogramas. Una vez obtenida y , solo hay que operar para obtener el resto de resultados y gráficas. Para obtener los datos de áreas se repite el mismo proceso intercambiando 'Radius' por 'Area'. El índice de polidispersión se calcula de la siguiente forma

$$PDI = 100 \cdot \frac{SD}{RadioMedio} \tag{3.1}$$

En la Figura 3.13 se muestra la distribución de radios y áreas de las burbujas.

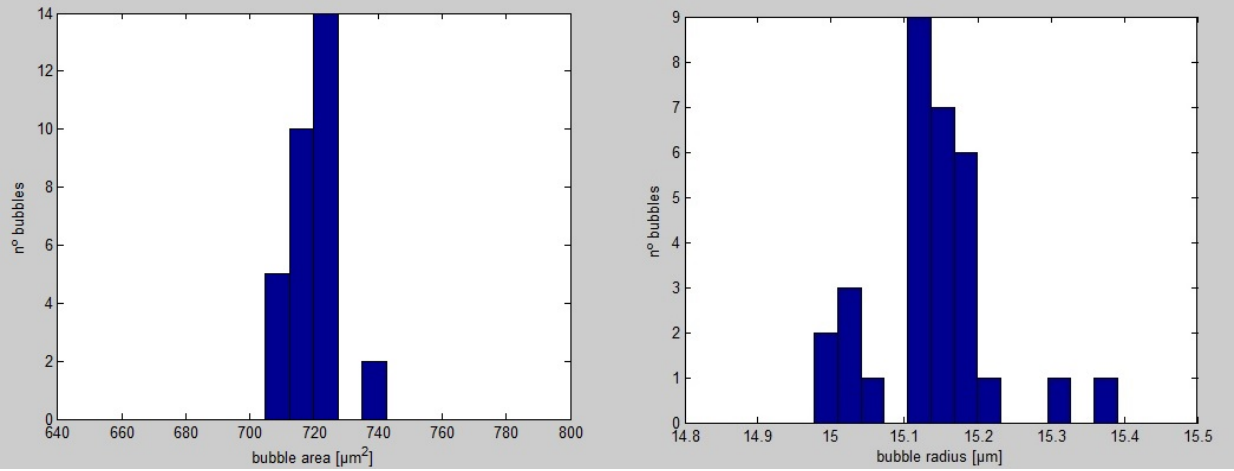

Figura 3.13: Distribución de áreas (1) y de radios (2).

3.5.2. Frecuencia de generación de burbujas

Para el cálculo de la frecuencia de generación de burbujas el primer objetivo es buscar el fotograma en el que aparecen por primera vez las burbujas. Para ello, para cada burbuja (salvo las que ya se encuentran en el primer fotograma, ya que estas ya han aparecido antes de comenzar el vídeo) se van recorriendo los fotogramas hasta que se encuentra por primera vez la id de cada burbuja en el apartado *idrange* de la variable *results*. Una vez obtenidos los fotogramas de creación de cada burbuja se calculan los tiempos en que se crean a través de los fotogramas por segundo que tenemos almacenados en *param.resolution*. Hecho esto, se calcula la recta de mejor ajuste de la recta [burbuja, tiempo de creación] cuya pendiente es la inversa de la frecuencia de generación de burbujas. La Figura 3.14 muestra la burbuja creada por cada instante y la recta de mejor ajuste en una gráfica que se muestra por pantalla.

3.5.3. Velocidad de la burbuja, caudal interior y exterior

Al igual que en el apartado anterior, el primer paso es calcular los tiempos en que aparecen las distintas burbujas. A continuación se busca el último fotograma en el que aparece cada burbuja de forma muy parecida y se calculan los tiempos de desaparición de igual forma que en el apartado de la

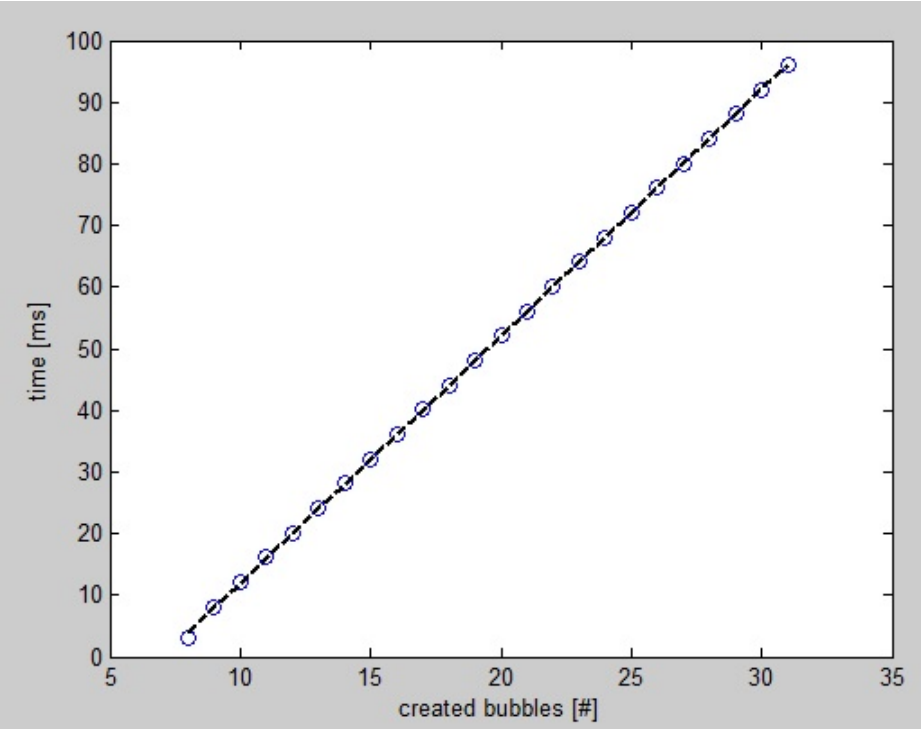

Figura 3.14: Burbujas generadas frente al tiempo de generación y determinación de la frecuencia de producción.

frecuencia. Con estos dos datos, ya tenemos el tiempo que tardan en recorrer las burbujas la región de interés, cuya longitud tenemos almacenada en píxeles en *param.roi* y podemos obtenerla en μm con la resolución. Se calcula la velocidad de cada burbuja como:

$$velocidad = \frac{longitud \quad roi}{t_{\text{desaparición}} - t_{\text{aparición}}} \tag{3.2}$$

Una vez obtenida la velocidad de todas las burbujas se muestra por pantalla la velocidad media de todas las burbujas. Cabe destacar que las burbujas que siguen apareciendo en el último fotograma analizado no se tienen en cuenta para el cálculo de la velocidad ya que estas no han recorrido por completo la región de interés. Además, en este apartado se muestran otras dos variables que suelen ser de interés en este tipo de experimentos, los caudales interior(Q_i) y exterior(Q_o). El **caudal interno** se calcula de la

siguiente forma:

$$Q_i = \frac{4}{3} \cdot \pi \cdot R_b^2 \cdot frecuencia, \tag{3.3}$$

donde la frecuencia es la de generación calculada en el apartado anterior y Rb es la media de todos los radios.

El **caudal externo** se determina como

$$Q_o = Area_{\text{sección}} \cdot velocidad_{burbuja} \tag{3.4}$$

Para el cálculo del área se piden tres datos por pantalla. W(ancho), H(alto) y R(radio). En caso de que la sección sea rectangular se introducen los dos primeros dejando cero el radio, y si es una sección circular al contrario. En caso de que no se deje ninguno como cero lo calculará como si fuera circular.

3.5.4. Longitud y diámetro del chorro

En este módulo se utiliza la información almacenada en *results* durante el módulo de análisis para mostrar la siguiente información:

1. Longitud media calculada como la media de las longitudes obtenidas para cada fotograma analizado.
2. Índice de polidispersión y desviación estándar de las longitudes.
3. Diámetro medio. Cabe destacar la forma en que se calcula el diámetro del chorro para que el usuario lo tenga en cuenta. En cada fotograma se almacena la moda(valor más repetido) del diámetro del chorro a lo largo de toda su longitud. Lo que se muestra por pantalla es la media de esos valores a lo largo de todos los fotogramas. Dado que en muchos casos hay una parte del chorro que permanece muy constante será algo frecuente que la polidispersidad y la desviación estándar sean cero.
4. Índice de polidispersión y desviación estándar de los diámetros.

3.5.5. Visualización del vídeo

Este módulo tiene como objetivo visualizar de una forma rápida lo que el programa ha identificado como chorro y como burbujas para comprobar si el procesado ha sido correcto. De esta forma, en caso de haber analizado las burbujas y de haber calculado longitud y diámetro del chorro mostrará por pantalla cada fotograma uno detrás de otro con la numeración y las

fronteras de las burbujas y del chorro. En caso de que todo salga según lo previsto en el vídeo tenemos una gran certeza de que los resultados que el programa ha calculado son correctos.

El código empleado consiste en mostrar los fotogramas uno detrás de otro con las fronteras que delimitan todos los elementos superpuestas. Ambas habían sido almacenadas en distintas variables. En cuanto a la frontera y numeración de las burbujas se guardaron ambas en *results.bubbles*, en el apartado 'boundary' e 'id'. La frontera y numeración del chorro se guardó en la variable *results.jet*.

3.5.6. Elegir roturas

En este módulo el usuario tiene la oportunidad de elegir las roturas de las burbujas que desee para que se guarden, al ejecutar la opción de salvar resultados, ficheros con las coordenadas de las fronteras de los chorros que seleccione.

El funcionamiento es sencillo. Como la información que relaciona el fotograma y la identificación del chorro que aparece cada uno se guardó en el módulo de análisis en la variable *results*, basta con guardar las roturas elegidas por el usuario en la variable global *Breakups*. Luego, al ejecutar el módulo de guardar resultados, se guardaran tantos ficheros como roturas se seleccionen con las fronteras de dichas roturas. En la Figura 3.15 se muestra el cuadro de diálogo mostrado para que el usuario aporte la información descrita.

3.5.7. Elegir instantes de tiempo

Además de poder seleccionar roturas para que se almacenen sus fronteras posteriormente, el usuario también tiene la posibilidad de guardar distintos instantes de tiempo de una determinada rotura. Para ello, se le pide al usuario por pantalla que indique la rotura de la cual quiere guardar distintos instantes de tiempo, y las divisiones de tiempo que desea.

Para las divisiones de tiempo, se ha decidido estructurar a través de la variable adimensional *tau*, que se calculará de la siguiente manera:

$$tau = \frac{i-1}{Divisiones\,de\,tiempo} \tag{3.5}$$

Donde i es el índice del componente del vector. De esta manera, los

fotogramas que se guardarán serán el total de fotogramas en los que aparece la rotura seleccionada multiplicado por el vector tau, o visto de una manera más sencilla:

$$Incremento = \frac{Fotograma\ final - Fotograma\ incial}{Divisiones\ de\ tiempo} \tag{3.6}$$

$$Fotogramas\ guardados = Fotograma\ incial + Incremento * (i - 1) \tag{3.7}$$

En la Figura 3.15 se muestran los cuadros de diálogo que se le muestran al usuario para que introduzca la información indicada en este sub-apartado.

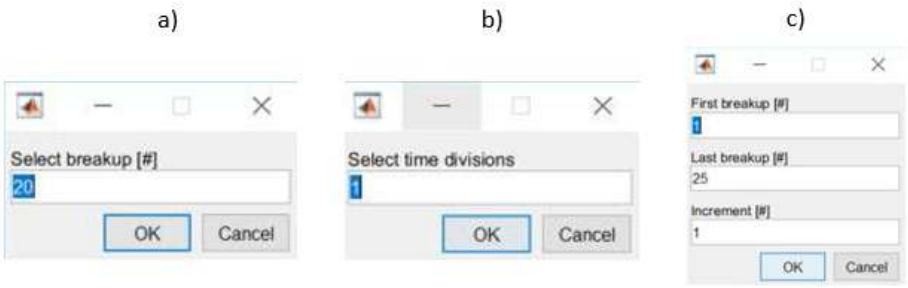

Figura 3.15: Cuadros de diálogo mostrados para que el usuario elija la rotura de la que quiere guardar los instantes de tiempo (a), el número de divisiones de tiempo (b) y las roturas que se quieran guardar (c).

3.5.8. Salvar resultados

El objetivo de este módulo es generar archivos de salida que el usuario pueda mantener y utilizar una vez se cierre el programa. Para ello se utilizan las funciones de Matlab *fopen, fprintf y fclose*. Los ficheros de texto se guardrán en la carpeta en la que se encuentra el vídeo con los nombres que se detallarán a continuación. Los ficheros que se generan son los siguientes:

1. Fichero con un resumen de todos los parámetros introducidos por el usuario durante el pre-proceso y con los resultados que el usuario haya pedido al programa calcular. El nombre de este fichero es '*NombreVídeo*Outputfile.txt'.

2. Fichero con los radios medios de todas las burbujas que aparecen en el vídeo. El nombre de este fichero es '*NombreVídeo*Radius.txt'.

3. Tantos ficheros como fronteras de chorros se haya pedido guardar. La regla que sigue el nombre es la siguiente:

   - En primer lugar coloca el nombre del vídeo analizado.
   - A continuación coloca la burbuja a la que corresponde la frontera.
   - Por último coloca el tau (fracción de tiempo entre roturas) correspondiente.

   Un ejemplo de nombre de fichero de frontera seía 'f5v500.avib06tau0.00', que corresponde a la sexta burbuja y tau 0 del vídeo 'f5v500.avi'.

```
--> INPUT PARAMETERS
Resolution: 1.560000e+00 µm/px
Framerate: 20000 fps
First frame:   1
Last frame:    1001
Increment:     10

--> RESULTS
Nº of bubbles captured: 32
Mean bubble radius: 9.712121e+00 µm
Bubble radius PDI: 0.6826 %
Bubble radius SD: 0.10342 µm
Mean generation frequency: 4.950495e+02 Hz
Mean jet length: 338.7053 µm
Jet length PDI: 4.2763 %
Jet length SD: 9.2847 µm
Mean jet diameter: 9.36 µm
Jet diameter PDI: 0 %
Jet diameter SD: 0 µm
Mean bubble velocity: 32767 µm/s
Qi: 25.9631 µl/h
Qo: 412.8642 µl/h
```

Figura 3.16: Archivo de texto que se genera con los parámetros de análisis y los resultados obtenidos al seleccionar la opción *save results*.

3.6. Representación de las fronteras extraídas

Por último, se ha desarrollado una segunda sencilla interfaz de usuario en Matlab que ofrece al usuario una forma rápida y adaptada al caso en estudio de representar los chorros utilizando el mismo formato extraído del programa principal.

Esta interfaz, representada en la Figura 3.17, dispone de tres gráficas en las que se pueden representar tantos chorros superpuestos como se deseen. La gráfica representa las fronteras en micrómetros, adapta los límites al chorro en cuestión, superpone tantos chorros como se seleccionen y anãde la leyenda con los nombres de los ficheros (que por la definición que se les ha dado contienen información sobre la frecuencia y el voltaje). Dispone de tres botones con las siguientes funcionalidades:

1. **Plot jet**: Con este botón seleccionamos los ficheros que contienen las fronteras de los chorros a representar. Podemos seleccionar tantos como deseemos. En primer lugar, el programa abre un cuadro de diálogo para seleccionar los archivos con contienen las fronteras. A cotinuación, una vez dispone de los ficheros, realiza las siguientes operaciones para representar los chorros:
   - Cambio de coordenadas para representar el chorro con la línea media en 'y=0'.
   - Cambio de coordenadas para representar los chorros solapados en el inicio o en el final del chorro en función de la opción que esté seleccionada (funcionalidad número cuatro).
   - Configura la leyenda con los nombres de los ficheros.
   - Configura los límites horizontales. Para ello, se establece el máximo como el número con el mismo número de ceros que el punto máximo de la frontera y con la primera cifra significativa la del máximo de la frontera más uno. Como ejemplo, si el punto máximo de la frontera fuese 573 micras, el máximo se fijaría en 600 micras. Para el paso entre unidades en el eje hotizontal, se divide en 20 unidades entre el cero y el máximo calculado.
   - Configura los límites del eje vertical de la misma manera que el eje horizontal, con la diferencia de que el chorro está centrado en cero, por lo que hay que fijar un límite superior y otro inferior.
2. **Save figure**: Guarda la representación actual en formato '*.fig*'.

3. **Clear figure**: Elimina la representación actual y resetea la configuración de ejes, leyenda, etc.

4. **Set reference**: Dos botones seleccionables que definen si los chorros deben representarse solapados al inicio del chorro o al final del chorro.

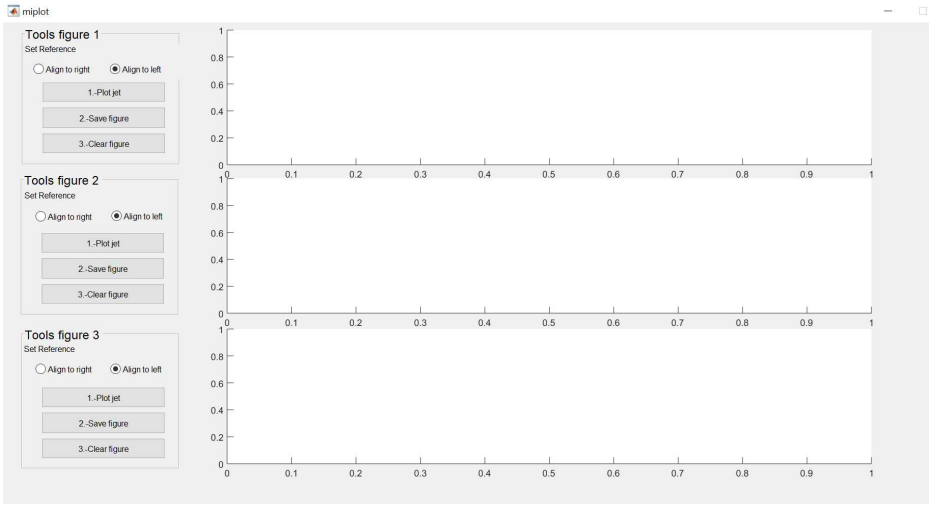

Figura 3.17: Interfaz de usuario que permite representar las fronteras de los chorros seleccionando los ficheros exxtraídos en el programa principal.

Esta interfaz ofrece al usuario la posibilidad de representar chorros en distintas condiciones de frencuencia, conductividad y voltaje en distintos instantes de tiempo entre roturas o en distintas roturas en tres gráficas en simultáneo. De esta manera, se cierra por completo el análisis de vídeos de microfluídica, quedando para el usuario la extracción de conclusiones en base a los resultados extraídos y representados.

Capítulo 4

Breve manual de uso

En este capítulo se exponen los pasos que el usuario debe seguir para realizar con éxito el procesado de vídeos, donde encontrar la información deseada y finalmente cómo comprobar de forma rápida que el procesado o análisis ha sido correcto.

Para iniciar el programa basta con escribir *analizar2* en el command window de Matlab. Antes de aparecer el menú se mostrará una ventana que nos pedirá que seleccionemos el vídeo correspondiente. En esta ventana podremos encontrar en la esquina inferior derecha un menú desplegable en el que deberemos indicar el formato del vídeo o imagen en caso de serlo del archivo que vayamos a analizar. En la Figura 4.1 se muestra dicha ventana. Una vez elegido el formato aparecerán los archivos con ese formato y podremos seleccionar el archivo que deseemos analizar.

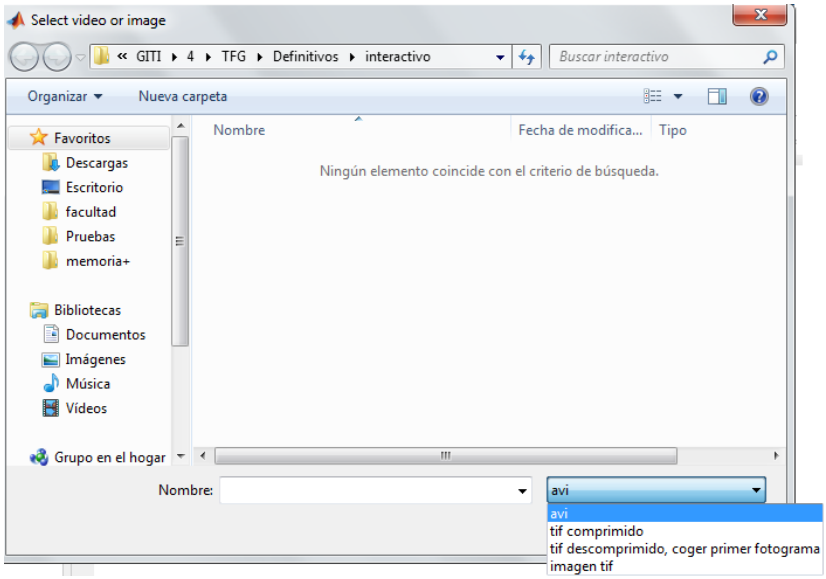

Figura 4.1: Ventana para seleccionar archivo. En la esquina inferior izquierda se muestra el menú desplegable para seleccionar el formato.

En la Figura 4.2 se muestra el menú interactivo tal y como aparece tras seleccionar un vídeo, con el primer fotograma mostrado en la gráfica superior.

A continuación se enumeran las acciones que se realizan al pulsar cada botón. La división en distintos módulos se debe a la diferencia de objetivos de cada grupo de botones. En primer lugar el módulo de *preprocessor* está

Figura 4.2: Menú interactivo.

compuesto por el conjunto de acciones que el usuario debe realizar antes del análisis. El módulo de *Analysis* se utiliza para, como su propio nombre indica, realizar el análisis. Por último, en el apartado *postprocessor* se muestran por pantalla los resultados correspondientes.

1. **Change video/image**: Puede utilizarse en caso de error al seleccionar el archivo cuando se abrió por primera vez el programa o en caso de querer comenzar un nuevo análisis sin cerrar el programa. En este segundo caso, antes de cambiar el vídeo/imagen, es necesario pulsar el botón *clear data*, para evitar confundir datos o resultados de un análisis y otro. Una vez comenzado un análisis de nuevo es necesario realizar todo el proceso por completo con el nuevo vídeo.

2. **Change rotation angle**: Para que el programa analice correctamente el vídeo es necesario que las burbujas viajen desde el lado izquierdo de la imagen hacia el izquierdo. Por ello, en caso de que el vídeo no esté grabado de esa forma basta con girarlo en esta opción. Pueden introducirse grados tanto en negativo como en positivo, tomando el criterio convencional de signos para giros y momentos(contrario a las agujas del reloj es positivo). En la Figura 4.3 se muestra una imagen colocada con la orientación correcta y en la Figura 4.4 una imagen que

requeriría un giro de 90 grados.

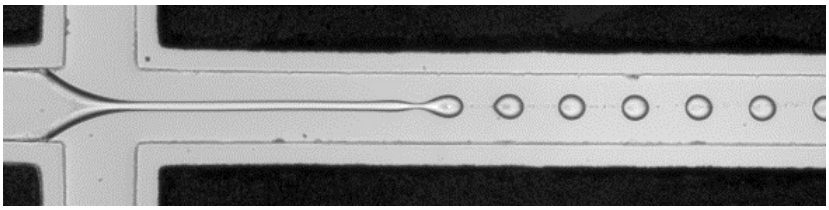

Figura 4.3: Imagen que muestra un fotograma de un vídeo colocado con la orientación adecuada.

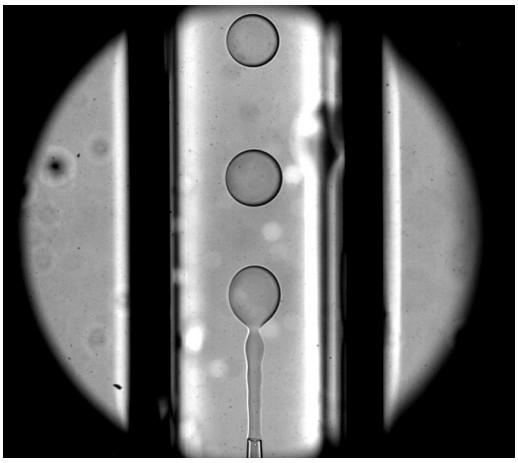

Figura 4.4: Imagen que muestra un fotograma de un vídeo colocado con la orientación incorrecta.

3. **Select region of interest**: En esta sección se muestra el primer fotograma en la gráfica superior si no estaba ya mostrado y se pide indicar las regiones de interés para el chorro y para las burbujas y la línea de referencia para el chorro. Esta línea de referencia será la que se utilice como punto de partida para medir la longitud del chorro. En los tres casos se piden dos puntos, para los dos primeros los extremos opuestos de un rectángulo que defina la región de interés, y para el segundo caso los dos puntos de una recta que defina la línea de referencia. En el último caso, la coordenada horizontal es recibida solo en el primer punto, y se coloca la misma pero en la coordenada vertical indicada en el segundo punto. De esta manera se fuerza a que la línea de referencia

sea perpendicular al chorro.

El objetivo de este módulo es reducir la región en la que buscar las burbujas y el chorro para optimizar el proceso, quitando la parte en la que estén las líneas del borde del conducto. En la Figura 4.5 se muestra un ejemplo de región de interés para las burbujas. En los casos en los que el enfoque del vídeo no sea el adecuado se recomienda seleccionar una región de interés lo más pequeña posible para evitar fallos del programa, y en los casos en los que haya manchas en la imagen se recomienda coger una región de interés en la que no se encuentren dichas manchas. El tamaño de la región de interés no tendrá incidencia en los resultados, simplemente se analizarán menos burbujas, las que pasen por esta región en el tiempo que dura el vídeo.

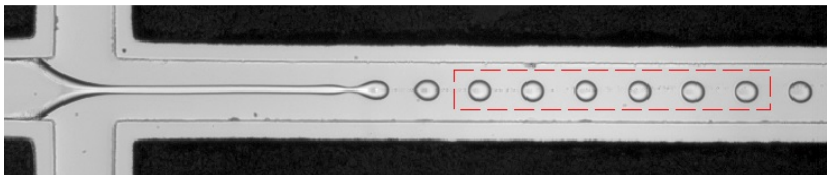

Figura 4.5: Imagen que muestra la región de intereés para las burbujas. Se observa que en este vídeo el enfoque es muy bueno y la frontera que delimita las burbujas es claramente diferente al fondo. En caso de no ser así se recomienda coger una región de interés más reducida.

En la Figura 4.6 se muestra un ejemplo de región de interés para el chorro y en la Figura 4.7 un ejemplo de línea de referencia para medir la longitud del chorro.

4. **Change resolution**: Permite seleccionar la resolución del vídeo en $\mu m/pixel$. Es un dato clave para el cálculo de los resultados, ya que todo lo que el programa calcule lo hará en píxeles y lo transformará a μm a través de este valor que se le indique.

5. **Change framerate**: Este es otro dato de importancia para el cálculo de la frecuencia de generación y velocidad de la burbuja. Simplemente hay que introducir la velocidad de adquisición en fps de nuestro vídeo.

6. **Change analysis range**: Si el vídeo lo permite y con el objetivo de reducir el tiempo de análisis puede elegirse no analizar todos los

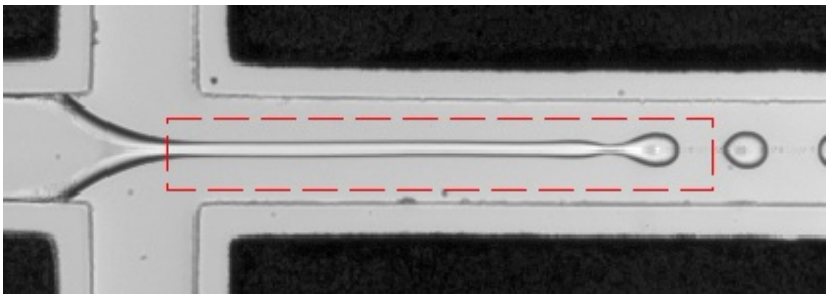

Figura 4.6: Imagen que muestra la región de intereés para el chorro.

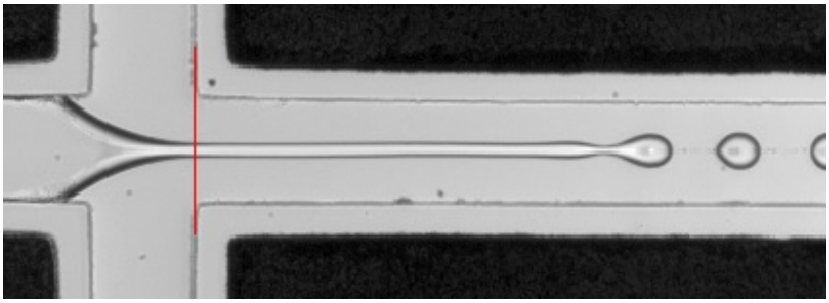

Figura 4.7: Imagen que muestra un ejemplo de línea de referencia para medir la longitud del chorro.

fotogramas que el vídeo contiene. Así puede por ejemplo analizarse solo los 100 primeros o analizarlos de diez en diez o con el incremento deseado. En este último caso hay que tener precaución. Si entre dos fotogramas que analizamos una burbuja viaja una distancia igual o superior a la distancia que separa dos burbujas el análisis no será correcto, por ello se debe prestar especial atención al incremento que escogemos. Para comprobar si el análisis ha sido correcto bastará con utilizar el módulo de ver película que se explicará más adelante.

7. **Object size**: En este apartado se pretende localizar una burbuja y comprobar que área tiene para utilizarlo como criterio de validación de burbujas en el apartado de análisis. Se mostrará en la pantalla superior el primer fotograma y deberá seleccionar dos puntos(al igual que en la región de interés) pero esta vez que delimiten una región en la que solo haya una burbuja. En caso de que la burbuja se detecte correctamente se mostrará la burbuja con su frontera pintada de rojo(Figura 4.8) .

En caso contrario no se mostrará nada y el área de la burbuja será el que había por defecto.

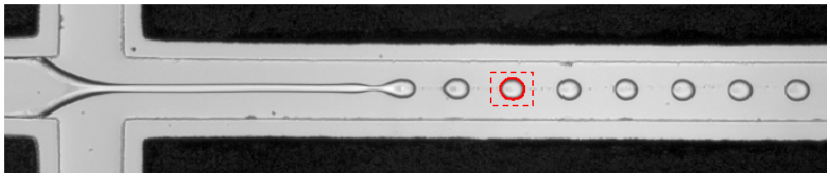

Figura 4.8: Imagen que muestra la selección de una burbuja para calcular su tamaño. Se muestra la burbuja correctamente detectada con su frontera pintada de rojo y en línea discontinua la región aproximada que habría que seleccionar para el correcto funcionamiento de este módulo.

8. **Analyze frames**: Una vez realizadas las correspondientes acciones del módulo preprocessor estamos en disposición de analizar los fotogramas. El programa recorre los fotogramas almacenando información acerca de los mismos necesaria para luego mostrar los resultados. Aparecerá una barra de estado que mostrarán los fotogramas que quedan por analizar. Una vez termine y desaparezca la barra podemos pasar al módulo de *postprocessor*. Se muestra en la Figura 4.9 un ejemplo de barra de estado.

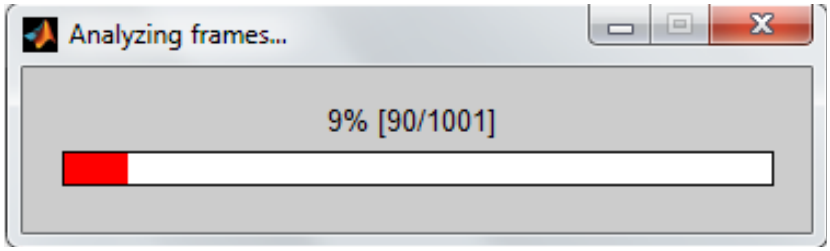

Figura 4.9: Imagen que muestra una barra de carga en el momento en que se está analizando el fotograma 90 de un total de 1001.

9. **Mean dameter, PDI & SD**: En este módulo se calcula el diámetro medio a lo largo de todos los fotogramas así como su ínidice de polidispersión y desviación estándar. El índice de polidispersión se calcula como:

$$PDI = 100 \cdot \frac{SD}{RadioMedio} \tag{4.1}$$

En la Figura 4.10 se muestran una gráfica de distribución de radios y otra de áreas que son mostradas por pantalla al ejecutar este módulo.

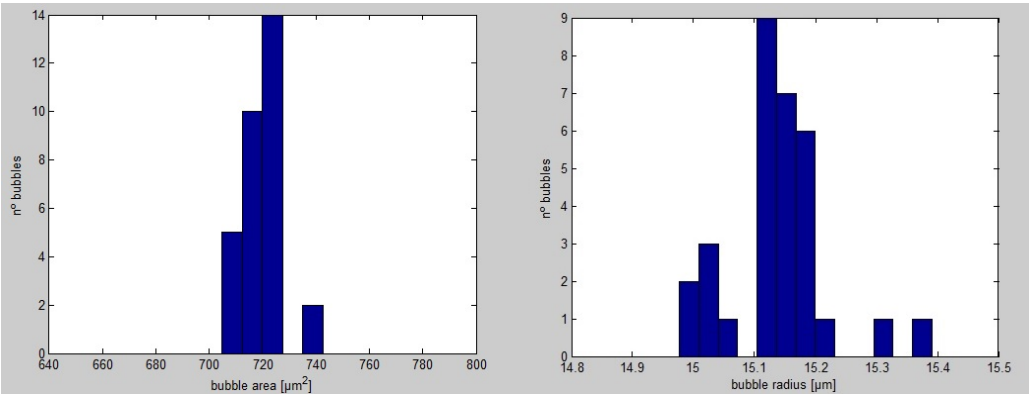

Figura 4.10: Distribución de áreas (1) y de radios (2).

10. **Generation frequency**: Al presionar este botón se mostrará en el cuadro de diálogo la frecuencia media de generación de burbujas. Además se muestra una gráfica en la que se representa las burbujas generadas frente al tiempo, como se ve en la Figura 4.11.

11. **Bubble velocity**: Al pulsar este botón en primer lugar se nos pide que caractericemos la sección del conducto. El motivo es que, además de calcular la velocidad de la burbuja también calcula el caudal interno(Q_i) y el caudal externo(Q_o). En caso de sección rectangular rellenar los apartados de W(ancho) y H(alto) del canal y dejar en cero el valor de R(radio). En caso de que sea una sección circular debe rellenarse solamente el apartado R dejando en cero el resto. Los caudales se calculan como:

    $Q_i = \frac{4}{3} \cdot \pi \cdot R_b^2 \cdot frecuencia$,donde R_b es el radio medio de la burbuja.

    $Q_o = Area_{\text{sección}} \cdot velocidad_{burbuja}$

12. **Jet length and diameter**: En este módulo se utiliza la información almacenada en *results* durante el módulo de análisis para mostrar la siguiente información:

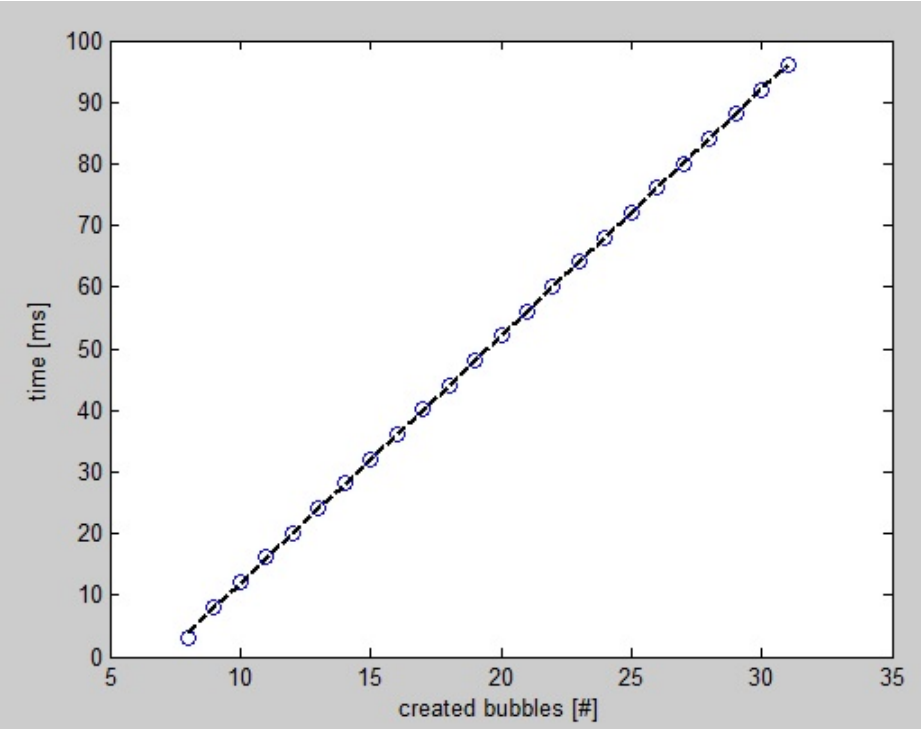

Figura 4.11: Burbujas creadas en sus instantes de tiempo y recta de mejor ajuste.

a) Longitud media calculada como la media de las longitudes obtenidas para cada fotograma analizado.

b) Índice de polidispersión y desviación estándar de las longitudes.

c) Diámetro medio. Cabe destacar la forma en que se calcula el diámetro del chorro para que el usuario lo tenga en cuenta. En cada fotograma se almacena la moda(valor más repetido) del diámetro del chorro a lo largo de toda su longitud. Lo que se muestra por pantalla es la media de esos valores a lo largo de todos los fotogramas. Dado que en muchos casos hay una parte del chorro que permanece muy constante será algo frecuente que la polidispersidad y la desviación estándar sean cero.

d) Índice de polidispersión y desviación estándar de los diámetros.

13. **Choose breakups**: En este módulo el usuario tiene la oportunidad de elegir las roturas de las burbujas que desee para que se guarden,

al ejecutar la opción de salvar resultados, ficheros con las coordenadas de las fronteras de los chorros que seleccione. En la Figura 4.12 se muestra el cuadro de diálogo mostrado para que el usuario aporte la información descrita.

14. **Choose times**: Además de poder seleccionar roturas para que se almacenen sus fronteras posteriormente, el usuario también tiene la posibilidad de guardar distintos instantes de tiempo de una determinada rotura. Para ello, se le pide al usuario por pantalla que indique la rotura de la cual quiere guardar distintos instantes de tiempo, y las divisiones de tiempo que desea.

    Para las divisiones de tiempo, se ha decidido estructurar a través de la variable adimensional *tau*, que se calculará de la siguiente manera:

$$tau = \frac{i-1}{Divisiones\,de\,tiempo} \tag{4.2}$$

    Donde i es el índice del componente del vector. De esta manera, los fotogramas que se guardarán serán el total de fotogramas en los que aparece la rotura seleccionada multiplicado por el vector tau, o visto de una manera más sencilla:

$$Incremento = \frac{Fotograma\,final - Fotograma\,incial}{Divisiones\,de\,tiempo} \tag{4.3}$$

$$Fotogramas\,guardados = Fotograma\,incial + Incremento * (i-1) \tag{4.4}$$

    En la Figura 4.12 se muestran los cuadros de diálogo que se le muestran al usuario para que introduzca la información indicada en este subapartado.

15. **Watch movie**: Como ya se comentó al principio de esta sección, el objetivo de esta parte del programa es dotar al usuario de una herramienta que le permita, de forma rápida y eficaz, comprobar si los resultados calculados son o no válidos. Al pulsar el botón aparecerá un cuadro de diálogo en el que tenemos que especificar las id de las burbujas que queremos que se muestren coloreadas. Una vez lo seleccionemos y pulsemos 'ok' comenzarán a mostrarse los fotogramas con las fronteras de las burbujas y de los chorros coloreados con sus id

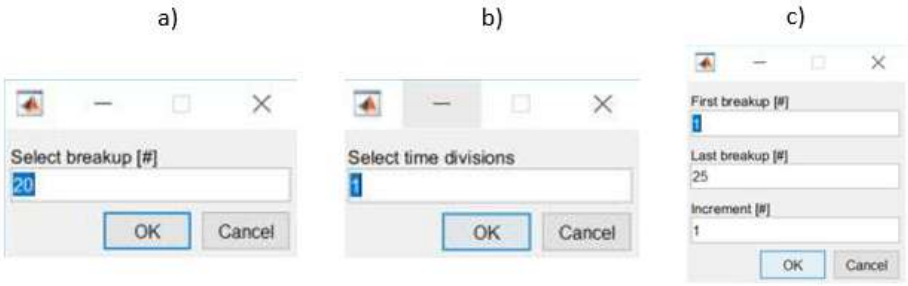

Figura 4.12: Cuadros de diálogo mostrados para que el usuario elija la rotura de la que quiere guardar los instantes de tiempo (a), el número de divisiones de tiempo (b) y las roturas que se quieran guardar (c).

colocados en la parte inferior de cada burbuja y en el principio del chorro.

Bastaría con comprobar que la numeración sea correcta y las fronteras estén bien delimitadas para tener una gran garantía de que el análisis se ha efectuado correctamente.

16. **Save results**: El objetivo de este módulo es generar archivos de salida que el usuario pueda mantener y utilizar una vez se cierre el programa. Los ficheros de texto se guardrán en la carpeta en la que se encuentra el vídeo con los nombres que se detallarán a continuación. Los ficheros que se generan son los siguientes:

    a) Fichero con un resumen de todos los parámetros introducidos por el usuario durante el pre-proceso y con los resultados que el usuario haya pedido al programa calcular. El nombre de este fichero es '*NombreVídeo*Outputfile.txt'.

    b) Fichero con los radios medios de todas las burbujas que aparecen en el vídeo. El nombre de este fichero es '*NombreVídeo*Radius.txt'.

    c) Tantos ficheros como fronteras de chorros se haya pedido guardar. La regla que sigue el nombre es la siguiente:

        - En primer lugar coloca el nombre del vídeo analizado.

- A continuación coloca la burbuja a la que corresponde la frontera.
- Por último coloca el tau (fracción de tiempo entre roturas) correspondiente.

Un ejemplo de nombre de fichero de frontera seía 'f5v500.avib06tau0.00', que corresponde a la sexta burbuja y tau 0 del vídeo 'f5v500.avi'.

```
--> INPUT PARAMETERS
Resolution: 1.560000e+00 µm/px
Framerate: 20000 fps
First frame:   1
Last frame:    1001
Increment:     10

--> RESULTS
Nº of bubbles captured: 32
Mean bubble radius: 9.712121e+00 µm
Bubble radius PDI: 0.6826 %
Bubble radius SD: 0.10342 µm
Mean generation frequency: 4.950495e+02 Hz
Mean jet length: 338.7053 µm
Jet length PDI: 4.2763 %
Jet length SD: 9.2847 µm
Mean jet diameter: 9.36 µm
Jet diameter PDI: 0 %
Jet diameter SD: 0 µm
Mean bubble velocity: 32767 µm/s
Qi: 25.9631 µl/h
Qo: 412.8642 µl/h
```

Figura 4.13: Ejemplo del archivo de texto que se genera con los parámetros de análisis y los resultados obtenidos.

En caso de desearlo en esta parte del código se encuentran dos líneas comentadas que en el caso no estarlo generarían dos archivos con formato '.m' que contendrían las variables *param* y *results* ya descritas, que permitirían evitar realizar el apartado de pre-proceso y análisis. Bastaría con colocar esas dos variables en la carpeta del vídeo en cuestión y el programa las leería automáticamente. Debe tenerse cuidado

porque en caso de haber más variables de este tipo en la carpeta de análisis se induciría a error.

Una vez comprendida esta sección el usuario está en disposición de analizar los vídeos de forma correcta teniendo en cuenta las precauciones comentadas.

Para completar el análisis y facilitar aún más al usuario la representación de resultados y extracción de conclusiones, se ha desarrollado una interfaz que permite representar las fronteras de los chorros directamente en el formato extraído en el programa principal.

Esta interfaz, representada en la Figura 4.14, dispone de tres gráficas en las que se pueden representar tantos chorros superpuestos como se deseen. La gráfica representa las fronteras en micrómetros, adapta los límites al chorro en cuestión, superpone tantos chorros como se seleccionen y anãde la leyenda con los nombres de los ficheros (que por la definición que se les ha dado contienen información sobre la frecuencia y el voltaje). Dispone de tres botones con las siguientes funcionalidades:

1. **Plot jet**: Con este botón seleccionamos los ficheros que contienen las fronteras de los chorros a representar. Podemos seleccionar tantos como deseemos. En primer lugar, el programa abre un cuadro de diálogo para seleccionar los archivos con contienen las fronteras. A cotinuación, una vez dispone de los ficheros, realiza las siguientes operaciones para representar los chorros:

   - Cambio de coordenadas para representar el chorro con la línea media en 'y=0'.
   - Cambio de coordenadas para representar los chorros solapados en el inicio o en el final del chorro en función de la opción que esté seleccionada (funcionalidad número cuatro).
   - Configura la leyenda con los nombres de los ficheros.
   - Configura los límites horizontales.
   - Configura los límites del eje vertical de la misma manera que el eje horizontal, con la diferencia de que el chorro está centrado en cero, por lo que hay que fijar un límite superior y otro inferior.

2. **Save figure**: Guarda la representación actual en formato '*.fig*'.

3. **Clear figure**: Elimina la representación actual y resetea la configuración de ejes, leyenda, etc.

4. **Set reference**: Dos botones seleccionables que definen si los chorros deben representarse solapados al inicio del chorro o al final del chorro.

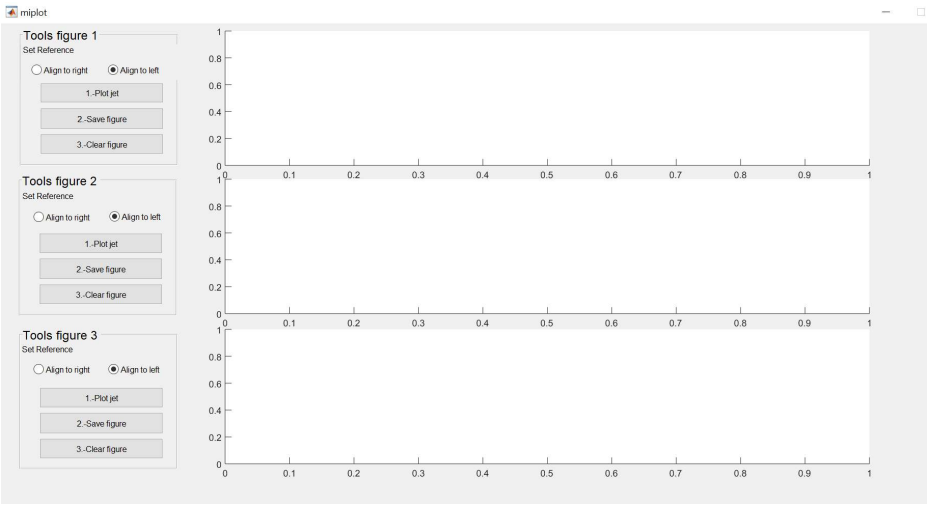

Figura 4.14: Interfaz de usuario que permite representar las fronteras de los chorros seleccionando los ficheros exxtraídos en el programa principal.

De esta manera, completamos las funcionalidades del código principal con una interfaz que facilita la representación de chorros en distintas condiciones de frecuencia, conductividad y voltaje en distintos instantes de tiempo y para distintas roturas, quedando para el usuario la tarea de extracción de conclusiones en base a los resultados obtenidos y representados.

Capítulo 5

Evolución de la longitud del chorro al variar las condiciones experimentales

5.1. Introducción y descripción de las condiciones del análisis

Por último, y para finalizar el trabajo realizado en el trabajo fin de máster plasmado en esta memoria, se va a realizar un an álisis sobre experimentos reales realizados por Javier Alzaga en su Trabajo Fin de Grado. Nos vamos a centrar en la parte más novedosa aportada en este Trabajo Fin de Máster, que es la referente al análisis del chorro. Para ello, vamos a estudiar diferentes vídeos con diferentes condiciones de frecuencia, voltaje y conductividad. Los parámetros comunes de los vídeos son los siguientes:

1. Resolución: 2 micrómetros por píxel.
2. Velocidad de adquisición: 20000 fotogramas por segundo.
3. Caudal de la fase contínua (Qi): 400 microlitros por hora.
4. Caudal de la fase dispersa (Qo): 50 microlitros por hora.

En cuanto a los análisis que se van a hacer, son los siguientes:

1. Un primer análisis en el que se fija la conductividad en 0.3 mS/m, y para dos valores distintos del voltaje se analizará la influencia de la frecuencia:
   - Para V = 700V, valores de frecuencia de 5, 10, 20, 30, 40 y 50 Hz.
   - Para V = 100V, valores de frecuencia de 5, 10, 20, 30, 40 y 50 Hz.
2. Un segundo análisis en el que se fija la conductividad en 0.3 mS/m y para dos valores distintos de frecuencia se analizará la influencia del voltaje:
   - Para f = 10Hz, valores de voltaje de 700, 800, 900 y 1000 V.
   - Para f = 50Hz, valores de voltaje de 700, 800, 900 y 1000 V.
3. Por último, se analizará la influencia de la conductividad para una frecuencia de 50 Hz y distintos valores del voltaje:
   - Para V = 700V, valores de conductividad de 0.3, 1, 3, 10 y 30 mS/m.
   - Para V = 1000V, valores de conductividad de 0.3, 1, 3, 10 y 30 mS/m.

5.2. Análisis de la influencia de la frecuencia

Analizamos el primero de los casos expuestos en la introducción, en el que observamos la evolución de la longitud del chorro al variar la frecuencia para dos valores distintos de voltaje (700V y 1000V) y con la conductividad fijada en 0.3 mS/m. En la Figura 5.1 se representa la evolución de la longitud del chorro para las distintas condiciones de frecuencia y para los dos voltajes analizados, y en las Figuras 5.2 y 5.3 se han representado los chorros en las distintas condiciones de frecuencia y voltaje estudiados para distintos instantes de tiempo entre dos roturas diferentes.

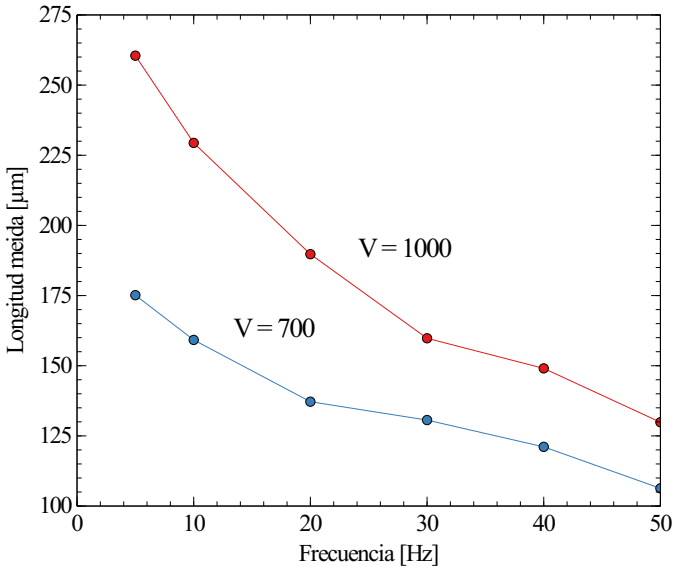

Figura 5.1: Gráfico en el que se muestra la evolución de la longitud media del chorro para los distintos experimentos al aumentar la frecuencia.

De las figuras que muestran los resultados estudiados podemos sacar la conclusión de que **la relación existente entre la longitud del chorro y la frecuencia es inversamente** proporcional, es decir, que al aumentar la frecuencia estaremos reduciendo la longitud del chorro que genera las burbujas. Además, también podemos observar que esta relación es **más acentuada para valores del voltaje altos**. La pendiente de la gráfica es mayor para el caso de 1000 V.

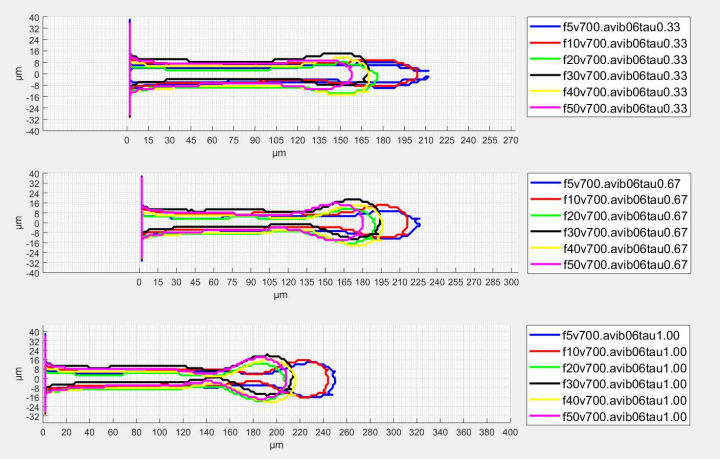

Figura 5.2: Superposición de los chorros para distintos valores de frecuencia y distintos instantes de tiempo para el voltaje fijado en 700V y la conductividad fijada en 0.3 mS/m.

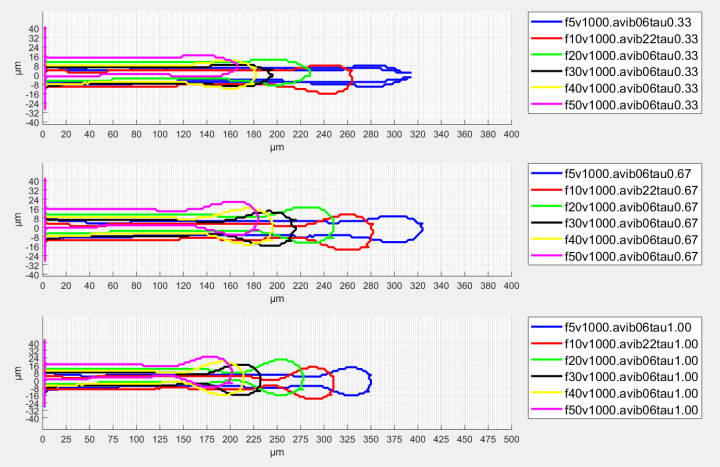

Figura 5.3: Superposición de los chorros para distintos valores de frecuencia y distintos instantes de tiempo para el voltaje fijado en 1000V y la conductividad fijada en 0.3 mS/m.

5.3. Análisis de la influencia del voltaje

Analizamos el segundo de los casos expuestos en la introducción, en el que observamos la evolución de la longitud del chorro al variar el voltaje para dos valores distintos de frecuencia (10Hz y 50Hz) y con la conductividad fijada en 0.3 mS/m. En la Figura 5.4 se representa la evolución de la longitud del chorro para las distintas condiciones de voltaje y para las dos frecuencias analizadas, y en las Figuras 5.5 y 5.6 se han representado los chorros en las distintas condiciones de voltaje y frecuencia estudiados para distintos instantes de tiempo entre dos roturas diferentes.

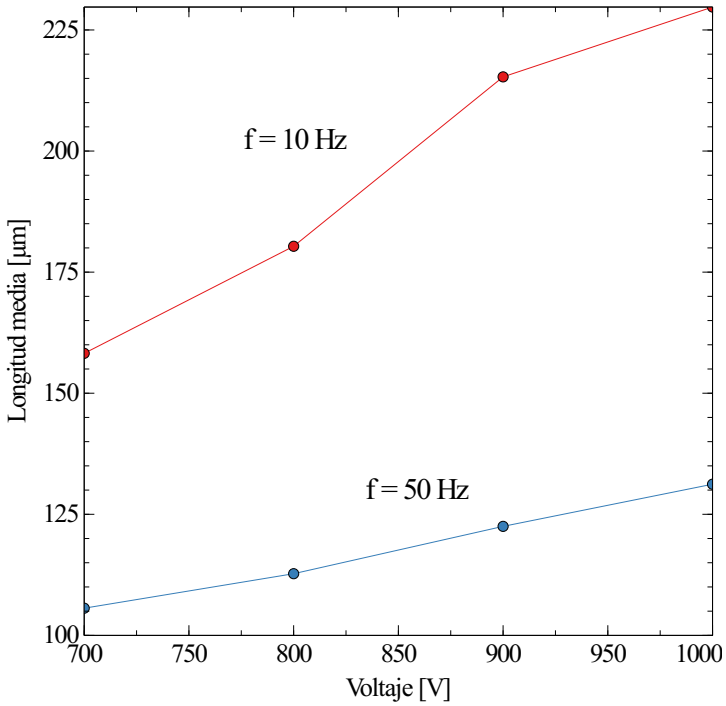

Figura 5.4: Gráfico en el que se muestra la evolución de la longitud media del chorro para los distintos experimentos al aumentar el voltaje.

De las figuras que muestran los resultados estudiados podemos sacar la conclusión de que **la relación existente entre la longitud del chorro y el voltaje es directamente proporcional**, es decir, que al aumentar el voltaje estaremos aumentando la longitud del chorro que genera las burbujas. Además, también podemos observar que esta relación es **más acentuada**

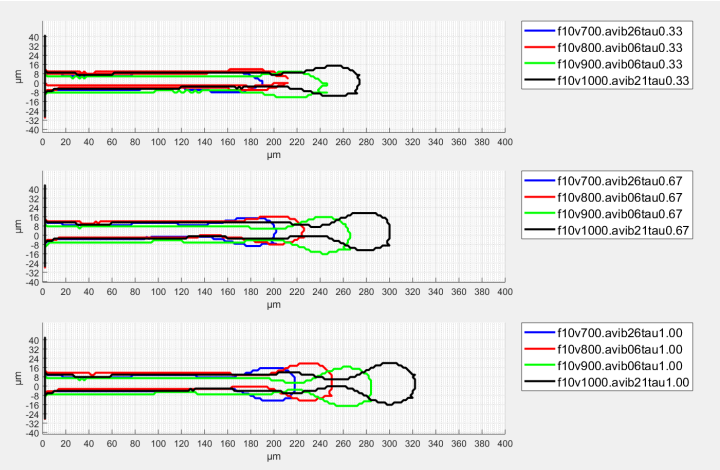

Figura 5.5: Superposición de los chorros para distintos valores de voltaje y distintos instantes de tiempo para la frecuencia fijada en 10Hz y la conductividad fijada en 0.3 mS/m.

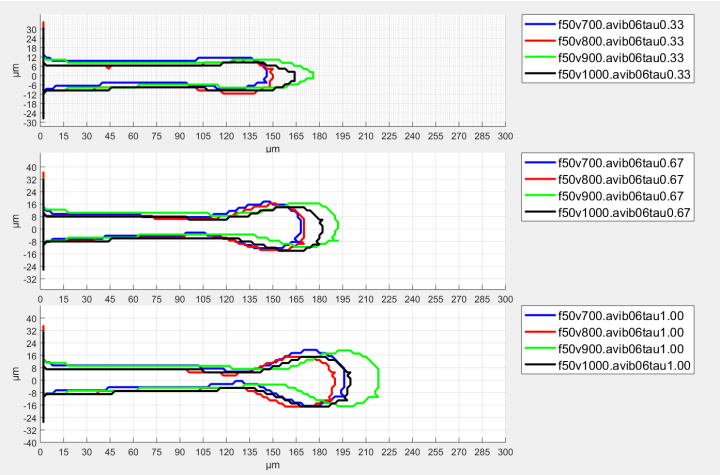

Figura 5.6: Superposición de los chorros para distintos valores de voltaje y distintos instantes de tiempo para la frecuencia fijada en 50Hz y la conductividad fijada en 0.3 mS/m.

para valores de frecuencia bajos. La pendiente de la gráfica es mayor para el caso de 10Hz.

5.4. Análisis de la influencia de la conductividad

Analizamos el tercero de los casos expuestos en la introducción, en el que observamos la evolución de la longitud del chorro al variar la conductividad para dos valores distintos de voltaje (700V y 1000V) y con la frecuencia fijada en 50 Hz. En la Figura 5.7 se representa la evolución de la longitud del chorro para las distintas condiciones de conductividad y para los dos voltajes analizados, y en las Figuras 5.8 y 5.9 se han representado los chorros en las distintas condiciones de conductividad y voltaje estudiados para distintos instantes de tiempo entre dos roturas diferentes.

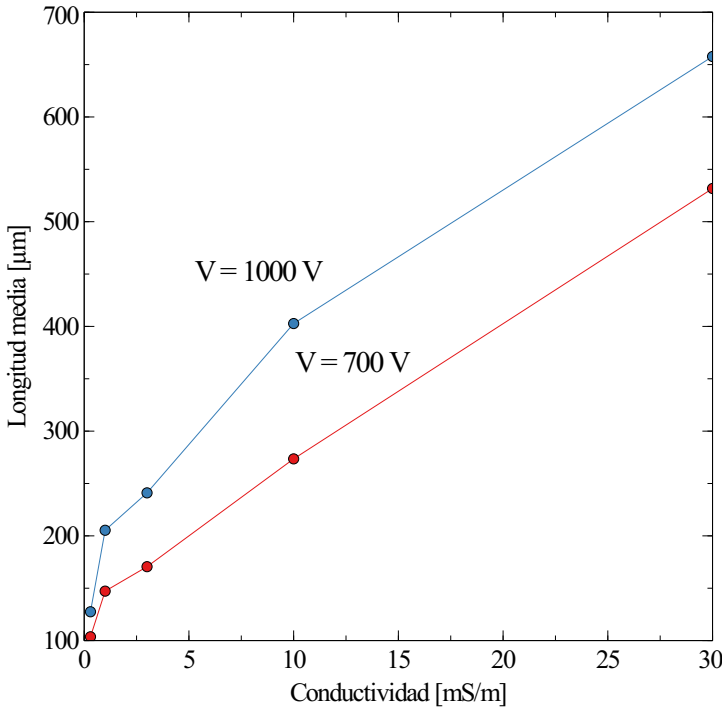

Figura 5.7: Gráfico en el que se muestra la evolución de la longitud media del chorro para los distintos experimentos al aumentar la conductividad.

De las figuras que muestran los resultados estudiados podemos sacar la conclusión de que **la relación existente entre la longitud del chorro y la conductividad es directamente proporcional**, es decir, que al aumentar la conductividad estaremos aumentando la longitud del chorro

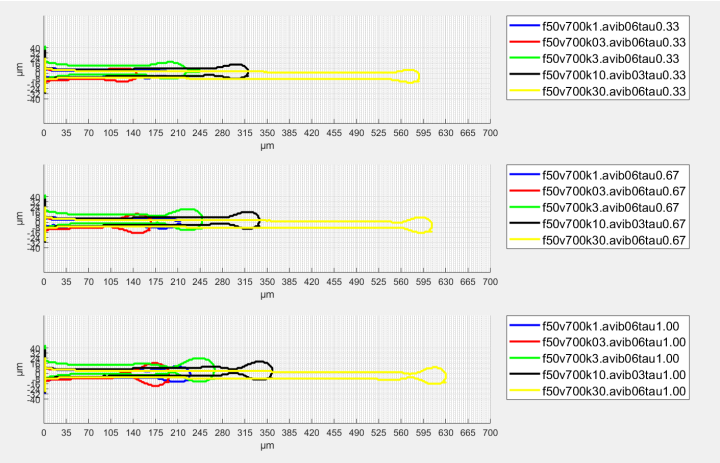

Figura 5.8: Superposición de los chorros para distintos valores de conductividad y distintos instantes de tiempo para el voltaje fijado en 700V y la frecuencia fijada en 50Hz.

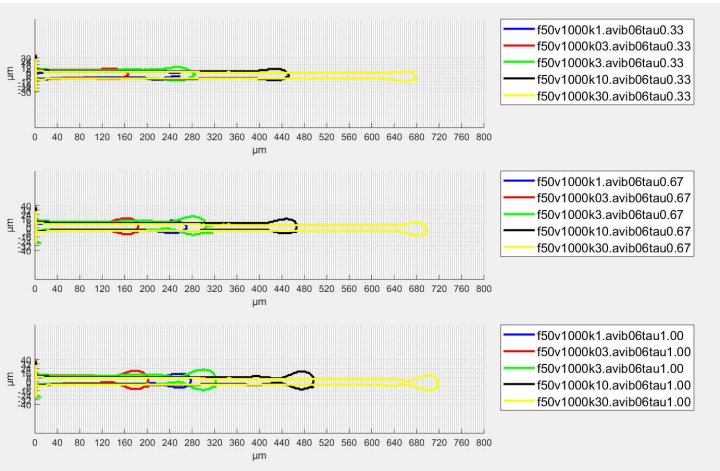

Figura 5.9: Superposición de los chorros para distintos valores de conductividad y distintos instantes de tiempo para el voltaje fijado en 1000V y la frecuencia fijada en 50Hz.

que genera las burbujas. Además, también podemos concluir que en este caso no se observa un aumento de la pendiente al cambiar los valores de voltaje, ya que las gráficas son paralelas.

Capítulo 6

Conclusiones

En este trabajo se ha llevado a cabo la programación de un código para el procesado de vídeos generados en microfluídica. En particular permite, partiendo de vídeos o imágenes, calcular los siguientes parámetros:

- Índice de polidispersión y desviación estándar de tamaños.
- Longitud y diámetro del chorro.
- Frecuencia de generación de burbujas.
- Velocidad de la burbuja.
- Caudales interior y exterior.

Y permite extraer los siguientes archivos una vez concluido el análisis:

1. Fichero con un resumen de todos los parámetros utilizados para el análisis y de los resultados extraídos.
2. Fichero con los diámetros medios de todas las burbujas analizadas en el vídeo.
3. Ficheros con las coordenadas de las fronteras de los chorros que se hayan seleccionado en las opciones "Choose breakupsz "Choose times".

Además, gracias al orden y la cantidad de información útil que contienen las variables que genera el código, el programa es muy versátil y permite al usuario, como se ha demostrado en el presente Trabajo Fin de Máster, introducir módulos y cálculos nuevos de forma sencilla y rápida.

Para facilitar la presentación de las fronteras extraídas en el formato comentado en el desarrollo de la presente memoria, también se ha creado una interfaz de usuario que permite representar los chorros de una manera sencilla y personalizada para los casos de estudio.

Respecto al análisis realizado en la Sección 5, se pueden extraer las siguientes conclusiones respecto a la influencia del voltaje, la conductividad y la frecuencia en la longitud del chorro:

- La longitud del chorro aumenta al disminuir la frecuencia. Este efecto es más acusado cuanto mayor es el voltaje.

- La longitud del chorro aumenta al aumentar el voltaje. Este efecto es más acusado cuanto menor es la frecuencia.
- La longitud del chorro aumenta al aumentar la conductividad. En este caso, no se observa un aumento de la influencia al aumentar ninguno de los parámetros.

Apéndices

Apéndice A

Bibliografía

Bibliografía

[1] E. Castro-Hernández, *Análisis de los mecanismos de generación y rotura de burbujas y gotas en corrientes gas-líquido y líquid líquido.* PhD thesis, Universidad de Sevilla, 2011.

[2] A. Sánchez Evangelio, "Selective withdrawal: Reproducción del fenómeno, modelo teórico y predicción numérica," Master's thesis, Escuela Técnica Superior de Ingeniería de Sevilla, 2103.

[3] P. Fernández Pisón, "Rotura de chorros para enriquecimiento de agentes de contraste," Master's thesis, Escuela Técnica Superior de Ingeniería de Sevilla & University of Twente, 2015.

[4] S. L. Anna, N. Bontoux, and H. A. Stone, "Formation of dispersions using Flow Focusing in microchannels," *Appl. Phys. Lett.*, vol. 82, pp. 364–366, 2003.

Apéndice B

Código empleado

Table of Contents

..... 1
OPENING 2
--- Executes on button press in ChangeImage1. 4
--- Executes on button press in ChangeAngle2. 6
--- Executes on button press in Roi3. 7
--- Executes on button press in Resolution4. 10
--- Executes on button press in Framerate5. 11
--- Executes on button press in AnalysisRange6. 11
--- Executes on button press in ObjectSize7. 13
--- Executes on button press in analyze8. 14
Bubble analysis 16
Jet analysis 16
--- Executes on button press in polidispersity9. 18
--- Executes on button press in diameter10. 18
--- Executes on button press in frecuency11. 20
--- Executes on button press in velocity12. 21
--- Executes on button press in jetlength13. 24
--- Executes on button press in movie14. 25
--- Executes on button press in save15. 27
--- Executes on button press in pushbutton16. 31
--- Executes on button press in BubbleRange15. 31
--- Executes on button press in TimeRange16. 31


```
function varargout = prueba1(varargin)

% PRUEBA1 MATLAB code for prueba1.fig


% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @prueba1_OpeningFcn, ...
                   'gui_OutputFcn',  @prueba1_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

Undefined function or variable 'nrfiles'.

Error in analizar2>prueba1_OpeningFcn (line 87)
```

```
            param.frnum = 1:nrfiles;                % (default) frames
 to be analyzed

Error in gui_mainfcn (line 220)
    feval(gui_State.gui_OpeningFcn, gui_hFigure, [],
 guidata(gui_hFigure), varargin{:});

Error in analizar2 (line 20)
    gui_mainfcn(gui_State, varargin{:});
```

OPENING

--- Executes just before prueba1 is made visible.

```
function prueba1_OpeningFcn(hObject, eventdata, handles, varargin)

%-- Put the menu in the center of the screen
scrsz=get(0,'ScreenSize');
pos_act=get(gcf,'Position');
xr=scrsz(3)-pos_act(3);
xp=round(xr/2);
yr=scrsz(4)-pos_act(4);
yp=round(yr/2);
set(gcf,'Position',[xp yp pos_act(3) pos_act(4)]);

% Choose default command line output for prueba1
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

global sdir files param background results results_profile info mov
 format_index filename;%make the main variables global
results = [];                   % (initial) results from data analysis
results_profile= [];
background.imbg = [];           % (optional) background image

% -> parameters to be saved
param.resolution = 1.56;        % (default) image resolution [um/px]
param.framerate = 20000;         % (default) framerate [fps]
param.rotate = 0;               % (default) rotation angle
param.object_area = 20;         % (default) bubble area
param.frnum = [];               % (initial) frame numbers to be
 analyzed
param.bbnum = [];               % (initial) bubble numbers to be
 analyzed in bubble range
param.bbnum2 = [];              % (initial) bubble numbers found in
 time range
param.bgframe = -1;             % (optional) background image
param.roi = [];                 % (initial) bubbles region of interst
param.roi_jet = [];             % (initial) jet region of interest
param.ref_line = [];            % (initial) reference line for jet
 length calcs
```

```
% -> set source directory and read files
           filtro  = {'*.avi' 'avi' ; '*.tif' 'tif
comprimido';'*.tif' 'tif descomprimido, coger primer
fotograma';'*.tif' 'imagen tif'};
           [filename,sdir,format_index]    = uigetfile(filtro,'Select
video or image');   % source directory


           %-- Find out the format of the video which is going to be
           %analyzed

          if format_index == 1 %.avi
              mov = VideoReader(strcat(sdir,filename));
              nrfiles = mov.NumberOfFrames;
          end

          if format_index == 2 %.tif compressed
              info = imfinfo(strcat(sdir,filename));
              nrfiles = length(info);
          end

          if format_index == 3 %.tif descompressed
              files = dir(strcat(sdir,'*.tif'));
              nrfiles = length(files);
          end

          if format_index == 4 %.tif image
              nrfiles = 1;
          end

           param.frnum = 1:nrfiles;                % (default) frames
to be analyzed

           % -> read first image
           if format_index == 1 %.avi
              im = read(mov,1);
          end

          if format_index == 2 %.tif compressed
              im = imread(strcat(sdir,filename),1);
          end

          if format_index == 3 %.tif descompressed
              im = imread(strcat(sdir,files(1).name));
          end

          if format_index == 4 %.tif image
              im = imread(strcat(sdir,filename));
          end

           param.roi = round([1 size(im,2) 1 size(im,1)]);          %
(default) region of interest
           param.roi_jet = round([1 size(im,2) 1 size(im,1)]);
```

```
            % -> read and overwrite parameter
            if exist([sdir,'parameters.mat'],'file')
                load([sdir,'parameters.mat']);
%                 fprintf([' -> Parameter file loaded.\n'])
%                 param
            end

            if exist([sdir,'background.mat'],'file')
                load([sdir,'background.mat']);
%                 fprintf([' -> Background image loaded.\n'])
            end

            if exist([sdir,'results.mat'],'file')
                load([sdir,'results.mat']);
%                 fprintf([' -> Results loaded.\n'])
%                 results
            end
            axes(handles.axes1);%show axes1 the first frame
            imshow(im);
            axes(handles.axes2);%show an empty page in axes 2
            im=imread('LogoEtsi.png');
            imshow(im);


% --- Outputs from this function are returned to the command line.
function varargout = prueba1_OutputFcn(hObject, eventdata, handles)

% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;
```

--- Executes on button press in ChangeImage1.

```
function ChangeImage1_Callback(hObject, eventdata, handles)

%This part is like the one which is executed when opening the program,
 the
%objective is to be able to change the video without closing the
 program

global sdir files param background results results_profile info mov
 format_index filename;%make the main variables global

set(handles.text1,'String','');
set(handles.text2,'String','');
set(handles.text3,'String','');
set(handles.text4,'String','');

results = [];                  % (initial) results from data analysis
```

```
results_profile= [];
background.imbg = [];           % (optional) background image

% -> parameters to be saved
param.resolution = 1.56;        % (default) image resolution [um/px]
param.framerate = 20000;         % (default) framerate [fps]
param.rotate = 0;               % (default) rotation angle
param.object_area = 20;         % (default) bubble area
param.frnum = [];               % (initial) frame numbers to be
 analyzed
param.bbnum = [];               % (initial) bubble numbers to be
 analyzed in bubble range
param.bbnum2 = [];              % (initial) bubble numbers found in
 time range
param.bgframe = -1;             % (optional) background image
param.roi = [];                 % (initial) bubbles region of interst
param.roi_jet = [];             % (initial) jet region of interest
param.ref_line = [];            % (initial) reference line for jet
 length calcs
 % -> set source directory and read files
            filtro  = {'*.avi' 'avi' ; '*.tif' 'tif
 comprimido';'*.tif' 'tif descomprimido, coger primer
 fotograma';'*.tif' 'imagen tif'};
            [filename,sdir,format_index]    = uigetfile(filtro,'Select
 video or image');   % source directory


            %-- Find out the format of the video which is going to be
            %analyzed

           if format_index == 1 %.avi
               mov = VideoReader(strcat(sdir,filename));
               nrfiles = mov.NumberOfFrames;
           end

           if format_index == 2 %.tif compressed
               info = imfinfo(strcat(sdir,filename));
               nrfiles = length(info);
           end

           if format_index == 3 %.tif descompressed
               files = dir(strcat(sdir,'*.tif'));
               nrfiles = length(files);
           end

           if format_index == 4 %.tif image
               nrfiles = 1;
           end

            param.frnum = 1:nrfiles;                % (default) frames
 to be analyzed

            % -> read first image
            if format_index == 1 %.avi
```

```
            im = read(mov,1);
        end

        if format_index == 2 %.tif compressed
            im = imread(strcat(sdir,filename),1);
        end

        if format_index == 3 %.tif descompressed
            im = imread(strcat(sdir,files(1).name));
        end

        if format_index == 4 %.tif image
            im = imread(strcat(sdir,filename));
        end

         param.roi = round([1 size(im,2) 1 size(im,1)]);          %
 (default) region of interest
         param.roi_jet = round([1 size(im,2) 1 size(im,1)]);

         % -> read and overwrite parameter
         if exist([sdir,'parameters.mat'],'file')
             load([sdir,'parameters.mat']);
%              fprintf([' -> Parameter file loaded.\n'])
%              param
         end

         if exist([sdir,'background.mat'],'file')
             load([sdir,'background.mat']);
%              fprintf([' -> Background image loaded.\n'])
         end

         if exist([sdir,'results.mat'],'file')
             load([sdir,'results.mat']);
%              fprintf([' -> Results loaded.\n'])
%              results
         end
         axes(handles.axes1);%show axes1 the first frame
         imshow(im);
         axes(handles.axes2);%show an empty page in axes 2
         im=imread('LogoEtsi.png');
         imshow(im);
```

--- Executes on button press in ChangeAngle2.

```
function ChangeAngle2_Callback(hObject, eventdata, handles)

global sdir files param mov format_index filename;
% Change the angle of the frames in case it is necessary. The image
 should
% be oriented so that the droplets travel from the left part of the
 image
% to the right one.
         if format_index == 1 %.avi
```

```
            im0 = read(mov,1);
        end

        if format_index == 2 %.tif compressed
            im0 = imread(strcat(sdir,filename),1);
        end

        if format_index == 3 %.tif descompressed
            im0 = imread(strcat(sdir,files(1).name));
        end

        if format_index == 4 %.tif image
            im0 = imread(strcat(sdir,filename));
        end

         im = im0;

         im = imrotate(im,param.rotate,'bilinear','crop');
 % rotate image
        axes(handles.axes1);
         imshow(im)

         rotate_valid = param.rotate;     % original (valid)
rotation angle
         button = questdlg('Rotation ok?','','Yes','No','Yes');
         while ~strcmp(button,'Yes')
             answer = inputdlg({'Enter rotation angle: [°
CCW]'},'Input for image rotation',1,{num2str(rotate_valid)});
             rotate = str2num(answer{1});

             if isnumeric(rotate)
                 im = imrotate(im0,rotate,'bilinear','crop');
                 axes(handles.axes1);
                 imshow(im)
                 rotate_valid = rotate;   % new (valid) rotation
angle
             else
                 rotate = rotate_valid; % set previous (valid)
rotation angle
             end

             button = questdlg('Rotation ok?','','Yes','No','Yes');
         end
         param.rotate = rotate_valid;  % update rotation angle
```

--- Executes on button press in Roi3.

```
function Roi3_Callback(hObject, eventdata, handles)

%Select the bubble region of interest

global sdir files param mov format_index filename;
```

```
            if format_index == 1 %.avi
                im = read(mov,1);
            end

            if format_index == 2 %.tif compressed
                im = imread(strcat(sdir,filename),1);
            end

            if format_index == 3 %.tif descompressed
                im = imread(strcat(sdir,files(1).name));
            end

            if format_index == 4 %.tif image
                im = imread(strcat(sdir,filename));
            end

            im = imrotate(im,param.rotate,'bilinear','crop');
  % rotate image

            axes(handles.axes1);
            imshow(im)
            hold on, plot([param.roi(1), param.roi(2),...
                    param.roi(2),param.roi(1),...
                    param.roi(1)],[param.roi(3),...
                    param.roi(3),param.roi(4),...
                    param.roi(4), param.roi(3)], 'r--'), hold off

%             axes(handles.axes2)
%             mix=imread('mix.png');
%             imshow(mix)

            button = questdlg('Bubble region of interest
 ok?','','Yes','No','Yes');
            while ~strcmp(button,'Yes')
                [x y] = ginput(2);
  % get two coordinates for the roi
                  x = sort(x);
                  y = sort(y);
                if x(1) < 1,         x(1) = 1;          end
  % validate the coordinates
                if x(2) > size(im,2), x(2) = size(im,2);  end
                if y(1) < 1,         y(1) = 1;          end
                if y(2) > size(im,1), y(2) = size(im,1);  end

                param.roi = int16([x(1) x(2) y(1) y(2)]);
        % update the roi
                axes(handles.axes1);
                imshow(im)
                hold on, plot([param.roi(1), param.roi(2),...
                    param.roi(2),param.roi(1),...
                    param.roi(1)],[param.roi(3),...
                    param.roi(3),param.roi(4),...
                    param.roi(4), param.roi(3)], 'r--'), hold off
```

```
                button = questdlg('Region of interest
 ok?','','Yes','No','Yes');
            end
            texto1=' -> Region of interest is set';%show the message

                set(handles.text1,'String',texto1);
                set(handles.text2,'String',' ');
                set(handles.text3,'String',' ');
                set(handles.text4,'String',' ');

%                 axes(handles.axes2);%show an empty page in axes 2
%                 im=imread('LogoEtsi.png');
%                 imshow(im);

 %Select the jet region of interest

 im = imrotate(im,param.rotate,'bilinear','crop');
             roi_jet = param.roi_jet;
             axes(handles.axes1);
            imshow(im)
            hold on, plot([roi_jet(1), roi_jet(2),...
                    roi_jet(2),roi_jet(1),...
                    roi_jet(1)],[roi_jet(3),...
                    roi_jet(3),roi_jet(4),...
                    roi_jet(4), roi_jet(3)], 'r--'), hold off

            button = questdlg('Jet region of interest
 ok?','','Yes','No','Yes');
            while ~strcmp(button,'Yes')
                [x y] = ginput(2);
 % get two coordinates for the roi
                  x = sort(x);
                  y = sort(y);
                if x(1) < 1,         x(1) = 1;          end
 % validate the coordinates
                if x(2) > size(im,2), x(2) = size(im,2);  end
                if y(1) < 1,         y(1) = 1;          end
                if y(2) > size(im,1), y(2) = size(im,1);  end

                roi_jet = int16([x(1) x(2) y(1) y(2)]);
    % update the roi

                imshow(im)
                hold on, plot([roi_jet(1), roi_jet(2),...
                    roi_jet(2),roi_jet(1),...
                    roi_jet(1)],[roi_jet(3),...
                    roi_jet(3),roi_jet(4),...
                    roi_jet(4), roi_jet(3)], 'r--'), hold off

                button = questdlg('Jet region of interest
 ok?','','Yes','No','Yes');
            end
            dial=' -> Jet region of interest is set';
            set(handles.text1,'String',dial);
```

```
            set(handles.text2,'String',' ');
            set(handles.text3,'String',' ');
            set(handles.text4,'String',' ');

            param.roi_jet = roi_jet;

    %Select the jet reference line

             button = '';

            button = questdlg('Select jet reference
    line','','Ok','Ok');
            button= 'No';

            while ~strcmp(button,'Yes')
                imshow(im)
                [x_ref y_ref] = ginput(2);
                x_ref=[x_ref(1) x_ref(1)];
                hold on
                plot(x_ref,y_ref,'r'),hold off
                button = questdlg('Correct reference
    line?','','Yes','No','Yes');

            end
            param.ref_line=[x_ref(1) x_ref(2) y_ref(1) y_ref(2)];

            dial1=' -> Jet reference line is set';
            dial2=' -> Jet region of interest is set';
            dial3=' -> Bubble region of interest is set';
            set(handles.text1,'String',dial1);
            set(handles.text2,'String',dial2);
            set(handles.text3,'String',dial3);
            set(handles.text4,'String','');
```

--- Executes on button press in Resolution4.

```
    function Resolution4_Callback(hObject, eventdata, handles)

    %Change the resolution of the video [µm/px]

    global param ;
    answer = inputdlg({'Enter image resolution: [µm/px]'},...
                '',1,{num2str(param.resolution)});
            if str2num(answer{1}) > 0
                param.resolution = str2num(answer{1});
            else
                param.resolution = 1.56;    % default
                disp('wrong input')
            end
            texto1=strcat(' -> Resolution set to:
    ',num2str(param.resolution),' µm/px');
                set(handles.text1,'String',texto1);
                set(handles.text2,'String',' ');
```

```
                set(handles.text3,'String',' ');
                set(handles.text4,'String',' ');
```

--- Executes on button press in Framerate5.

```
function Framerate5_Callback(hObject, eventdata, handles)

%Change the framerate [frames/second]

global param;
answer = inputdlg({'Enter framerate: [fps]'},...
                '',1,{num2str(param.framerate)});
            if str2num(answer{1}) > 0
                param.framerate = str2num(answer{1});
            else
                param.framerate = 20000;    % default
                disp('wrong input')
            end
            texto1=strcat(' -> Framerate set
 to:',num2str(param.framerate), ' fps');
                set(handles.text1,'String',texto1);
                set(handles.text2,'String',' ');
                set(handles.text3,'String',' ');
                set(handles.text4,'String',' ');
```

--- Executes on button press in Analysis-Range6.

```
function AnalysisRange6_Callback(hObject, eventdata, handles)

%Change the frames which are analyzed. Take care with aliasing.

global sdir files param  info mov increment format_index filename;
if length(param.frnum) >= 2
                def_inc = param.frnum(2) - param.frnum(1);
            else
                def_inc = 1;
            end
            answer = inputdlg({'First frame [#]','Final frame
 [#]','Increment [#]'},...
                '',1,
{num2str(param.frnum(1)),num2str(param.frnum(end)),num2str(def_inc)});

            if format_index == 1 %.avi
               total_archivos = mov.NumberOfFrames;
           end

           if format_index == 2 %.tif compressed
               total_archivos = length(info);
           end

           if format_index == 3 %.tif descompressed
```

```
                total_archivos = length(files);
            end

            if format_index == 4 %.tif image
                total_archivos = 1;
            end

             if ~isempty(answer) %Validate the analysis range
                 if round(str2num(answer{1})) >= 1 &
round(str2num(answer{1})) <= total_archivos
                     frstart = round(str2num(answer{1}));
                 else
                     frstart = 1;
                 end

                 if round(str2num(answer{2})) >= frstart &
round(str2num(answer{2})) <= total_archivos
                     frstop = round(str2num(answer{2}));
                 else
                     frstop = total_archivos;
                 end

                 if round(str2num(answer{3})) > 0
                     increment = round(str2num(answer{3}));
                 else
                     increment = 1;
                 end
                 param.frnum = frstart:increment:frstop;     % update

                 texto1=' -> Frames selected';
                 texto2=strcat('     First frame:
',int2str(param.frnum(1)));
                 texto3=strcat('     Last frame:
',int2str(param.frnum(end)));
                 texto4=strcat('     Increment:
',int2str(increment));

             else
                 texto1=' -> No changes made in the frame selection';
                 texto2=strcat('     First frame:
',int2str(param.frnum(1)));
                 texto3=strcat('     Last frame:
',int2str(param.frnum(end)));
                 texto4=strcat('     Increment:
',int2str(def_inc));
             end
                 set(handles.text1,'String',texto1);
                 set(handles.text2,'String',texto2);
                 set(handles.text3,'String',texto3);
                 set(handles.text4,'String',texto4);
```

--- Executes on button press in ObjectSize7.

```
function ObjectSize7_Callback(hObject, eventdata, handles)

%Get a first aproximation of the bubble area to the following analysis

global sdir files param background results info mov format_index
 filename;

           if format_index == 1 %.avi
               im = read(mov,1);
           end

           if format_index == 2 %.tif compressed
               im = imread(strcat(sdir,filename),1);
           end

           if format_index == 3 %.tif descompressed
               im = imread(strcat(sdir,files(1).name));
           end

           if format_index == 4 %.tif image
               im = imread(strcat(sdir,filename));
           end

            im = imrotate(im,param.rotate,'bilinear','crop');
 % rotate image

            object_area = 20;    % initial bubble area
            button = '';
            while ~strcmp(button,'Yes')
                axes(handles.axes1);
                imshow(im)

                [x y] = ginput(2);
 % get two coordinates for the roi
                  x = round(sort(x));
                  y = round(sort(y));
                if x(1) < 1,          x(1) = 1;           end
   % validate the coorinates
                if x(2) > size(im,2), x(2) = size(im,2);  end
                if y(1) < 1,          y(1) = 1;           end
                if y(2) > size(im,1), y(2) = size(im,1);  end

                I = im(y(1):y(2),x(1):x(2));            % region to
 find a bubble in

                global idstart maxxpos idmax, idstart = 1; maxxpos =
 -1; idmax = 1;
                  area = (y(2)-y(1))*(x(2)-x(1));       % square area
                  ecc = 0.6;                            % minimum
 eccentricity [-]
```

```
                S = get_objects(I,[area/4, area],ecc);  % find a
droplet in the region

                if ~isempty(S.Area)
                    B = S.Boundary; %Boundary of the droplet
                    object_area = round(S.Area);

                    hold on, plot(B(:,2)+x(1)-1,B(:,1)+y(1)-1,...
                        'Color','r',...
                        'LineWidth',2), hold off

                    button = questdlg('Correct bubble
found?','','Yes','No','Yes');
                else
                    texto2='No bubble found!';
                    set(handles.text1,'String',texto2);
                    object_area = 20;
                    break
                end

            end
            param.object_area = object_area; %First aproximation of
the droplets area
            objAr= object_area*(param.resolution^2);
            texto1=strcat(' -> Bubble area is set to:
',num2str(objAr),' µm^2');
                set(handles.text1,'String',texto1);
                set(handles.text2,'String',' ');
                set(handles.text3,'String',' ');
                set(handles.text4,'String',' ');
```

--- Executes on button press in analyze8.

```
function analyse8_Callback(hObject, eventdata, handles)

%Analyze all frames

global sdir files param background results mov format_index filename;
            global idstart maxxpos idmax;   % global variables
            idstart = 1;                    % first object identity
            maxxpos = -1;                   % initial condition
            idmax   = 1;                    % last object identity

            % initialize the structure where the solution for every
frame
            % is going to be saved
            results = struct(...
                'width',{},... %image width
                'height',{},... %image height
                'framenumber',{},... %frame number
                'nrBubblesCaptured',{},... %number of bubbles captured
in each frame
                'bubbles',{},...
```

```
                'jet',{}); %identity of the bubbles which appear in
 each frame

            imbg = [];
            if ~isempty(background.imbg)
                imbg = background.imbg;
                imbg = imrotate(imbg,param.rotate,'bilinear','crop');
                imbg =
 imbg(param.roi(3):param.roi(4),param.roi(1):param.roi(2));
            end

            ecc = .5;   % eccentricity threshold
            object_area = param.object_area;
            area = round([object_area*0.5,object_area*2]);

            voortgang=waitbar(0,cat(2,'0%
 [0/',int2str(length(param.frnum)),']'),'NumberTitle','off','Name','Analyzing
 frames...'); %Inicializa la barra de espera

           %Read roijet and reference line
            roi_jet=param.roi_jet;
            x_ref= [param.ref_line(1) param.ref_line(2)];
            y_ref= [param.ref_line(3) param.ref_line(4)];
            x_refroi=[x_ref(1)-min([roi_jet(1) roi_jet(2)]), x_ref(1)-
min([roi_jet(1) roi_jet(2)])];%Reference line in the roi coordinates
 system

            jet_length=zeros(1,length(param.frnum));
            jet_diam=zeros(1,length(param.frnum));

            for i = 1:length(param.frnum) %for every frame

                framenumber = param.frnum(i);

           if format_index == 1 %.avi
               im = read(mov,framenumber);
           end

           if format_index == 2 %.tif compressed
               im = imread(strcat(sdir,filename),framenumber);
           end

           if format_index == 3 %.tif descompressed
               im = imread(strcat(sdir,files(framenumber).name));
           end

           if format_index == 4 %.tif image
               im = imread(strcat(sdir,filename));
           end



            image=im; %Image saved for jet analysis
            im = imrotate(im,param.rotate,'bilinear','crop');
```

```
            im =
im(param.roi(3):param.roi(4),param.roi(1):param.roi(2));
```

Bubble analysis

```
                [width height nrBubblesCaptured idrange data] =
captureBubbles(im,area,ecc,imbg);

                results(i).width = width;
                results(i).height = height;
                results(i).framenumber = framenumber;
                results(i).nrBubblesCaptured = nrBubblesCaptured;
                results(i).idrange = idrange;
                results(i).bubbles = data;
                results(i).totalNrBubbles = idmax;

                % -> update the object_area in case of growing
bubbles.
                if nrBubblesCaptured > 0
%                     object_area = results(i).bubbles(1).Area;
                end
```

Jet analysis

```
                image =
imrotate(image,param.rotate,'bilinear','crop');
                a_size=size(image);
                if length(a_size) > 2
                image=rgb2gray(image);
                end

                %-Process image
                image =
image(roi_jet(3):roi_jet(4),roi_jet(1):roi_jet(2));
                a_size=size(image);
                image = ~imbinarize(image,graythresh(image));
                image = bwareaopen(image,10);
                image(:,1)=1;%Fill with 1s the left border

                %-Complete the jet boundary in case there are some pxs
missing
                %Calculate mean line
                mean_line = [];
                mean_line_diam = [];
                [pxs_y,pxs_x] = size(image);
                object_diam_pxs = 2*sqrt(object_area/pi);
                    for i_xaxis=1:pxs_x
                        y_top_process =
max(find(image(:,i_xaxis)));%Top jet limit
                        y_bot_process =
min(find(image(:,i_xaxis)));%Bottom jet limit
                    if (~isempty(y_top_process)) && (y_top_process ~=
y_bot_process)
```

```
                        mean_line = [mean_line; (y_top_process
+y_bot_process)/2];
                        mean_line_diam = [mean_line_diam;
 y_top_process-y_bot_process];
                    end
                    end
                y_mean_line = round(mean(mean_line));
                mean_diam = round(mean(mean_line_diam));
                %Stop criteria for algorythm defined as first pxs
 along the
                %mean line
                mean_line_pxs = find(image(y_mean_line,3:end));
                pxs_stopprocess = min(mean_line_pxs);
                for i_xaxis=1:pxs_stopprocess
                    limites=[];
                 %Look for top and bottom limits
                    for j=2:pxs_y
                       if image(j,i_xaxis) == 0 && image(j-1,i_xaxis)
 == 1
                           limites = [limites; j-1 i_xaxis];
                       end
                    end
                  %If there is only one limit, the algorythm is
 applied
                   [n_limites,na] = size(limites);
                   if n_limites == 1
                       if limites(1,1) > y_mean_line
                           y_bot_process =
 min(find(image(:,i_xaxis-1)));%Bottom jet limit
                           image(y_bot_process,i_xaxis)=1;
                       end
                       if limites(1,1) <= y_mean_line
                           y_top_process =
 max(find(image(:,i_xaxis-1)));%Top jet limit
                           image(y_top_process,i_xaxis)=1;
                       end
                   end

                end

                image=imfill(image,'holes');%Fill the jet with 1s

                %-Remove lines with less width than 10% of the bubble
 size

                    for i_xaxis=pxs_stopprocess+1:pxs_x
                       y_bot_process =
 min(find(image(:,i_xaxis)));%Bottom jet limit
                       y_top_process =
 max(find(image(:,i_xaxis)));%Top jet limit
                       diam_process = y_top_process-y_bot_process;
                              if diam_process < 0.05*object_diam_pxs
                                   image(:,i_xaxis) = zeros(pxs_y,1);
                              end
```

```
                    end


                %-Calculate jet boundary
                B=bwtraceboundary(image,[double(a_size(1)-1)
 double(1)],'E',8,800,'clockwise');

                %-Calculate jet length & jet diameter
                jet_length(i)=max(B(:,2))-x_refroi(1);
                a=1;
                for j=x_refroi(1):jet_length(i)-1 %Look for top limits
 from reference line to the end of the jet
                     y_top = max(find(image(:,j)));%Top jet limit
                     y_bot = min(find(image(:,j)));%Bottom jet limit
                     diam(a) = y_top-y_bot;%Diameter for every
 vertical line
                     a=a+1;
                end
                jet_diam(i)=mode(diam);

                %Apply breakpup criteria to identify jet id
                if i==1
                    jetId = 1;
                elseif jet_length(i-1)-jet_length(i) >
 sqrt(param.object_area/pi) %CRITERIO ROTURA
                    jetId = jetId+1;
                end

                %Save jet data in results
                jetData = struct('jetBoundary',{},'JetId',
{},'jetLength',{},'jetDiam',{});
                jetData = struct('jetBoundary',{B},'JetId',
{jetId},'jetLength',{jet_length(i)},'jetDiam',{jet_diam(i)});

                results(i).jet=jetData;

                waitbar(i/
length(param.frnum),voortgang,strcat(int2str(round(100*i/
length(param.frnum))),'%
 [',int2str(i),'/',int2str(length(param.frnum)),']'));

            end
            close(voortgang); %Una vez terminado el bucle la cierra
```

--- Executes on button press in polidispersity9.

```
function polidispersity9_Callback(hObject, eventdata, handles)

%removed
```

--- Executes on button press in diameter10.

```
function diameter10_Callback(hObject, eventdata, handles)
```

```
global results PDI stdr param;
      totalNrBubbles = results(end).totalNrBubbles;
%             totalNrBubbles = results(end).idrange(1);    % max id
            y = bubbleInfo(results,'Radius',1:totalNrBubbles);
            for i = 1:totalNrBubbles
                r(i) = mean(y{i});  % mean radius for each bubble in
 all frames (r(1) = mean bubble 1 radius)
            end

            if totalNrBubbles == 0
                PDI = -1;
                texto1=' -> No bubbles detected';
                texto2=' ';
                texto3=' ';
                texto4=' ';
            elseif totalNrBubbles == 1
                PDI = 0;
                texto1=' -> Only 1 bubble detected. The polydispersity
 index is 0 %';
                texto2=' ';
                texto3=' ';
                texto4=' ';
            else

           %-- Area histogram
                y = bubbleInfo(results,'Area',1:totalNrBubbles);
            for i = 1:totalNrBubbles
                a(i) = mean(y{i});  %mean bubble area for every
 captures bubble
            end
            a = param.resolution^2 * a; % area in µm2;

            x = linspace(0.9*mean(a), 1.1*mean(a), 20);
            axes(handles.axes1);
            hist(a,x), xlabel('bubble area [µm^2]'),ylabel('n°
 bubbles')

           %-- Radius histogram
                y = bubbleInfo(results,'Radius',1:totalNrBubbles);
            for i = 1:totalNrBubbles
                a(i) = mean(y{i});
            end
            a = param.resolution * a;

            x = linspace(0.98*mean(a), 1.02*mean(a), 20);
            axes(handles.axes2);
            hist(a,x), xlabel('bubble radius [µm]'),ylabel('n°
 bubbles')


            %-- Write text
                PDI = 100 * std(r) / mean(r);
                PDI=num2str(PDI);
```

```
                r=r.*param.resolution;
                meanr=num2str(mean(r));

                stdr=num2str(std(r));
                totalBubblesN=int2str(totalNrBubbles);

                texto1=strcat('Mean bubble radius: ',meanr,' µm');
                texto2=strcat('PDI: ',PDI,' %');
                texto3=strcat('SD: ',stdr,' µm' );
                texto4=strcat('Captured bubbles: ',totalBubblesN);



            end

                set(handles.text1,'String',texto1);
                set(handles.text2,'String',texto2);
                set(handles.text3,'String',texto3);
                set(handles.text4,'String',texto4);
```

--- Executes on button press in frecuency11.

```
function frecuency11_Callback(hObject, eventdata, handles)

global param results bubbleFrequency;
            totalNrBubbles = results(end).totalNrBubbles;
            frameBubbleCreation = [];
            createdBubble = [];
            for id = 1:totalNrBubbles
                f = 1;
                %%%Look for the first frame where the bubble id appear

                while isempty(find(results(f).idrange == id))%Find
 will be empty until the bubble id appear
                    f = f+1;
                end
                if f > 1 %The bubbles which already appear in the
 first frame are not taken into acount
                    frameBubbleCreation = [frameBubbleCreation,
 results(f).framenumber];%Frames where a bubble is created
                    createdBubble = [createdBubble, id];%Save the
 created bubbles id numbers
                end
            end

            creationTime = zeros(size(frameBubbleCreation));
            for i = 1:length(creationTime)
                creationTime(i) = frameBubbleCreation(i) /
 param.framerate;%Instante de tiempo en el que se crea una burbuja
            end
```

```
            timeError = ( param.frnum(2) - param.frnum(1) ) /
 param.framerate;
            errBar = ones(size(creationTime)) * timeError / 2;

            h = findobj('Name','Bubble frequency');

            set(h,'NumberTitle','on','Name','Bubble frequency')
            set(gca,'FontName','Times','Box','on')

            axes(handles.axes1);

            plot(createdBubble,creationTime*1000,'o')
            xlabel('created bubbles [#]')
            ylabel('time [ms]')

            if length(createdBubble) > 1
                P = polyfit(createdBubble,creationTime,1);  %Create a
 sraight minor adjustment line
                bubbleFrequency = 1 / P(1);

                hold on, plot(createdBubble,(P(1)*createdBubble +
 P(2))*1000,'k--','LineWidth',2), hold off
                texto1=' -> Bubble frequency:';
                bubbleFrequency=num2str(bubbleFrequency);
                texto2=' Hz';
                textof=strcat(texto1,bubbleFrequency,texto2);
                set(handles.text1,'String',textof);
                set(handles.text2,'String',' ');
                set(handles.text3,'String',' ');
                set(handles.text4,'String',' ');

            else
%                fprintf(' -> The bubble frequency could not be
 determined.\n')
            end

            axes(handles.axes2);
            im=imread('LogoEtsi.png');
            imshow(im);
```

--- Executes on button press in velocity12.

```
function velocity12_Callback(hObject, eventdata, handles)

global param results MeanDropletSpeed Qi Qe;
%----- 1º CREATION TIME (Just like when the frequency is calculated,
 but variables are local)

            if ~exist('creationTime','var')

            totalNrBubbles = results(end).totalNrBubbles;
            frameBubbleCreation = [];
            createdBubble = [];
```

```
            for id = 1:totalNrBubbles
                f = 1;
                while isempty(find(results(f).idrange == id))
                    f = f+1;
                end
                if f > 1
                    frameBubbleCreation = [frameBubbleCreation,
results(f).framenumber];
                    createdBubble = [createdBubble, id];
                end
            end

            creationTime = zeros(size(frameBubbleCreation));
            for i = 1:length(creationTime)
                creationTime(i) = frameBubbleCreation(i) /
param.framerate;
            end
            end %END CREATION TIME

            %----- 2º DISAPPEAR TIME
            frameDisappear= [];
            a_2=1;
            increment=param.frnum(2)-param.frnum(1);
            for id= createdBubble(1):createdBubble(end) %For every
created bubble

                fc_2= frameBubbleCreation(a_2);%Start from the frame
where id is created
                index_cf=(fc_2+(increment-1))/(increment);%Index
corresponding with the frame fc_2

                for i_10=index_cf:length(results) %Look for the frame
where the bubble id disappear


                    if ~isempty(find(results(i_10).idrange == id))
                        f_2=i_10;%Last frame where id appear
                    end

                end
                 a_2= a_2+1;
                frameDisappear=
[frameDisappear,results(f_2).framenumber];%Save the disappear frames

            end


            DisappearTime=zeros(size(frameDisappear));
            for i_2=1:length(frameDisappear)%Calcular tiempos de
desaparicion
                DisappearTime(i_2)= frameDisappear(i_2)/
param.framerate;
            end
```

```
            %----3º CALCULATE VELOCITY


           for i_3= 1:length(createdBubble) %Calcular velocidades
                    L_3= param.resolution*(param.roi(2)-
param.roi(1)); %length travel

                    if
 frameDisappear(i_3)<results(end).framenumber %To avoid the bubbles
 which are still in the video when the analysis finishes
                    speedDroplet(i_3)= L_3/(DisappearTime(i_3)-
creationTime(i_3));
                    end

           end
           MeanDropletSpeed= mean(speedDroplet);

           %--- 4º CALCULATE FLOW RATE
           P = polyfit(createdBubble,creationTime,1);
           bubbleFrequency = 1 / P(1);

           im_help=imread('Medidas.png');
           axes(handles.axes1);
           imshow(im_help)

           axes(handles.axes2);
            im_2=imread('LogoEtsi.png');
            imshow(im_2);

           answer = inputdlg({'W rectangular section [µm]','H
 rectangular section [µm]','R  ciruclar section [µm]'},...
                '',1,{'0','0','0'});

           W= str2num(answer{1});
           H= str2num(answer{2});
           R= str2num(answer{3});

           y = bubbleInfo(results,'Radius',1:totalNrBubbles);
            for i = 1:totalNrBubbles
                r(i) = mean(y{i});  % mean bubble radius for every
 captured bubble
            end

           Rg=mean(r)*param.resolution;

           if W == 0 %Circular section
               Qi= (4/3)*pi*(Rg^3)*bubbleFrequency;
               Qe= pi*(R^2)*MeanDropletSpeed;
           end

           if R == 0 %Rectangular section
               Qi=(4/3)*pi*(Rg^3)*bubbleFrequency;
               Qe= W*H*MeanDropletSpeed;
           end
```

```
Qe=Qe*(3600*10^(-9));% microliters per hour
Qi=Qi*(3600*10^(-9));

texto1=' -> Mean bubble velocity: ';
MeanDropletSpeed=num2str(MeanDropletSpeed);
texto2=' µm/s';
textof=strcat(texto1,MeanDropletSpeed,texto2);

texto11=' -> Qi: ';
Qi=num2str(Qi);
texto21=' µl/h';
textof1=strcat(texto11,Qi,texto21);

texto12=' -> Qe: ';
Qe=num2str(Qe);
texto22=' µl/h';
textof2=strcat(texto12,Qe,texto22);

     set(handles.text1,'String',textof);
     set(handles.text2,'String',textof1);
     set(handles.text3,'String',textof2);
     set(handles.text4,'String',' ');
```

--- Executes on button press in jetlength13.

```
function jetlength13_Callback(hObject, eventdata, handles)

global sdir files param background results info mov format_index
 filename mean_jetlength mean_jetdiam std_length PDI_length std_diam
 PDI_diam jet;

%Load jet data from results
jet_length=zeros(1,length(results));
jet_diam=zeros(1,length(results));
for i=1:length(results)
jet_length(i)=results(i).jet.jetLength;
jet_diam(i)=results(i).jet.jetDiam;
end
       %-- PDI and STD jet length and diameter
       std_length=std(jet_length);
       PDI_length= (100*std_length)/mean(jet_length);
       std_length=num2str(std_length);
       PDI_length=num2str(PDI_length);

       std_diam=std(jet_diam);
       PDI_diam=(100*std_diam)/mean(jet_diam);
       std_diam=num2str(std_diam);
       PDI_diam=num2str(PDI_diam);

       mean_jetdiam= (sum(jet_diam)/
length(jet_diam))*param.resolution;
       mean_jetdiam=num2str(mean_jetdiam);
```

```
        mean_jetlength=(sum(jet_length)/
length(jet_length))*param.resolution;
        mean_jetlength=num2str(mean_jetlength);


        %-- Show results
        texto1=strcat(' -> Mean jet length: ',' ');
        texto2=' µm // ';
        texto2=strcat(mean_jetlength,texto2);


        texto11=strcat(' -> Mean jet diameter: ',' ');
        texto21=' µm // ';
        texto21=strcat(mean_jetdiam,texto21);


        texto31= ' SD: ';
        texto32=' µm // PDI: ';
        texto3f=strcat(texto31,std_length,texto32,PDI_length,' %');
        texto1=strcat(texto1,texto2,texto3f);

        texto41= ' SD: ';
        texto42=' µm // PDI: ';
        texto4f=strcat(texto41,std_diam,texto42,PDI_diam,' %');
        texto11=strcat(texto11,texto21,texto4f);

        set(handles.text1,'String',texto1);
        set(handles.text2,'String',texto11);
        set(handles.text3,'String',' ');
        set(handles.text4,'String',' ');
```

--- Executes on button press in movie14.

```
    function movie14_Callback(hObject, eventdata, handles)

    global sdir files param background results info mov format_index
     filename jet;

                axes(handles.axes2);
                im=imread('LogoEtsi.png');
                imshow(im);
                totalNrBubbles = results(end).totalNrBubbles;

                answer = inputdlg({['Select bubble identities
     [1:',int2str(totalNrBubbles),']']},...
                    '',1,{'all'});
                if strncmpi(answer{1},'all',1)
                    ids = 1:totalNrBubbles;
                elseif ~isempty(answer)
                    ids = round(str2num(answer{1}));
                else
                    ids = 1:totalNrBubbles;
                end
```

```
            % Plot the figures.
            %-------------------------
            axes(handles.axes1);
            cmap = colormap('Lines');
            for i = 1:size(results,2) % for every analyzed frame
                framenumber = results(i).framenumber;

           if format_index == 1 %.avi
               im = read(mov,framenumber);
           end

           if format_index == 2 %.tif compressed
               im = imread(strcat(sdir,filename),framenumber);
           end

           if format_index == 3 %.tif descompressed
               im = imread(strcat(sdir,files(framenumber).name));
           end

           if format_index == 4 %.tif image
               im = imread(strcat(sdir,filename));
           end

                im = imrotate(im,param.rotate,'bilinear','crop');
  % rotate image
                imshow(im)

                if results(i).nrBubblesCaptured > 0

                    for b = 1:length(ids)
                        id = ids(b);

                        idx = find(results(i).idrange == id);
                        if ~isempty(idx)
                            B = results(i).bubbles(idx).Boundary;
                            C = results(i).bubbles(idx).Centroid;
                            R = results(i).bubbles(idx).Radius;


                            hold on

plot(B(:,2)+double(param.roi(1)-1),B(:,1)+double(param.roi(3)-1),'-','Color',cmap

text(C(1)+double(param.roi(1)-1)-4,C(2)+double(param.roi(3)-1)+2*R,int2str(id),..
                                'Color','w',...
                                'FontSize',10,...
                                'FontWeight','bold');
                            hold off
                        end

                    end

                   if ~isempty(results(i).jet)
```

```
                    hold on
                    B_jet=results(i).jet.jetBoundary;
                    index=find(max(B_jet(:,1)));
                    IdJet = results(i).jet.JetId;

 plot(B_jet(:,2)+double(param.roi_jet(1)-1),B_jet(:,1)+double(param.roi_jet(3)-1),

 text(B_jet(index,2)+double(param.roi_jet(1)-1),B_jet(index,1)+double(param.roi_je
                    hold off
                    end
                     pause(1/10)
                end
                drawnow
                pause(0.2)
            end
```

--- Executes on button press in save15.

```
function save15_Callback(hObject, eventdata, handles)
global sdir param results increment PDI stdr MeanDropletSpeed
 bubbleFrequency mean_jetlength Qi Qe filename mean_jetdiam std_length
 std_diam PDI_length PDI_diam bubb selected_bubble tau Breakups
 Recorded_frames;
fclose('all');
delete *.txt ;%Remove previous txt files
%--Save parameters and results
% save([sdir,'results.mat'],'results');%Save in matlab file
% save([sdir,'parameters.mat'],'param');


%%%%%FICHERO 1 - OUTPUT FILE%%%%%

title=strcat(filename,'OutputFile.txt');
parametros=fopen(title,'wt');%Save in text file

fprintf(parametros,'--> INPUT PARAMETERS\n');
fprintf(parametros,'Resolution: %d µm/px',param.resolution);
fprintf(parametros,'\nFramerate: %d fps\n',param.framerate);
fprintf(parametros,['First frame:   ',int2str(param.frnum(1)),'\n']);
fprintf(parametros,['Last frame:
 ',int2str(param.frnum(end)),'\n']);
fprintf(parametros,['Increment:     ',int2str(increment),'\n']);

fprintf(parametros,'\n--> RESULTS\n');
if ~isempty(results)
fprintf(parametros,'Nº of bubbles captured: %d
 \n',results(end).totalNrBubbles);
fprintf(parametros,'Nº of breakups captured: %d
 \n',results(end).jet.JetId);

r=[];
y = bubbleInfo(results,'Radius',1:results(end).totalNrBubbles);
            for i = 1:results(end).totalNrBubbles
```

```
                r(i) = mean(y{i});  % mean bubble radius for every
 captured bubble
            end
            r_m=mean(r);
fprintf(parametros,'Mean bubble radius: %d µm\n',r_m);
if ~isempty(PDI)
PDI=strcat(PDI,' %');
fprintf(parametros,'Bubble radius PDI: %s  \n',PDI);
fprintf(parametros,'Bubble radius SD: %s µm \n',stdr);
else
fprintf(parametros,'\n');
fprintf(parametros,'\n');
end

if ~isempty(bubbleFrequency)
fprintf(parametros,'Mean generation frequency: %s Hz
\n',bubbleFrequency);
else
fprintf(parametros,'\n');
end

if ~isempty(mean_jetlength)
PDI_length=strcat(PDI_length,' %');
PDI_diam=strcat(PDI_diam,' %');
fprintf(parametros,'Mean jet length: %s µm\n',mean_jetlength);
fprintf(parametros,'Jet length PDI: %s  \n',PDI_length);
fprintf(parametros,'Jet length SD: %s µm \n',std_length);
fprintf(parametros,'Mean jet diameter: %s µm\n',mean_jetdiam);
fprintf(parametros,'Jet diameter PDI: %s  \n',PDI_diam);
fprintf(parametros,'Jet diameter SD: %s µm \n',std_diam);
else
fprintf(parametros,'\n');
fprintf(parametros,'\n');
fprintf(parametros,'\n');
fprintf(parametros,'\n');
fprintf(parametros,'\n');
fprintf(parametros,'\n');
end

if ~isempty(MeanDropletSpeed)
fprintf(parametros,'Mean bubble velocity: %s µm/s
\n',MeanDropletSpeed);
else
fprintf(parametros,'\n');
end

if ~isempty(Qi)
fprintf(parametros,'Qi: %s µl/h\n',Qi);
fprintf(parametros,'Qo: %s µl/h\n',Qe);
else
fprintf(parametros,'\n');
fprintf(parametros,'\n');
end
```

```
end
fclose(parametros);
SOURCE=title;
DESTINATION=strcat(sdir,title);
copyfile(SOURCE,DESTINATION);


%%%%%FICHERO 2 - BUBBLE RADIUS FILE%%%%%

%-- Save all droplet mean radius
title2 = strcat(filename,'Radius.txt');
radios=fopen(title2,'wt');
for j=1:length(r)
    rg=num2str(r(j));
    fprintf(radios,'%s\n',rg);
end
fclose(radios);
SOURCE=title2;
DESTINATION=strcat(sdir,title2);
copyfile(SOURCE,DESTINATION);


%%%%%FICHEROS COORDENADAS CHORROS SELECCIONADOS%%%%%

if ~isempty(Breakups)
    jj=1;

        JetIdVector = zeros(1,length(results));%Vector with JetIds
        for re=1:length(results)
            JetIdVector(re)=results(re).jet.JetId;
        end


    for ii=1:length(Breakups)

        index=num2str(Breakups(ii),'%02d');
        title=strcat(filename,'b',index,'tau','2.00','.txt');
        fileID=fopen(title,'wt');

        BreakupIndex = find(JetIdVector == Breakups(ii),1,'last');
        A=results(BreakupIndex).jet.jetBoundary;
        A=A.*param.resolution;

        save(title,'A','-ascii');

        jj=jj+1;

        fclose(fileID);
        SOURCE=title;
        DESTINATION=strcat(sdir,title);
        copyfile(SOURCE,DESTINATION);

    end
end
```

```
jj=1;
if ~isempty(tau)
    for ii=1:length(tau)

        bubble=num2str(selected_bubble,'%02d');

 title2=strcat(filename,'b',bubble,'tau',num2str(tau(ii),'%10.2f'),'.txt');
        fileID=fopen(title2,'wt');

        A =
 (results(Recorded_frames(ii)).jet.jetBoundary).*param.resolution;

        save(title2,'A','-ascii');

        jj=jj+1;

        fclose(fileID);
        SOURCE=title2;
        DESTINATION=strcat(sdir,title2);
        copyfile(SOURCE,DESTINATION);
    end
end

% --- Executes during object creation, after setting all properties.
function analyse8_CreateFcn(hObject, eventdata, handles)
% hObject    handle to analyse8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
 called


% --- Executes during object creation, after setting all properties.
function polidispersity9_CreateFcn(hObject, eventdata, handles)
% hObject    handle to polidispersity9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
 called


% --- Executes during object deletion, before destroying properties.
function analyse8_DeleteFcn(hObject, eventdata, handles)
% hObject    handle to analyse8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --------------------------------------------------------------------
function Untitled_1_Callback(hObject, eventdata, handles)

% hObject    handle to Untitled_1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

--- Executes on button press in pushbutton16.

```
function pushbutton16_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton16 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
fclose('all');
delete *.txt ;
close(handles.figure1);


% --- Executes on button press in clear.
function clear_Callback(hObject, eventdata, handles)

% hObject    handle to clear (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
clear global sdir param results results_profile increment PDI stdr MeanDropletSpee
cla(handles.axes1)
dial1 = 'To start a new analysis, push button "1"';
set(handles.text1,'String',dial1);

%PROFILE
```

--- Executes on button press in Bub-bleRange15.

```
function BubbleRange15_Callback(hObject, eventdata, handles)

global sdir files param background results results_profile info mov
 format_index filename jet bubb Breakups


TotalBreakups = results(end).jet.JetId-1;

answer=inputdlg({'First breakup [#]','Last breakup [#]','Increment
 [#]'},...
    '',1,{num2str(results(1).jet.JetId),num2str(TotalBreakups),'1'});


FirstBubble=str2num(answer{1});
LastBubble=str2num(answer{2});
Increment=str2num(answer{3});
Breakups=FirstBubble:Increment:LastBubble;
```

--- Executes on button press in TimeRange16.

```
function TimeRange16_Callback(hObject, eventdata, handles)
% hObject    handle to TimeRange16 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
global sdir files param background results results_profile info mov
 format_index filename jet selected_bubble tau Recorded_frames;



TotalBreakups = results(end).jet.JetId-1;
texto1=strcat(' -> Number of breakups: ',num2str(TotalBreakups));

set(handles.text1,'String',texto1);
set(handles.text2,'String',' ');
set(handles.text3,'String',' ');
set(handles.text4,'String',' ');


answer_1=inputdlg({'Select breakup [#]'},...
    '',1,{num2str(TotalBreakups)});

selected_bubble = str2double(answer_1(1));

JetIdVector = zeros(1,length(results));%Vector with JetIds
    for re=1:length(results)
            JetIdVector(re)=results(re).jet.JetId;
    end

Frame_inicio = find(JetIdVector == selected_bubble,1);
Frame_final = find(JetIdVector == selected_bubble,1,'last');

texto1=strcat(' -> Breakup frames for selected bubble: ');
texto2=strcat(' -> From: ',num2str(Frame_inicio),' To:
 ',num2str(Frame_final));

set(handles.text1,'String',texto1);
set(handles.text2,'String',texto2);
set(handles.text3,'String',' ');
set(handles.text4,'String',' ');

answer=inputdlg({'Select time divisions'},...
    '',1,{num2str(1)});

answer=str2num(answer{1});
TimeDivisions = answer;

Recorded_frames = zeros(1,TimeDivisions+1);
Increment = round((Frame_final - Frame_inicio)/(TimeDivisions),0);
tau = [];
for i=1:TimeDivisions+1
    Recorded_frames(i) = Frame_inicio + Increment*(i-1);
    tau(i)=(1/TimeDivisions)*(i-1);
end

Recorded_frames(end) = Frame_final;
```

```
ndecimals=2;
tau=round(tau,ndecimals);

texto1=strcat(' -> Dimensionless times: ');
texto2=strcat(' -> ',num2str(tau));

set(handles.text1,'String',texto1);
set(handles.text2,'String',texto2);
set(handles.text3,'String',' ');
set(handles.text4,'String',' ');




function BubbleRange15_CreateFcn(varargin)

function AnalysisRange6_CreateFcn(varargin)
```

Published with MATLAB® R2018a

Table of Contents

..... 1
--- Executes on button press in DeleteFigure3: Clear Figure 1 3
--- Executes on button press in PlotFigure1: Plot Figure 1 3
--- Executes on button press in SaveFigure2: Save Figure 1 5
--- Executes on button press in pushbutton11: Plot figure 3 5
--- Executes on button press in pushbutton10: Save figure 3 8
--- Executes on button press in pushbutton9: Clear figure 3 8
--- Executes on button press in pushbutton8: Clear figure 2 8
--- Executes on button press in pushbutton7: Save figure 2 8
--- Executes on button press in pushbutton6: Plot figure 2 9
--- Executes on button press in btnRight3. 11


```
function varargout = miplot(varargin)
% MIPLOT MATLAB code for miplot.fig
%      MIPLOT, by itself, creates a new MIPLOT or raises the existing
%      singleton*.
%
%      H = MIPLOT returns the handle to a new MIPLOT or the handle to
%      the existing singleton*.
%
%      MIPLOT('CALLBACK',hObject,eventData,handles,...) calls the
 local
%      function named CALLBACK in MIPLOT.M with the given input
 arguments.
%
%      MIPLOT('Property','Value',...) creates a new MIPLOT or raises
 the
%      existing singleton*.  Starting from the left, property value
 pairs are
%      applied to the GUI before miplot_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
 application
%      stop.  All inputs are passed to miplot_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows
 only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help miplot

% Last Modified by GUIDE v2.5 23-Jun-2019 12:31:30

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
```

```
                   'gui_OpeningFcn', @miplot_OpeningFcn, ...
                   'gui_OutputFcn',  @miplot_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before miplot is made visible.
function miplot_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to miplot (see VARARGIN)

%Put the menu in the center of the screen
scrsz=get(0,'ScreenSize');
pos_act=get(gcf,'Position');
xr=scrsz(3)-pos_act(3);
xp=round(xr/2);
yr=scrsz(4)-pos_act(4);
yp=round(yr/2);
set(gcf,'Position',[xp yp pos_act(3) pos_act(4)]);

%Choose default command line output for miplot
handles.output = hObject;

%Update handles structure
guidata(hObject, handles);

%UIWAIT makes miplot wait for user response (see UIRESUME)
%uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = miplot_OutputFcn(hObject, eventdata, handles)

% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;
```

--- Executes on button press in DeleteFigure3: Clear Figure 1

```
function DeleteFigure3_Callback(hObject, eventdata, handles)

% hObject    handle to DeleteFigure3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
cla(handles.axes1)
reset(handles.axes1)
```

--- Executes on button press in PlotFigure1: Plot Figure 1

```
function PlotFigure1_Callback(hObject, eventdata, handles)

% hObject    handle to PlotFigure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
axes(handles.axes1);

%Read files to be represented
[filename1,sdir1,~]=uigetfile('*.txt','Select the txt
 file','MultiSelect','on');

%Cell array of colors
C={'b','r','g','k','y','m','c','b','r','g','k','y','m','c','b','r','g','k','y','m'

if ~isempty(filename1)

%Read first jet
if iscell(filename1)
    filedirectory = strcat(sdir1,filename1{1});
else
    filedirectory = strcat(sdir1,filename1);
end

%Load jet boundary and set reference
B=load(filedirectory);
max_ref = max(B(:,2));
reference = get(handles.btnAlignLeft1,'Value');

%Number of jets to be represented
if iscell(filename1)
    n_iter = length(filename1);
else
    n_iter = 1;
end

%Loop to represent all jets acording to representation options
for i=1:n_iter
```

```
    hold on

    %Read jet boundary file
    if iscell(filename1)
      filedirectory = strcat(sdir1,filename1{i});
    else
      filedirectory = strcat(sdir1,filename1);
    end

    %Load jet boundary and save max x value
    B=load(filedirectory);
    max_actual(i) = max(B(:,2));

    %Coordinates change to set middle of jet as 0
    index_ChangeCoordinates = find(B(:,2) == max_actual(i));
    ChangeCoordinates_aux = [];
    for j=1:length(index_ChangeCoordinates)
        ChangeCoordinates_aux(j) = B(index_ChangeCoordinates(j),1);
    end
    ChangeCoordinates(i) = mean(ChangeCoordinates_aux);
    MeanChangeCoordinates = mean(ChangeCoordinates);

    %Coordinates change to move jets to right or left side
    if reference == 1
        columna_x = B(:,2);
    else
        columna_x = B(:,2)+(max_ref-max_actual(i));
    end
    columna_y = B(:,1)-MeanChangeCoordinates;
    max_actual_y(i) = max(columna_y);

    %Plot current jet
    plot(columna_x,columna_y,C{i},'lineWidth',2);

    %Generate legend with filename
    if iscell(filename1)
      filename_local = filename1{i};
      filename_legend{i} = filename_local(1:end-4);
    else
      filename_local = filename1;
      filename_legend{i} = filename_local(1:end-4);
    end


end

%Set figure properties

    if iscell(filename1)
    max_string_x = num2str(round(max(max_actual)));
    num_cifras_x = length(max_string_x);
    x_lim = (round(max(max_actual)/
(10^(num_cifras_x)),1)+0.1)*(10^num_cifras_x);
```

```
    xlim ([0 x_lim])
    paso_x = x_lim/20;
    xticks(0:paso_x:x_lim)

    max_string_y = num2str(round(max(max_actual_y)));
    num_cifras_y = length(max_string_y);
    y_lim = round(max(max_actual_y)/
(10^(num_cifras_y)),1)*(10^num_cifras_y);
    ylim ([-y_lim y_lim])
    paso_y = (2*y_lim)/10;
    yticks(-y_lim:paso_y:y_lim)
    end

    grid on
    grid minor
    axis equal
    xlabel('µm')
    ylabel('µm')
    lgd=legend(filename_legend,'Location','NorthEastOutside');
    set(lgd,'FontSize',12);

end
```

--- Executes on button press in SaveFigure2: Save Figure 1

```
function SaveFigure2_Callback(hObject, eventdata, handles)

% hObject    handle to SaveFigure2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
[nomb,ruta] = uiputfile('*.fig','GUARDAR IMAGEN');
fignew = figure('Visible','off'); % Invisible figure
newAxes = copyobj(handles.axes1,fignew);
set(newAxes,'Position',get(groot,'DefaultAxesPosition')); % The
 original position is copied too, so adjust it.
set(fignew,'CreateFcn','set(gcbf,''Visible'',''on'')'); % Make it
 visible upon loading
directory = strcat(ruta,nomb);
savefig(fignew,directory);
```

--- Executes on button press in pushbutton11: Plot figure 3

```
function pushbutton11_Callback(hObject, eventdata, handles)

axes(handles.axes3);

%Read files to be represented
[filename3,sdir3,~]=uigetfile('*.txt','Select the txt
 file','MultiSelect','on');
```

```
%Cell array of colors
C={'b','r','g','k','y','m','c','b','r','g','k','y','m','c','b','r','g','k','y','m'

if ~isempty(filename3)

%Read first jet
if iscell(filename3)
    filedirectory = strcat(sdir3,filename3{1});
else
    filedirectory = strcat(sdir3,filename3);
end

%Load jet boundary and set reference
B=load(filedirectory);
max_ref = max(B(:,2));
reference = get(handles.btnAlignLeft3,'Value');

%Number of jets to be represented
if iscell(filename3)
    n_iter = length(filename3);
else
    n_iter = 1;
end

%Loop to represent all jets acording to representation options
for i=1:n_iter

    hold on

    %Read jet boundary file
    if iscell(filename3)
      filedirectory = strcat(sdir3,filename3{i});
    else
      filedirectory = strcat(sdir3,filename3);
    end

    %Load jet boundary and save max x value
    B=load(filedirectory);
    max_actual(i) = max(B(:,2));

    %Coordinates change to set middle of jet as 0
    index_ChangeCoordinates = find(B(:,2) == max_actual(i));
    ChangeCoordinates_aux = [];
    for j=1:length(index_ChangeCoordinates)
        ChangeCoordinates_aux(j) = B(index_ChangeCoordinates(j),1);
    end
    ChangeCoordinates(i) = mean(ChangeCoordinates_aux);
    MeanChangeCoordinates = mean(ChangeCoordinates);

    %Coordinates change to put move jets to right or left side
    if reference == 1
        columna_x = B(:,2);
    else
```

```
        columna_x = B(:,2)+(max_ref-max_actual(i));
    end
    columna_y = B(:,1)-MeanChangeCoordinates;
    max_actual_y(i) = max(columna_y);

    %Plot current jet
    plot(columna_x,columna_y,C{i},'lineWidth',2);

    %Generate legend with filename
    if iscell(filename3)
      filename_local = filename3{i};
      filename_legend{i} = filename_local(1:end-4);
    else
      filename_local = filename3;
      filename_legend{i} = filename_local(1:end-4);
    end


end

%Set figure properties

    if iscell(filename3)
    max_string_x = num2str(round(max(max_actual)));
    num_cifras_x = length(max_string_x);
    x_lim = (round(max(max_actual)/
(10^(num_cifras_x)),1)+0.1)*(10^num_cifras_x);
    xlim ([0 x_lim])
    paso_x = x_lim/20;
    xticks(0:paso_x:x_lim)

    max_string_y = num2str(round(max(max_actual_y)));
    num_cifras_y = length(max_string_y);
    y_lim = round(max(max_actual_y)/
(10^(num_cifras_y)),1)*(10^num_cifras_y);
    ylim ([-y_lim y_lim])
    paso_y = (2*y_lim)/10;
    yticks(-y_lim:paso_y:y_lim)
    end

    grid on
    grid minor
    axis equal
    xlabel('µm')
    ylabel('µm')
    lgd=legend(filename_legend,'Location','NorthEastOutside');
    set(lgd,'FontSize',12);

end
```

--- Executes on button press in pushbutton10: Save figure 3

```
function pushbutton10_Callback(hObject, eventdata, handles)

% hObject    handle to pushbutton10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
[nomb,ruta] = uiputfile('*.fig','GUARDAR IMAGEN');
fignew = figure('Visible','off'); % Invisible figure
newAxes = copyobj(handles.axes3,fignew);
set(newAxes,'Position',get(groot,'DefaultAxesPosition')); % The
 original position is copied too, so adjust it.
set(fignew,'CreateFcn','set(gcbf,''Visible'',''on'')'); % Make it
 visible upon loading
directory = strcat(ruta,nomb);
savefig(fignew,directory);
```

--- Executes on button press in pushbutton9: Clear figure 3

```
function pushbutton9_Callback(hObject, eventdata, handles)

% hObject    handle to pushbutton9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
cla(handles.axes3)
reset(handles.axes3)
```

--- Executes on button press in pushbutton8: Clear figure 2

```
function pushbutton8_Callback(hObject, eventdata, handles)

% hObject    handle to pushbutton8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
cla(handles.axes2)
reset(handles.axes2)
```

--- Executes on button press in pushbutton7: Save figure 2

```
function pushbutton7_Callback(hObject, eventdata, handles)

% hObject    handle to pushbutton7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)
[nomb,ruta] = uiputfile('*.fig','GUARDAR IMAGEN');
fignew = figure('Visible','off'); % Invisible figure
newAxes = copyobj(handles.axes2,fignew);
set(newAxes,'Position',get(groot,'DefaultAxesPosition')); % The
 original position is copied too, so adjust it.
set(fignew,'CreateFcn','set(gcbf,''Visible'',''on'')'); % Make it
 visible upon loading
directory = strcat(ruta,nomb);
savefig(fignew,directory);
```

--- Executes on button press in pushbutton6: Plot figure 2

```
function pushbutton6_Callback(hObject, eventdata, handles)

% hObject    handle to pushbutton6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
axes(handles.axes2);

%Read files to be represented
[filename2,sdir2,~]=uigetfile('*.txt','Select the txt
 file','MultiSelect','on');

%Cell array of colors
C={'b','r','g','k','y','m','c','b','r','g','k','y','m','c','b','r','g','k','y','m'

if ~isempty(filename2)

%Read first jet
if iscell(filename2)
    filedirectory = strcat(sdir2,filename2{1});
else
    filedirectory = strcat(sdir2,filename2);
end

%Load jet boundary and set reference
B=load(filedirectory);
max_ref = max(B(:,2));
reference = get(handles.btnAlignLeft2,'Value');

%Number of jets to be represented
if iscell(filename2)
    n_iter = length(filename2);
else
    n_iter = 1;
end

%Loop to represent all jets acording to representation options
for i=1:n_iter
```

```
    hold on

    %Read jet boundary file
    if iscell(filename2)
      filedirectory = strcat(sdir2,filename2{i});
    else
      filedirectory = strcat(sdir2,filename2);
    end

    %Load jet boundary and save max x value
    B=load(filedirectory);
    max_actual(i) = max(B(:,2));

    %Coordinates change to set middle of jet as 0
    index_ChangeCoordinates = find(B(:,2) == max_actual(i));
    ChangeCoordinates_aux = [];
    for j=1:length(index_ChangeCoordinates)
        ChangeCoordinates_aux(j) = B(index_ChangeCoordinates(j),1);
    end
    ChangeCoordinates(i) = mean(ChangeCoordinates_aux);
    MeanChangeCoordinates = mean(ChangeCoordinates);

    %Coordinates change to put move jets to right or left side
    if reference == 1
        columna_x = B(:,2);
    else
        columna_x = B(:,2)+(max_ref-max_actual(i));
    end
    columna_y = B(:,1)-MeanChangeCoordinates;
    max_actual_y(i) = max(columna_y);

    %Plot current jet
    plot(columna_x,columna_y,C{i},'lineWidth',2);

    %Generate legend with filename
    if iscell(filename2)
      filename_local = filename2{i};
      filename_legend{i} = filename_local(1:end-4);
    else
      filename_local = filename2;
      filename_legend{i} = filename_local(1:end-4);
    end


end

%Set figure properties

    if iscell(filename2)
    max_string_x = num2str(round(max(max_actual)));
    num_cifras_x = length(max_string_x);
    x_lim = (round(max(max_actual)/
(10^(num_cifras_x)),1)+0.1)*(10^num_cifras_x);
    xlim ([0 x_lim])
```

```
    paso_x = x_lim/20;
    xticks(0:paso_x:x_lim)

    max_string_y = num2str(round(max(max_actual_y)));
    num_cifras_y = length(max_string_y);
    y_lim = round(max(max_actual_y)/
(10^(num_cifras_y)),1)*(10^num_cifras_y);
    ylim ([-y_lim y_lim])
    paso_y = (2*y_lim)/10;
    yticks(-y_lim:paso_y:y_lim)
    end

    grid on
    grid minor
    axis equal
    xlabel('µm')
    ylabel('µm')
    lgd=legend(filename_legend,'Location','NorthEastOutside');
    set(lgd,'FontSize',12);

end
```

--- Executes on button press in btnRight3.

```
function btnRight3_Callback(hObject, eventdata, handles)
% hObject    handle to btnRight3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of btnRight3


% --- Executes on button press in btnLeft3.
function btnLeft3_Callback(hObject, eventdata, handles)
% hObject    handle to btnLeft3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of btnLeft3


% --- Executes on mouse press over figure background, over a disabled
 or
% --- inactive control, or over an axes background.
function figure1_WindowButtonDownFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

Published with MATLAB® R2018a