Capítulo 3

Alcance del proyecto

En este capítulo se va a describir con detalle en qué ha consistido el trabajo desarrollado en el Proyecto Fin de Carrera.

En la primera parte se realizará una descripción general de las aplicaciones desarrolladas en el proyecto, para tener una visión de conjunto del trabajo realizado.

En la segunda parte se explicará la estructura y los componentes del sistema de adquisición y análisis de imágenes.

En la tercera parte se explicarán en detalle los programas realizados: su contenido, su estructura, los algoritmos que se utilizan, y ejemplos de cada parte de los programas.

3.1. Descripción de las aplicaciones desarrolladas

En el proyecto se han desarrollado dos programas: SparksSolver y SparksAnalyzer, cuya estructura y características se explicarán con más detalle en el apartado 3.3 en la página 47.

SparksSolver sirve como paso previo al funcionamiento de SparksAnalyzer, ya que es una aplicación diseñada para el estudio detallado de imágenes que contienen chispas, con el objetivo de determinar parámetros característicos de cada proceso de rectificado, en una fase de estudio e investigación.

El objetivo de SparksAnalyzer en cambio es un análisis menos preciso y más rápido de vídeos de procesos de rectificado de metales, que contienen gran cantidad de imágenes, con el objetivo de determinar si el proceso se ha llevado a cabo de forma correcta y poder monitorizarlo. Como se sacrifica precisión a cambio de velocidad en el procesamiento, se utilizan los parámetros obtenidos en el aprendizaje mediante SparksSolver.

3.1.1. SparksSolver.exe

SparksSolver es una aplicación para el análisis de imágenes independientes, esto es, imágenes que el usuario le proporciona al programa individualmente, no a través de un flujo de entrada de video.

El objetivo de la aplicación es obtener la mayor cantidad de información posible sobre el tipo de imágenes que se obtienen de un proceso de rectificado determinado. Cada proceso tendrá unas características muy concretas en cuanto a tipo de metal, composición de la muela, cantidad de material que se está desprendiendo, localización de la zona de trabajo, etc.

El funcionamiento es el siguiente: el usuario selecciona en primer lugar una imagen de la zona de trabajo en reposo, sin que se esté trabajando. Esta imagen servirá como fondo estático, ya que no varía durante el proceso, y se sustraerá a la imagen a analizar - que sí contendrá chispas -.
Una vez seleccionadas las dos imágenes y sustraído el fondo, se procede al análisis de la imagen. Los algoritmos que se aplican se detallan en el apartado 3.3 en la página 47. El usuario puede ver en la pantalla de la aplicación los datos extraídos, así como imágenes resultado de los algoritmos de Flujo óptico y Transformada de Hough, e histogramas de las bandas de la imagen. Además se genera un fichero de salida con todos los datos calculados.

Con la información obtenida, el usuario puede realizar un estudio de cada proceso por separado, y obtener los parámetros característicos de operaciones correctas e incorrectas en cada circunstancia. El siguiente paso es trasladar lo aprendido al programa SparksAnalyzer, para el análisis de vídeos completos de procesos de rectificado.

3.1.2. SparksAnalyzer.exe

SparksAnalyzer analiza imágenes pertenecientes a un vídeo en formato mpeg. El análisis que realiza es menos profundo que el de SparksSolver, ya que su objetivo no es tanto el estudio del vídeo sino obtener las características básicas del proceso. Además analiza gran cantidad de imágenes que componen el flujo de vídeo, no una.

El funcionamiento de este programa es el siguiente: el usuario selecciona el vídeo que se va a analizar. Automáticamente comienza el análisis, y se van presentando en la pantalla de la aplicación los principales datos extraídos, así como la imagen que se está analizando en cada momento y el resultado de los algoritmos de la Transformada de Hough y el flujo óptico. Estos y otros datos - como los histogramas de las bandas de la imagen - se recogen en un fichero de salida de texto.
Los parámetros de funcionamiento del programa se pueden configurar en la aplicación previamente a cada análisis. El funcionamiento óptimo de SparksAnalyzer se consigue utilizando la información obtenida del estudio previo con SparksSolver.

3.2. Componentes del sistema

En este apartado se describe la estructura y los componentes que forman parte del sistema de análisis de imágenes desarrollado.

En la primera parte se enumeran y describen los elementos que componen el sistema de trabajo.

En la segunda parte se especifica cómo debe ser la instalación del entorno de trabajo, en cuanto a localización de los elementos que lo componen y las condiciones de iluminación.

En la tercera parte se explican las características del entorno de programación utilizado en el desarrollo de los programas del proyecto, así como las librerías específicas empleadas.

3.2.1. Dispositivos

Los elementos que intervienen en el funcionamiento de las aplicaciones son la cámara para adquirir las imágenes, y el ordenador donde se ejecutan los programas.

Adquisición de imágenes

El programa SparksSolver trabaja con los formatos de imágenes admitidos por la librería Java Advanced Imaging (JAI, ver sección 3.2.3 en la página 43), como por ejemplo JPEG y BMP. Por tanto la aplicación es versátil en cuanto a formatos, y la mayoría de las cámaras fotográficas del mercado sirven para la adquisición de imágenes compatibles. Además en caso de usar una cámara que trabaje con un formato no admitido por JAI, existe software tanto comercial como libre que puede transformar la imagen a un formato adecuado.

SparksAnalyzer utiliza la librería Java Media Framework (JMF, ver sección 3.2.3 en la página 45) para trabajar con vídeo. Aunque la librería admite numerosos formatos, el programa se ha diseñado para trabajar con MPEG. En este caso no se ha permitido la versatilidad de la primera aplicación, ya que los diferentes formatos de vídeo requieren un tratamiento diferente a la hora de extraer frames para su análisis.

Por consiguiente la cámara de vídeo que se utilice para grabar el trabajo del robot debe proporcionar vídeo en formato MPEG, o utilizar posteriormente un software de conversión de vídeo.

En cuanto a la resolución de las imágenes y el vídeo, ambas aplicaciones son flexibles. En ambas se realiza un escalado de las imágenes o frames a un tamaño adecuado para su representación en la pantalla de la aplicación, y es la imagen escalada la que se analiza. Además en SparksAnalyzer es posible seleccionar la tasa de procesamiento de frames del vídeo.

Equipo donde ejecutar las aplicaciones

Las aplicaciones desarrolladas en el proyecto se han escrito en lenguaje Java, por lo que se pueden ejecutar sobre cualquier sistema operativo. Precisamente este es uno de los motivos por los que se ha elegido dicho lenguaje, dada su portabilidad o independencia de la plataforma (ver Apéndice B).

Cabe destacar que dado el elevado coste computacional de algunos de los algoritmos implementados, conviene que el ordenador donde se ejecuten los programas tenga buenas prestaciones en cuanto a velocidad y memoria RAM. De este modo las ejecuciones requerirán menos tiempo.

Como ejemplo, a continuación se detallan las características del equipo donde se ha desarrollado el proyecto:

- Procesador: Pentium 4 CPU 3.20GHz
- Memoria: 384MB de RAM
- Tarjeta gráfica: ATI Mobility Radeon 9000 IGP
- Sistema operativo: Windows XP

3.2.2. Entorno de trabajo

El entorno de trabajo es una instalación compuesta por la zona de trabajo, con el robot y la pieza que se va a rectificar, y la cámara que graba el proceso. En la figura 3.1 se muestra un esquema.

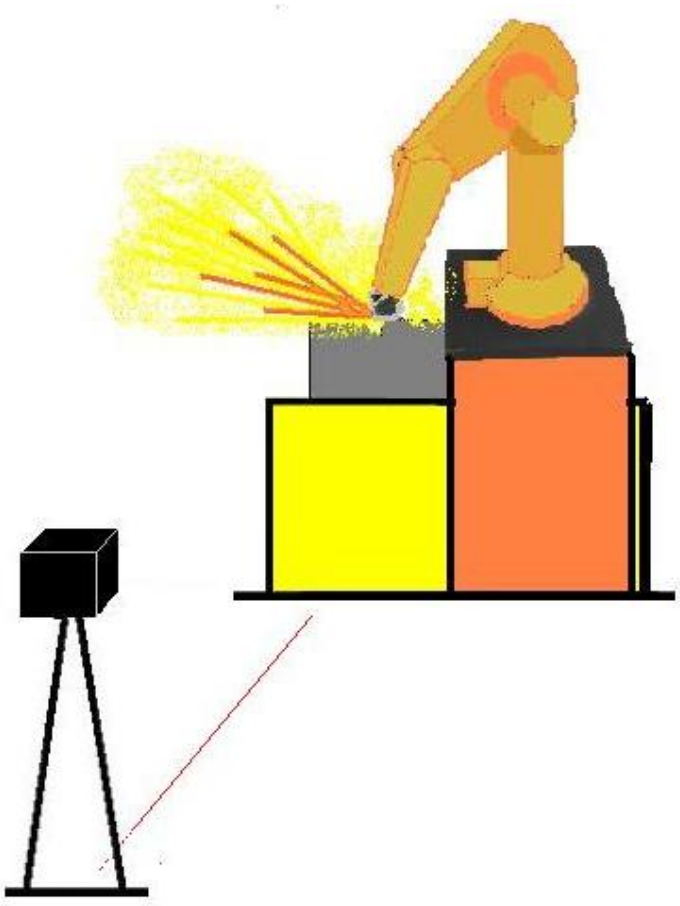

Figura 3.1: Esquema de instalación de trabajo

La cámara se debe colocar a una distancia lo suficientemente grande como para que el arco que describen las chispas quepa en la imagen, para tener así una visión más completa de la trayectoria de las chispas.

En cuanto a la iluminación, las características del proceso que se va a captar no lo hacen excesivamente dependiente de las condiciones de iluminación, ya que los elementos de interés son las chispas, que se caracterizan por un nivel de intensidad muy elevado. Prácticamente se corresponden con píxeles de intensidad saturada aún en imágenes oscuras.
En el caso de existir fuentes de iluminación estáticas que aparezcan en la imagen, se eliminan al sustraer la imagen de fondo a la imagen que se analiza, de manera que esto tampoco influye en el análisis.

3.2.3. Entorno de programación

The Eclipse Project

Como se ha mencionado anteriormente, las aplicaciones desarrolladas en el proyecto se han realizado en lenguaje Java. Como editor se ha utilizado Eclipse SDK versión 3.0.1. Información detallada sobre su instalación puede encontrarse en el Apéndice A.
Puede descargarse de la página web www.eclipse.org.

El Proyecto Eclipse es un proyecto de desarrollo de software de código abierto dedicado a proporcionar una plataforma industrial robusta, con amplias características y con calidad comercial para el desarrollo de herramientas altamente integradas.

Está compuesto de tres subproyectos: la Plataforma Eclipse, la Java Development Tool y el Plug-in Development Environment. El éxito de la Plataforma Eclipse depende de cómo sea capaz de admitir una amplia gama de herramientas de desarrollo para reproducir lo mejor posible las herramientas existentes en la actualidad.

Arquitectura Eclipse lo forman el núcleo, el entorno de trabajo (Workspace), el área de desarrollo (Workbench), la ayuda al equipo (Team support) y la ayuda o documentación (Help).

- Núcleo: su tarea es determinar cuales son los plug-ins disponibles en el directorio de plug-ins de Eclipse. Cada plug-in tiene un fichero XML

manifest que lista los elementos que necesita de otros plug-ins así como los puntos de extensión que ofrece. Como la cantidad de plug-ins puede ser muy grande, sólo se cargan los necesarios en el momento de ser utilizados con el objeto de minimizar el tiempo de arranque de Eclipse y los recursos.

- Entorno de trabajo o Workspace: maneja los recursos del usuario, organizados en uno o más proyectos. Cada proyecto corresponde a un directorio en el directorio de trabajo de Eclipse, y contienen archivos y carpetas.

- Interfaz de usuario o Workbench: muestra los menús y herramientas, y se organiza en perspectivas que configuran los editores de código y las vistas. A diferencia de muchas aplicaciones escritas en Java, Eclipse tiene el aspecto y se comporta como una aplicación nativa. No está programada en Swing, sino en SWT (Standard Widget Toolkit) y Jface (juego de herramientas construida sobre SWT), que emula los gráficos nativos de cada sistema operativo. Este ha sido un aspecto discutido sobre Eclipse, porque SWT debe ser portada a cada sistema operativo para interactuar con el sistema gráfico. En los proyectos de Java puede usarse AWT y Swing salvo cuando se desarrolle un plug-in para Eclipse.
  En la figura 3.2 se muestra el aspecto del workbench utilizado.

- Ayuda al grupo: este plug-in facilita el uso de un sistema de control de versiones para manejar los recursos en un proyecto del usuario y define el proceso necesario para guardar y recuperar de un repositorio. Eclipse incluye un cliente para CVS.

- Documentación: al igual que el propio Eclipse, el componente de ayuda es un sistema de documentación extensible. Los proveedores de herramientas pueden añadir documentación en formato HTML y, usando XML, definir una estructura de navegación.

Eclipse está escrito en su mayor parte en Java (salvo el núcleo), se ejecuta sobre la máquina virtual Java y su uso más popular es como un IDE para Java. Sin embargo Eclipse es adaptable a cualquier tipo de lenguaje, por ejemplo C/C++, Cobol, C#, XML, etc.

La característica clave de Eclipse es la extensibilidad. Eclipse es una gran estructura formada por un núcleo y muchos plug-ins que van conformando la funcionalidad final. La forma en que los plug-ins interactúan es mediante

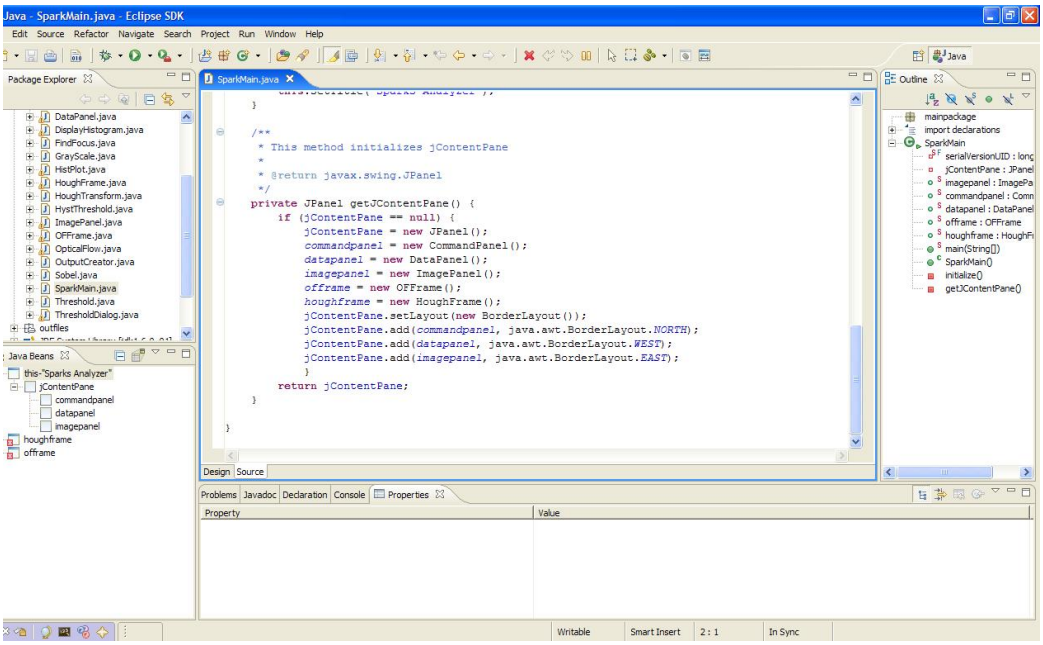

Figura 3.2: Entorno de trabajo o Workbench

interfaces o puntos de extensión; así, las nuevas aportaciones se integran sin dificultad ni conflictos.
Para el desarrollo de este proyecto se han instalado adicionalmente los plug-ins:
- GEF runtime 3.1
- VE runtime 1.1.0
- EMF SDO runtime 2.1.0

Librería JAI

Java Advanced Imaging (JAI) es la librería de procesamiento de imágenes desarrollada por Sun para Java. Se encuentra aún en fase de desarrollo, por lo que se espera que su funcionalidad y potencialidad (todavía no demasiado amplias) se incremente próximamente. Permite el trabajo con imágenes aisladas e implementa los algoritmos básicos de procesamiento y tratamiento.

El objetivo de JAI es soportar el procesamiento de imágenes en lenguaje Java de la forma más general posible, de manera que pocas o ninguna aplicación de procesamiento de imágenes esté fuera de su alcance. Al mismo tiempo, JAI presenta un modelo de programación sencillo listo para su uso

en aplicaciones, sin requerir un gran esfuerzo previo de programación o que el programador sea experto en todas las fases del diseño de la API (Application Programming Interface).

JAI encapsula formatos de datos de imagen e invocaciones a métodos remotos en un objeto imagen reutilizable, permitiendo que una imagen en un archivo, o un objeto imagen en red, o un flujo de datos en tiempo real sean procesados idénticamente. Así, JAI representa un modelos de programación simple ocultando la complejidad de sus mecanismos internos.

Características de JAI JAI pretende cumplir los requerimientos de todos los mercados de tratamiento de imágenes, y más. Ofrece numerosas ventajas para los desarrolladores de aplicaciones comparado con otras soluciones. Algunas de estas ventajas se describen a continuación.

- Independencia de la plataforma.
  Mientras que muchas APIs de procesamiento de imágenes están diseñadas para un sistema operativo específico, JAI sigue el modelo de librerías 'run time'de Java, consiguiendo independencia de la plataforma. Las implementaciones de las aplicaciones de JAI pueden ejecutarse en cualquier equipo donde esté instalada la Máquina Virtual Java (Java Virtual Machine, JVM). De esta forma, cumple la máxima del lenguaje Java: "Write once, run anywhere".

- Procesamiento de imágenes distribuido.
  JAI permite aplicaciones cliente-servidor a través de la arquitectura de red de la plataforma Java y de las tecnologías de ejecución remota. La ejecución remota se basa en RMI (Remote Method Invocation) de Java. Esta tecnología permite al código Java en un cliente invocar métodos en objetos que residen en otro equipo sin tener que trasladar esos objetos al cliente.

- API orientada a objetos.
  Como Java, JAI es totalmente orientado a objetos. Las imágenes y las operaciones de procesamiento están definidas como objetos. JAI unifica los conceptos de imagen y operador haciéndolos dos subclases de otra común. Un objeto operador puede ser 'instanciado'por una o varias imágenes y otros parámetros. Este operador puede transformarse en una imagen fuente para el siguiente operador. Las conexiones entre los objetos definen el flujo de los datos procesados.

- Flexible y extensible.
  Además de un conjunto de operadores para operaciones básicas y frecuentes, a veces se necesitan operadores más complejos que rara vez se utilizan en la mayoría de las aplicaciones. Para las aplicaciones más especializadas, JAI proporciona un entorno de trabajo extensible que permite añadir soluciones propias al núcleo de la API. JAI proporciona un conjunto de métodos de compresión y descompresión de imágenes para los tipos de archivos más comunes. Para otro tipos es posible añadir codecs.

- Independencia del dispositivo de visualización.
  El procesamiento de imágenes se puede especificar en coordenadas independientes del dispositivo. La traslación de píxeles se lleva a cabo en tiempo de ejecución. Las operaciones no tienen en cuenta el tamaño de la imagen ni su resolución, JAI se encarga de los detalles de las operaciones. Para poder desarrollar aplicaciones independientes de la plataforma, JAI no asume una resolución del dispositivo de salida, ni espacio o modelo de color. La API tampoco asume un determinado formato de archivo. Los archivos de imagen pueden ser adquiridos y manipulados sin que el programador tenga conocimiento alguno del formato.

- Potencia.
  JAI soporta formatos complejos, incluyendo imágenes de más de tres dimensiones y un número arbitrario de bandas. Incorpora numerosos algoritmos, y otros deben añadirse según las necesidades.

- Compatible.
  JAI está integrada con el resto de APIs multimedia de Java, permitiendo elaborar aplicaciones más complejas y sofisticadas. JAI trabaja bien con APIs como Java 3D y Java component. Además en la mayoría de los casos las clases de JAI son compatibles con las clases básicas de Java.

Librería JMF

Java Media Framework (JMF) es una librería para el trabajo multimedia en tiempo real. Permite capturar, almacenar y presentar tanto imágenes y video como audio. Se ha utilizado para trabajar con vídeos.

JMF es una API (Application Programming Interface) para incorporar medios basados en el tiempo en aplicaciones Java y en applets.

JMF 1.0 API (Java Media Player API) permitió a los programadores desarrollar programas en Java con datos basados en tiempo. JMF 2.0 amplía el entorno de trabajo para proporcionar apoyo para capturar y almacenar datos multimedia, controlar el tipo de procesamiento que se aplica, y realizar procesamiento propio en flujos de datos multimedia.
Además JMF 2.0 define una "plug-in API" o API basada en plug-ins que permite a los desarrolladores expertos personalizar y extender la funcionalidad de JMF.

Objetivos de JMF 2.0 JMF 2.0 ha sido diseñado para:

- Ser fácil de programar.
- Permitir la captura de datos multimedia.
- Permitir el desarrollo de aplicaciones de flujo de datos multimedia y conferencia en Java.
- Facilitar a los programadores expertos y proveedores de tecnología implementar soluciones personalizadas basadas en la API existente, e integrar nuevas funciones dentro del entorno de trabajo existente.
- Dar acceso a flujos de datos sin tratar.
- Permitir el desarrollo de demultiplexores, codecs, procesadores, multiplexores y visualizadores propios mediante plug-ins.
- Conservar la compatibilidad con JMF 1.0.

Java Media Framework proporciona una arquitectura unificada y un protocolo de mensajes para gestionar la adquisición, el procesamiento y el transporte de datos basados en tiempo.
JMF ha sido diseñado para soportar la mayoría de los tipos de archivo multimedia, como AIFF, AU, AVI, GSM, MIDI, MPEG, QuickTime, RMF, y WAV.
Con la ventajas de la plataforma Java, JMF cumple la máxima "Write Once, Run Anywhere" en programas que utilizan medios como audio y vídeo. JMF proporciona una API independiente de la plataforma para acceder a los medios que se encuentran en ella.

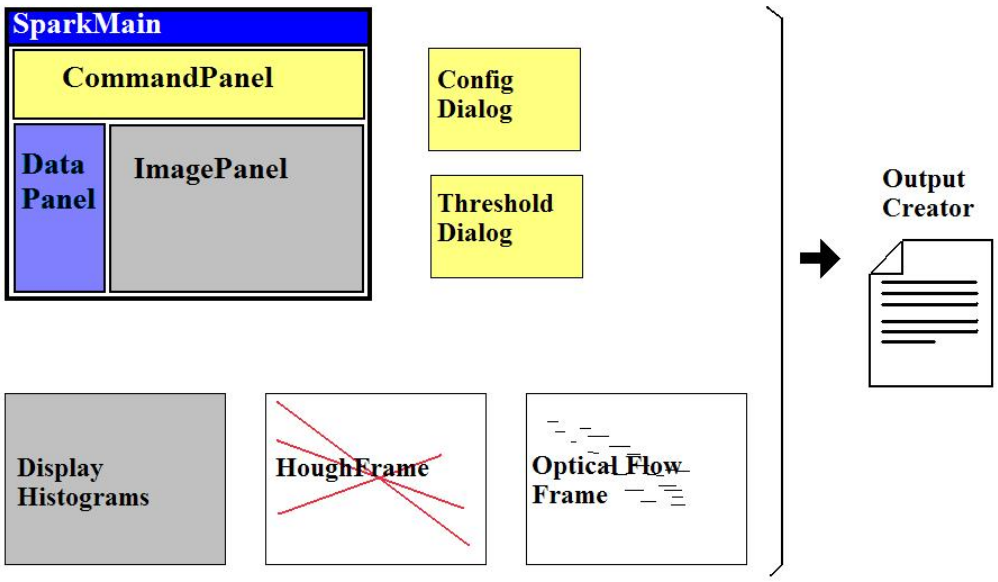

Figura 3.3: Estructura en bloques funcionales de SparksSolver

3.3. Implementación

En este apartado se va a explicar el trabajo realizado en cuanto a programación: los algoritmos desarrollados, la estructura de los programas, las técnicas empleadas y el uso de librerías.

El código fuente de los programas realizados se incluye en el CD adjunto a esta memoria. En el Apéndice B se explican con más detalle las funciones de librería que se han empleado en las funciones diseñadas.

3.3.1. Diseño

La estructura en bloques funcionales de los programas SparkSolver y SparksAnalyzer puede observarse en las figuras 3.3 y 3.4 respectivamente.

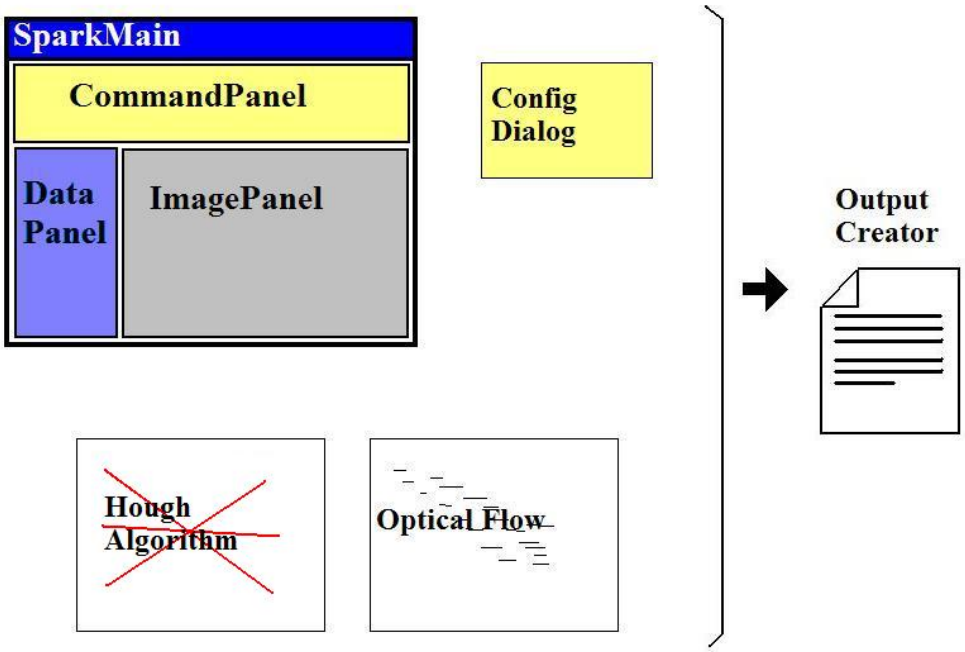

Figura 3.4: Estructura en bloques funcionales de SparksAnalyzer

Estructura de SparksSolver

La estructura en bloques funcionales de SparksSolver se corresponde prácticamente con su estructura visual en ventanas.

La clase principal del programa, SparkMain.java, se corresponde con el JFrame de la ventana principal de la aplicación, cuya función es recoger y declarar el resto de los componentes del programa. En el JFrame SparkMain se encuentran los paneles CommandPanel, DataPanel e ImagePanel, que a su vez se corresponden con bloques funcionales.

CommandPanel tiene la función de lanzar las operaciones según las ordena el usuario. A CommandPanel pertenecen los cuadros de configuración ConfigDialog y ThresholdDialog.
La función de ConfigDialog es presentar al usuario todos los parámetros de configuración del programa, y una vez seleccionados enviarlos a todas las clases que los utilizan.
ThresholdDialog tiene como función configurar la clase Threshold, que realiza una operación de evaluación del tamaño de las chispas según un nivel de intensidad fijado por el usuario, y ordenar el comienzo de dicha operación.

DataPanel engloba las funciones relacionadas con el procesamiento básico de la imagen: cálculo de valores máximos, medios y mínimos de cada banda y elaboración de histogramas. Como elemento visual, presenta todos los datos calculados, y permite ordenar la visualización de los histogramas.

ImagePanel es el encargado de abrir las imágenes a analizar, escalarlas para que tengan un tamaño adecuado, presentarlas en la ventana de la aplicación y establecerlas como fuente para el procesamiento posterior.

Existen bloques que no están contenidos en SparksMain, y que realizan otras operaciones independientes:
DisplayHistograms se encarga de presentar gráficamente los histogramas calculados en una ventana independiente.

HoughFrame realiza todas las operaciones implicadas en el cálculo de la Transformada de Hough de la imagen a analizar. Asimismo, representa sobre la imagen las líneas resultantes del algoritmo, y localiza el foco de las chispas. También recoge el resultado de la integración de este algoritmo y el del flujo óptico. Estos resultados se presentan visualmente en la ventana HoughFrame.

Optical Flow Frame o OFFrame, análogamente a HoughFrame, realiza las operaciones necesarias para calcular el flujo óptico de la imagen origen. Posteriomente crea un gráfico para representar el resultado, y lo presenta en su ventana.

Por último, OutputCreator es la clase que se corresponde con el bloque relativo al fichero de salida. Su función es crear el fichero, enviarle todos los datos calculados a lo largo de la ejecución del programa y cerrarlo al finalizar el análisis.

Estructura de SparksAnalyzer

La estructura de SparksAnalyzer es análoga y similar a la de SparksSolver, que se ha descrito en el punto anterior. Varían las funciones concretas de algunos bloques, que pasamos a explicar.

En este caso la clase principal del programa es el JFrame VideoMain.java, cuyas funciones y estructura son idénticas a SparkMain.java.

Sólo hay un cuadro de configuración dependiente de CommandPanel, ConfigDialog.
A ConfigDialog se añaden para esta aplicación parámetros de configuración

relacionados con el manejo de vídeo, y se incluye la configuración que en SparksSolver se llevaba a cabo con ThresholdDialog.

A DataPanel pertenecen las funciones relacionadas con el procesamiento básico de la imagen: cálculo de valores máximos, medios y mínimos de cada banda y elaboración de histogramas. También presenta los datos calculados en pantalla.

El bloque ImagePanel ve incrementadas sus funciones, ya que el trabajo con vídeo es más complejo que el trabajo con imágenes aisladas. Abre el vídeo seleccionado por el usuario, y crea las estructuras de datos necesarias para manejarlo. Una vez establecido el flujo o stream de datos, se encarga de la recepción y muestreo de frames. Cada frame se transforma en una imagen, que se escala, se presenta en el panel y se envía a las clases encargadas de procesarla. Por último, organiza y lanza todo el proceso del análisis.
Cuando termina el flujo de datos, se cierran los medios utilizados y se ordena la creación del fichero de salida.

En esta aplicación no hay más elementos visuales. El resto de los bloques funcionales no llevan ligada una representación gráfica en forma de ventana independiente.
HoughAlgorithm realiza todas las operaciones implicadas en el cálculo de la Transformada de Hough de la imagen a analizar. Representa sobre la imagen las líneas resultantes del algoritmo, y localiza el foco de las chispas. La imagen resultante se envía a ImagePanel para que la presente en pantalla.

OpticalFlow realiza las operaciones necesarias para calcular el flujo óptico de la imagen entre dos imágenes consecutivas del vídeo. Los datos obtenidos se envían a DataPanel para su presentación y al fichero de salida.

OutputCreator es la clase gestora del fichero de salida. Su función es crear el fichero, enviarle todos los datos calculados a lo largo de la ejecución del programa y cerrarlo al finalizar el vídeo.

3.3.2. Diagramas de flujo

En las figuras 3.5 y 3.6 se muestran los diagramas de flujo de SparksSolver y SparksAnalyzer, respectivamente. En ellos los cuadros grises representan los botones de comando del programa, los amarillos las acciones que se llevan a cabo, y los azules eventos que ocurren en tiempo de ejecución. Como puede verse, la ejecución de los programas sigue las órdenes del usuario.

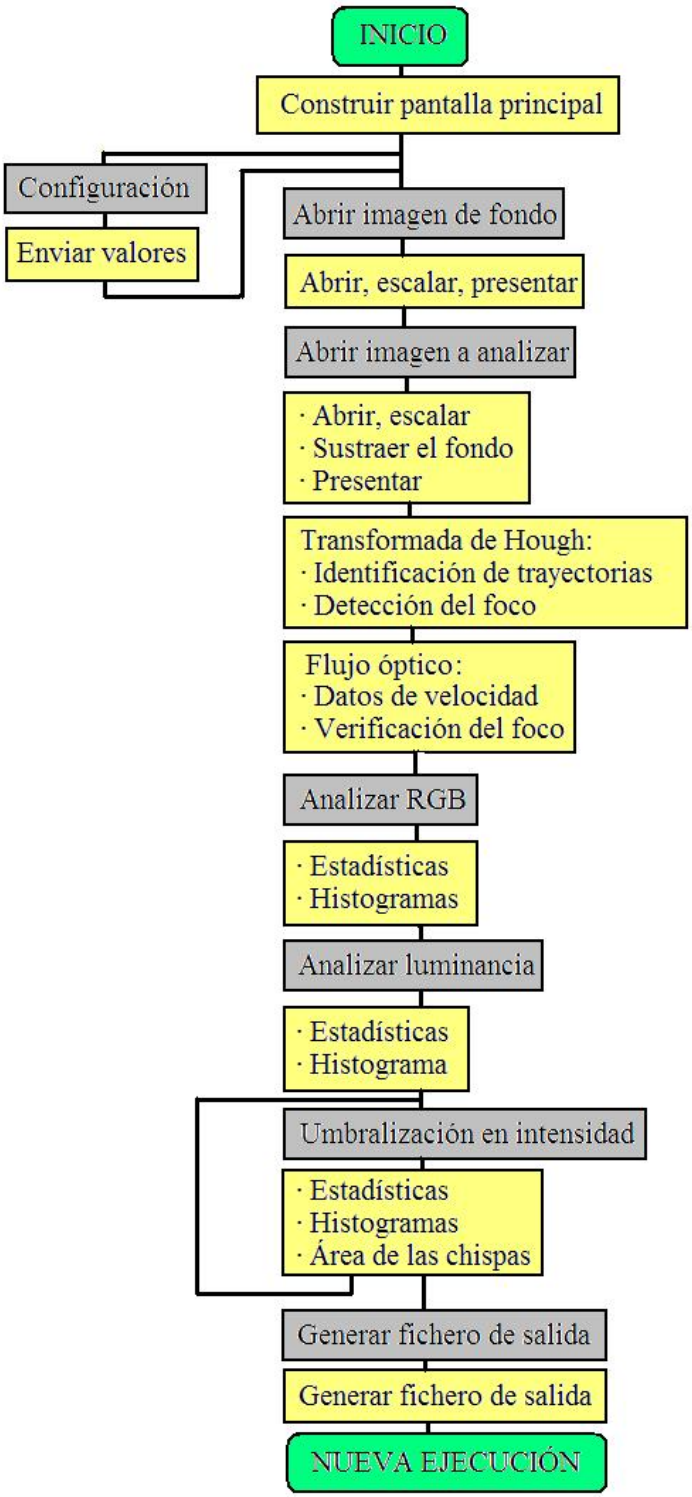

Figura 3.5: Diagrama de flujo de SparksSolver.

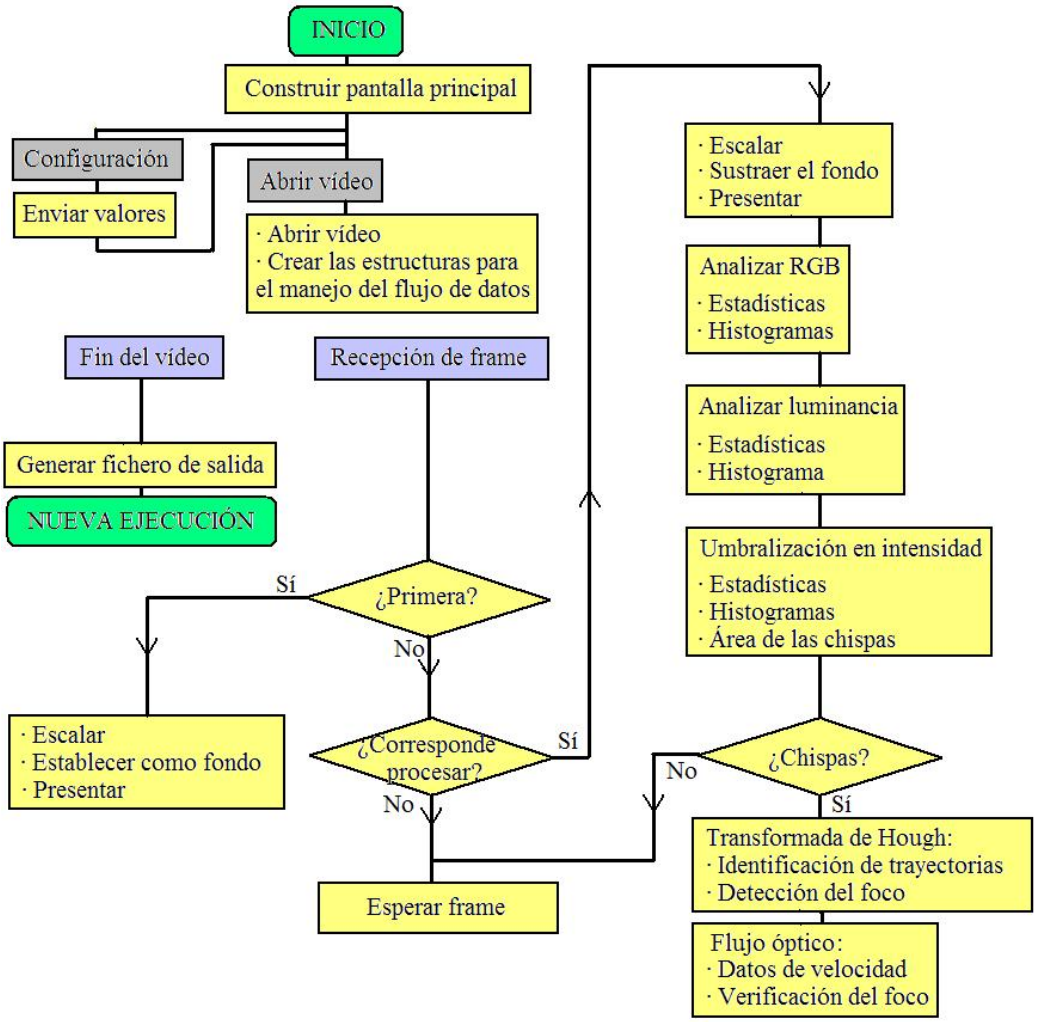

Figura 3.6: Diagrama de flujo de SparksAnalyzer.

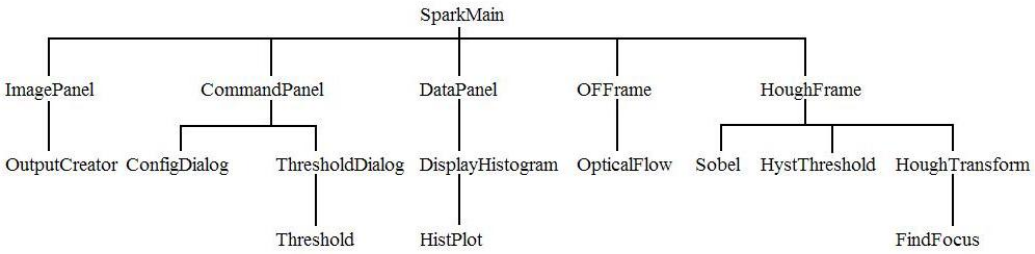

Figura 3.7: Estructura en clases de SparksSolver

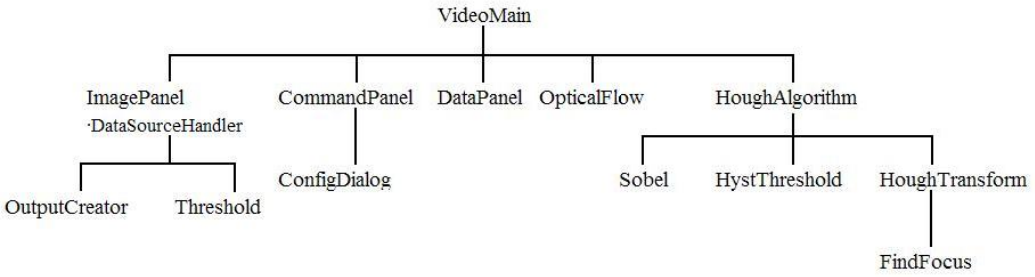

Figura 3.8: Estructura en clases de SparksAnalyzer

3.3.3. Diagrama de clases

La estructura de clases de las aplicaciones realizadas se ha diseñado de forma que cada clase realice un conjunto de funciones relacionadas entre sí. A cada clase se le asigna un cometido más o menos general, y se declaran los objetos necesarios para llevarlo a cabo. Para ilustrar el funcionamiento de la estructura de clases, se ha representado en las figuras 3.7 y 3.8 la división en clases y los objetos que se declaran en cada una.

Las operaciones y las estructuras de datos concretas que se deben utilizar en cada clase, se encuentran en las clases que se muestran partiendo de ellas en las figuras 3.7 y 3.8. En cada clase se declara un objeto definido por la clase que parte de ella, y de esta forma las operaciones se han estructurado en forma de árbol, con clases que realizan operaciones, y clases que realizan suboperaciones.

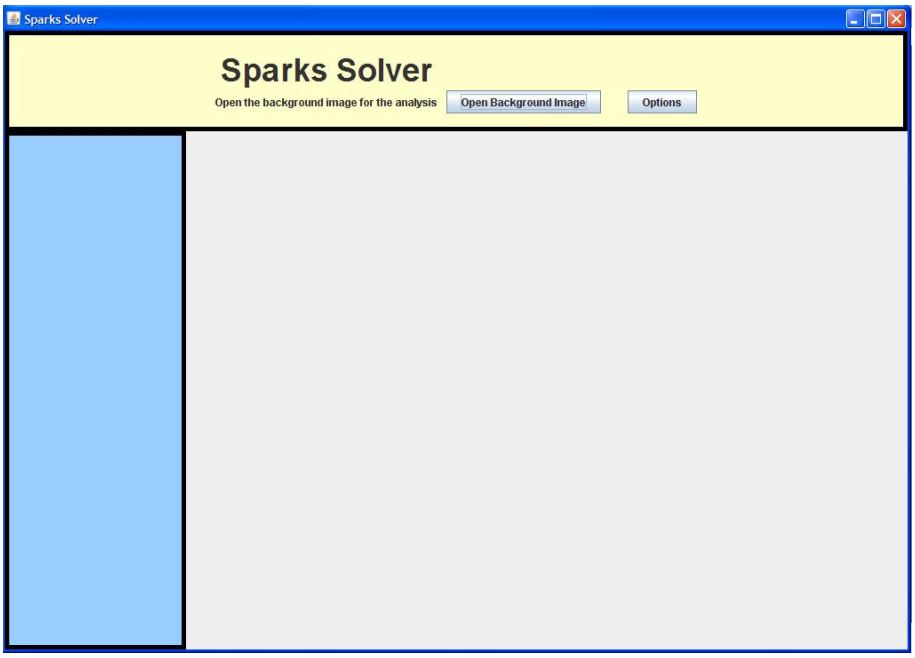

Figura 3.9: Aspecto de la pantalla principal de SparksSolver

Clases de SparksSolver

Como se ha explicado en el apartado anterior, 3.3.1, y como puede verse en la figura 3.7, existen cinco partes fundamentales en la división de las tareas del programa SparksSolver, que se corresponden con cinco clases: ImagePanel, CommandPanel, DataPanel, OFFrame y HoughFrame. Estas clases además son elementos visuales de la aplicación.
A continuación de explica el contenido y las funciones de cada clase:

- SparkMain: Clase principal del programa. Contiene el programa principal main. Es un componente visual JFrame. En él se declaran objetos de las clases que realizan las operaciones principales, y contiene a aquellos que son paneles visuales (ImagePanel, CommandPanel y DataPanel), de manera que se conforma la pantalla principal de la aplicación. El aspecto de dicha pantalla puede verse en la figura 3.9
- ImagePanel: Clase que abre, escala y presenta la imagen que está siendo procesada. Es un componente visual DisplayJAI, propio de la librería JAI, y que se comporta como un JPanel específico para presentar los tipos de objetos imagen definidos por JAI.

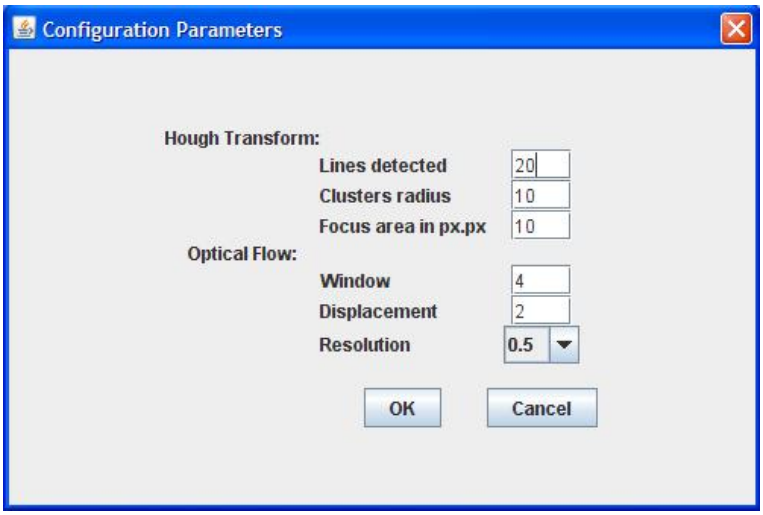

Figura 3.10: Aspecto del cuadro de configuración de SparksSolver

- OutputCreator: Clase que abre, gestiona y cierra el fichero de salida con los datos calculados por el programa. Para el manejo del fichero se importan import FileWriter, PrintWriter y Writer de java.io.

- CommandPanel: Componente visual JPanel que contiene todos los botones necesarios para el manejo del programa. Cuando se pulsa un botón, prepara lo necesario para ejecutar la orden recibida.

  - ConfigDialog: Clase cuadro de diálogo JDialog cuya función es permitir configurar los parámetros de funcionamiento del programa, enviando los valores seleccionados a las clases que los utilizan. El aspecto de ConfigDialog puede verse en la figura 3.10.
    Los parámetros de configuración de SparksSolver son los siguientes:
    Para la transformada de Hough:

    - Lines detected: número de líneas que va a detectar el algoritmo.
    - Clusters radius: radio en unidades de las nubes de puntos en r y theta que corresponden a una misma línea.
    - Focus area in px·px: área de la región que será identificada como foco de las chispas.

    Para el cálculo del flujo óptico:

Figura 3.11: Aspecto del cuadro de configuración ThresholdDialog de SparksSolver

- Window: tamaño (en px·px) de ventana de la imagen de la que se busca similitud.
- Displacement: desplazamiento máximo permitido en la búsqueda.
- Resolution: escalado previo a las imágenes a analizar en el algoritmo, para reducir el coste computacional a costa de perder precisión.

- ThresholdDialog: JDialog para configurar la operación 'thresholding'llevada a cabo por la clase Threshold. ThresholdDialog puede verse en la figura 3.11
  - Threshold: Clase que contiene los métodos asociados a la extracción de información de la imagen mediante el algoritmo threshold, que umbraliza en intensidad según el valor proporcionado por el usuario.

- DataPanel: Clase visual JPanel, cuya función es presentar en la pantalla de la aplicación los datos calculados a lo largo de la ejecución del programa. También permite ordenar la visualización de los histogramas calculados.
  Además de su función visual, lleva a cabo el procesamiento básico de la imagen: obtiene los valores máximo, medio y mínimo de cada banda de la imagen RGB y monocroma, y gestiona el cálculo de los histogramas.
  - DisplayHistogram: Clase visual JFrame que permite visualizar los histogramas generados por HistPlot. En la figura 3.12 puede verse un ejemplo.
    - HistPlot: Representa gráficamente los histogramas calculados.

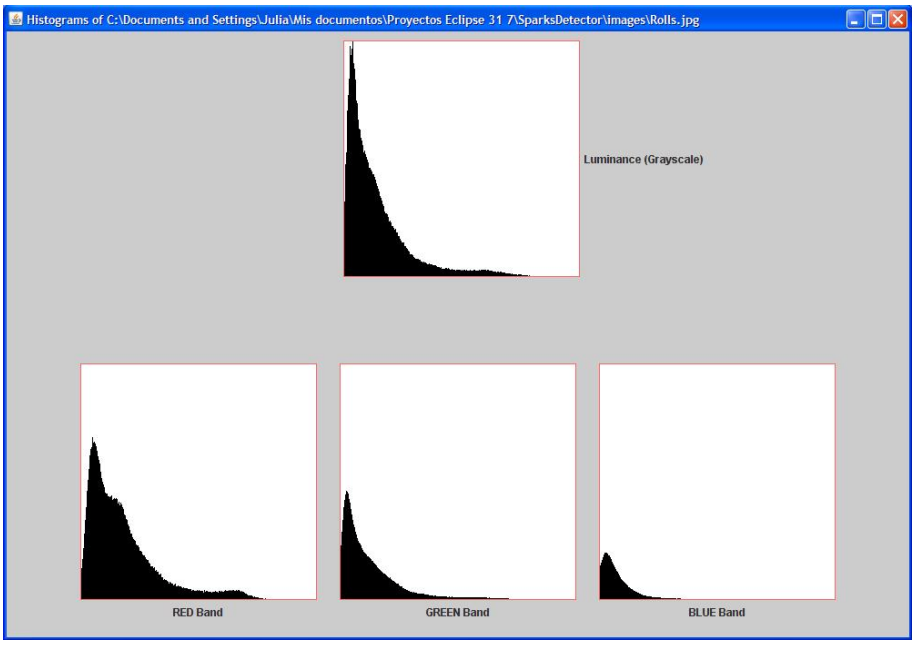

Figura 3.12: Ventana de representación de histogramas de SparksSolver

- OFFrame: Es un JFrame que presenta el la imagen resultado de calcular el flujo óptico a 2 imágenes, en este caso el background y la imagen a analizar proporcionadas al programa. Un ejemplo se muestra en la figura 3.13
  - OpticalFlow: Contiene los algoritmos y realiza todo el procesamiento necesario para calcular el flujo óptico entre dos imágenes. También relaciona los datos calculados con los obtenidos de la transformada de Hough.
- HoughFrame: Clase donde se realizan todas las operaciones para el cálculo de la transformada de Hough de la imagen, y donde se ordena la extracción de datos a partir de la misma. Como JFrame, permite presentar la imagen resultante en una ventana independiente.
  - Sobel: Realiza la detección de bordes en una imagen mediante el operador de Sobel.
  - HystThreshold: Clase donde se realiza una umbralización con histéresis a una imagen dada (concretamente al resultado de las operaciones realizadas por la clase Sobel.

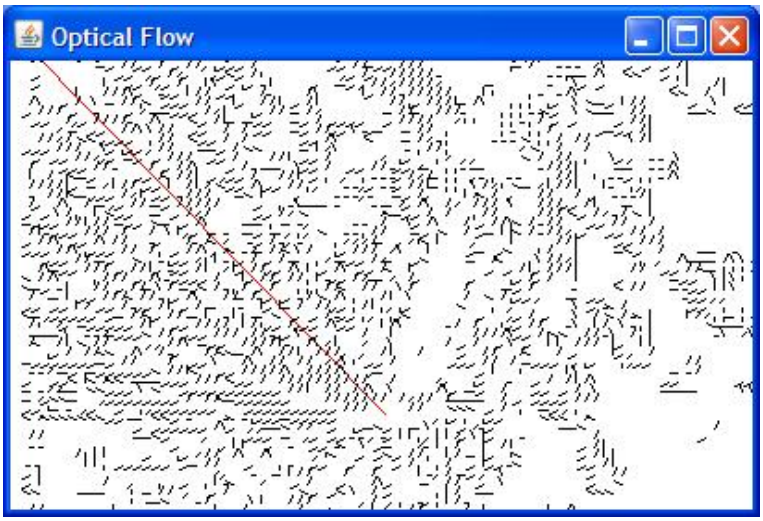

Figura 3.13: Ejemplo de representación del flujo óptico en SparksSolver

- HoughTransform: Se ocupa del cálculo de la transformada de Hough de una imagen de bordes binaria.
  - FindFocus: Contiene el algoritmo para la localización del foco de las chispas.

- Utilidad: GrayScale: Clase que contiene los mátodos para transformar una imagen RGB en monocroma.

Clases de SparksAnalyzer

Las clases de SparksAnalyzer son análogas a las de SparksSolver en cuanto a organización. Incorporan nuevas funcionalidades necesarias para operar con vídeos. Otras funcionalidades se han simplificado, ya que, como se ha comentado en el apartado 3.1 en la página 35, el procesamiento que se realiza en SparksAnalyzer es más simple en busca de mayor velocidad.

- VideoMain: Clase principal del programa. Contiene el programa principal main. Es un componente visual JFrame. En él se declaran objetos de las clases que realizan las operaciones principales, y contiene a aquellos que son paneles visuales (CommandPanel, DataPanel e ImagePanel), de manera que se conforma la pantalla principal de la aplicación. El aspecto de dicha pantalla puede verse en la figura 3.14

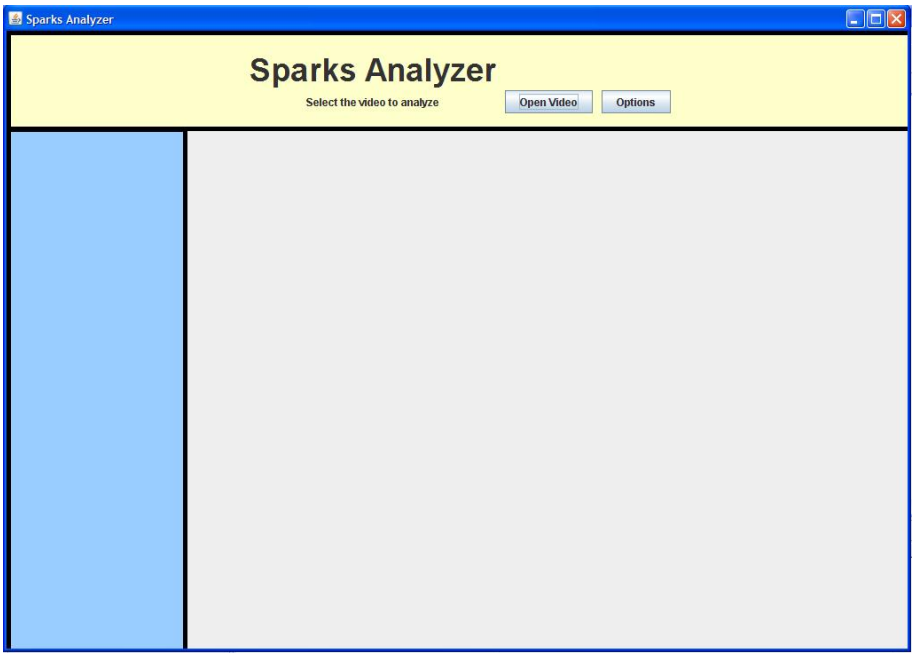

Figura 3.14: Aspecto de la pantalla principal de SparksAnalyzer

- ImagePanel: Clase que abre el vídeo seleccionado por el usuario, crea las estructuras de datos necesarias para manejarlo, recibe y muestrea el stream para obtener frames. Cada frame se transforma en una imagen, que se escala y se presenta en el panel. Es un componente visual DisplayJAI.
  - DataSourceHandler: Pertenece a ImagePanel. Su función es gestionar la utilización del DataSource asociado al vídeo que se está analizando, sus eventos y la transferencia de datos.
  - OutputCreator: Clase que abre, gestiona y cierra el fichero de salida con los datos calculados por el programa. Para el manejo del fichero se importan import FileWriter, PrintWriter y Writer de java.io.
  - Threshold: Su función es la extracción de información de la imagen mediante el algoritmo threshold, que umbraliza los píxeles de la imagen en intensidad según un valor proporcionado por el usuario.
- CommandPanel: Componente visual JPanel que contiene todos los botones necesarios para el manejo del programa. Cuando se pulsa un botón, prepara lo necesario para ejecutar la orden recibida.

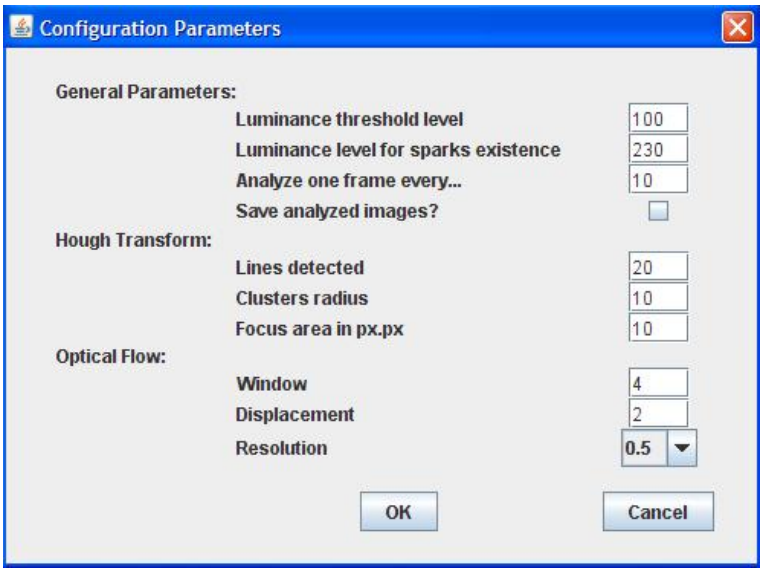

Figura 3.15: Aspecto del cuadro de configuración de SparksAnalyzer

- ConfigDialog: Clase cuadro de diálogo JDialog cuya función es permitir configurar los parámetros de funcionamiento del programa, enviando los valores seleccionados a las clases que los utilizan. Los parámetros son relativos tanto al procesamiento como al manejo de vídeo. El aspecto de ConfigDialog puede verse en la figura 3.15.

  Los parámetros de configuración de SparksAnalyzer son los siguientes:

  Parámetros generales:

  - Luminance threshold level: Nivel para la umbralización en intensidad que se utiliza en la evaluación del tamaño de la zona de chispas.
  - Luminance level for sparks existence: Nivel de intensidad máximo en la imagen a partir del cual se considera que en la imagen existen chispas. Permite ahorrar los algoritmos más costosos en imágenes en las que no hay chispas.
  - Analyze one frame every...: Tasa de análisis de las frames recibidas, para conseguir mayor velocidad. Por ejemplo una de cada... 10.
  - Save analyzed images?: Permite seleccionar si se guardan o no las imágenes analizadas en la carpeta 'images'.

Para la transformada de Hough:

- Lines detected: número de líneas que va a detectar el algoritmo.
- Clusters radius: radio en unidades de las nubes de puntos en r y theta que corresponden a una misma línea.
- Focus area in px·px: área de la región que será identificada como foco de las chispas.

Para el cálculo del flujo óptico:

- Window: tamaño de la ventana de la imagen de la que se busca similitud (en px·px).
- Displacement: desplazamiento máximo permitido en la búsqueda.
- Resolution: escalado previo a las imágenes a analizar en el algoritmo, para reducir el coste computacional a costa de perder precisión.

- DataPanel: Componente visual JPanel, cuya función es presentar en la pantalla de la aplicación los datos calculados a lo largo de la ejecución del programa. Además lleva a cabo el procesamiento básico de la imagen: obtiene los valores máximo, medio y mínimo de cada banda de la imagen RGB y monocroma, y gestiona el cálculo de los histogramas.

- OpticalFlow: Contiene los algoritmos y realiza todo el procesamiento necesario para calcular el flujo óptico entre dos imágenes. Relaciona los datos calculados con los obtenidos de la transformada de Hough.

- HoughAlgorithm: Clase donde se realizan todas las operaciones para el cálculo de la transformada de Hough de la imagen, y donde se ordena la extracción de datos a partir de la misma. El resultado se envía a ImagePanel para su presentación.

  - Sobel: Realiza la detección de bordes en una imagen mediante el operador de Sobel.
  - HystThreshold: Clase donde se realiza una umbralización con histéresis a una imagen dada (concretamente al resultado de las operaciones realizadas por la clase Sobel.
  - HoughTransform: Se ocupa del cálculo de la transformada de Hough de una imagen de bordes binaria.
    - FindFocus: Contiene el algoritmo para la localización del foco de las chispas.

- Utilidad: GrayScale: Clase que contiene los mátodos para transformar una imagen RGB en monocroma.

3.3.4. Funciones internas

En este apartado se van a explicar las funciones más importantes de los programas, que han sido elaboradas durante el proyecto. Se especifica el programa y la clase a la que pertenecen, la declaración, los parámetros, las funciones de librería que utilizan, y se describe su funcionamiento. Además se añaden algunos ejemplos.
Estas funciones son la realización práctica de las técnicas descritas en el apartado "Técnicas empleadas" del Capítulo 2.

- **SparksSolver.setBGImage y setImage**
  - Programa: SparksSolver
  - Clase: ImagePanel
  - Declaración: public void setBGImage(), public void setImage()
  - Funciones de librería utilizadas:
    - RenderedOp javax.media.jai.JAI.create(String arg0, Object arg1), con argumentos arg0 "stream" y "scale", para obtener la imagen de su archivo y escalarla unos valores dados, respectivamente.
  - Descripción: Estas dos funciones sirven para abrir las imágenes de fondo y fuente, respectivamente, y escalarlas para su presentación en ImagePanel. Utilizan los tipos RenderedOp y PlanarImage, definidos en JAI. Las imágenes se abren en RenderedOp, y tras el escalado se transforman en PlanarImage, tipo básico para operar con imágenes de JAI.
- **SparksAnalyzer.setBGImage y setImage**
  - Programa: SparksAnalyzer
  - Clase: ImagePanel
  - Declaración: private void setBGImage(int[] outpix), private void setImage(int[] outpix)
  - Parámetros:

    - int [] outpix: frame de vídeo obtenido en forma de matriz de enteros.
  - Funciones de librería utilizadas:
    - java.awt.image.MemoryImageSource.MemoryImageSource(int w, int h, ColorModel cm, int[] pix, int off, int scan), construye un objeto Image a partir de un vector de enteros.
    - RenderedOp javax.media.jai.JAI.create(String arg0, Object arg1), con argumento arg0 "AWTImage" para obtener una PlanarImage a partir de una Image.
  - Descripción: Estas dos funciones son análogas a sus homónimas de SparksSolver, con la particularidad de que no abren imágenes desde archivo, sino que reciben frames de vídeo en forma de matriz de enteros, y realizan la conversión a PlanarImage para su presentación en ImagePanel.

- **displayImage**
  - Programa: SparksSolver y SparksAnalyzer
  - Clase: ImagePanel
  - Declaración: public void displayImage (PlanarImage im)
  - Parámetros:
    - PlanarImage im: la imagen que se desea a presentar en pantalla.
  - Funciones de librería utilizadas:
    - void com.sun.media.jai.widget.DisplayJAI.set(RenderedImage arg0, int arg1, int arg2), para colocar una imagen arg0 en un panel DisplayJAI, con coordenadas arg1 y arg2.
  - Descripción: Presenta en ImagePanel la imagen que se le pasa como parámetro, permitiendo así su visualización en la pantalla principal de la aplicación.

- **subtract**
  - Programa: SparksSolver y SparksAnalyzer
  - Clase: ImagePanel
  - Declaración: public void subtract()
  - Funciones de librería utilizadas:

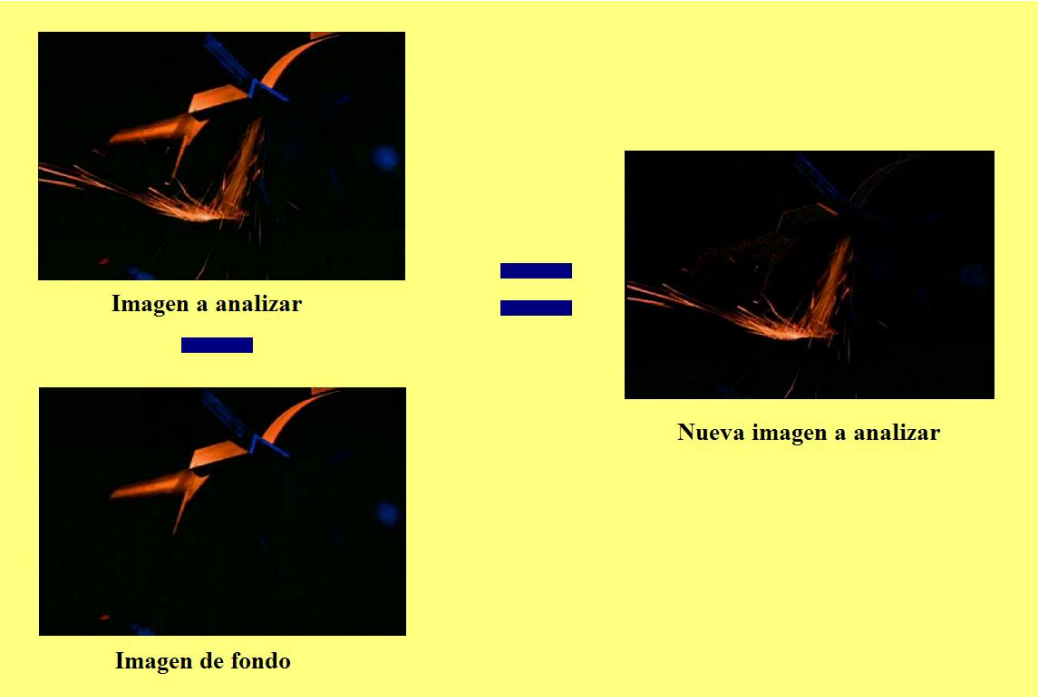

Figura 3.16: Operación de sustracción en la función subtract

  - RenderedOp javax.media.jai.JAI.create(String arg0, ParameterBlock arg1, RenderingHints arg2), con arg0="subtract" que es la operación que se lleva a cabo.

- Descripción: Este método sustrae a la imagen fuente la escena de fondo y establece el resultado como la nueva imagen fuente a analizar. Las imágenes implicadas en la operación son campos de la clase, por lo que no se pasan como parámetros.
  En SparksSolver, se llama a displayImage para mostrar el resultado.
  En SparksAnalyzer, no se muestra la sustracción, y además se realiza un escalado adicional de la imagen configurable por el usuario para el algoritmo del flujo óptico.
  En la figura 3.16 se ilustra el funcionamiento de esta función.

- **doAnalyze**

  - Programa: SparksAnalyzer
  - Clase: ImagePanel
  - Declaración: public void doAnalyze()

- Funciones de librería utilizadas:
  - java.net.URL.URL(String spec), crea un objeto URL a partir de una ruta de acceso en String.
  - javax.media.MediaLocator.MediaLocator(String arg0), crea un MediaLocator a partir de una representación en String de una URL.
- Descripción: Primera función encargada de la apertura del vídeo a analizar. En ella se realizan las conversiones necesarias a la ruta del fichero, para llamar finalmente a la función open, descrita a continuación.

- **open**
  - Programa: SparksAnalyzer
  - Clase: ImagePanel
  - Declaración: private boolean open(MediaLocator ml)
  - Parámetros:
    - MediaLocator ml: objeto MediaLocator obtenido en doAnalyze() a partir de la URL del fichero seleccionado por el usuario.
  - Tipo de datos de librería utilizado:
    - javax.media.Processor: objeto que permite manejar un flujo medios basados en el tiempo. Se utiliza para recibir el flujo de vídeo.
    - javax.media.protocol.DataSource: objeto que gestiona la transferencia de contenido de medios. Recoge los datos del flujo de vídeo.
  - Funciones de librería utilizadas:
    - Processor javax.media.Manager.createProcessor(MediaLocator arg0), crea un objeto Processor a partir de un MediaLocator.
    - void javax.media.Controller.addControllerListener(ControllerListener arg0), con arg0=this, permite que la clase ImagePanel atienda a los eventos del Processor.
    - Funciones para inicializar el Processor:
      - void javax.media.Processor.configure()
      - void javax.media.Processor.realize()
      - void javax.media.Processor.prefetch()
      - void javax.media.Processor.start()

    - DataSource javax.media.Processor.getDataOutput(), obtiene un objeto DataSource a partir del Processor.

  - Descripción: Esta función abre el archivo de video con su MediaLocator, y crea un objeto Processor para comenzar reproducirlo. A partir del Processor se crea un DataSource, que se pasa a la clase interna DataSourceHandler, para obtener frames del flujo de vídeo.

- **useFrameData**

  - Programa: SparksAnalyzer
  - Clase: ImagePanel
  - Declaración: private void useFrameData(Buffer inBuffer)
  - Parámetros:
    - Buffer inBuffer: el buffer de datos del DataSourceHandler que gestiona el flujo de vídeo.
  - Funciones de librería utilizadas:
    - Object javax.media.Buffer.getData(), para obtener los datos del Buffer.
  - Descripción: Esta función recibe un frame del Buffer del DataSource. Si esa frame debe ser analizada, según la tasa de análisis de frames definida por el usuario, continúa. Obtiene una imagen en forma de matriz de enteros y ordena su anñalisis llamando a la función doProcessing. Si se activó la opción de salvar las imágenes analizadas, se procede a llamar a las funciones encargadas.

- **doProcessing**

  - Programa: SparksAnalyzer
  - Clase: ImagePanel
  - Declaración: private void doProcessing()
  - Funciones de librería utilizadas:
    - BufferedImage javax.media.jai.PlanarImage.getAsBufferedImage(), convierte una PlanarImage en BufferedImage. Ésta se convierte a Image con un cast, para proporcionarla a los algoritmos que operan con este tipo.

- Descripción: Función que llama a todos los métodos implicados en el procesamiento del frame (algunos de ellos definidos en otras clases).
  La secuencia es la siguiente:
  1. Se sustrae a la imagen actual el background, de esta forma en la imagen sólo quedan las chispas.
  2. Se extraen los datos de la imagen RGB.
  3. Se extraen los datos de la imagen monocroma (este proceso requiere una transformación previa).
  4. Se repite el procesamiento anterior para una imagen con un umbral de intensidad preestablecido por el usuario.
  5. Si existen chispas en la imagen, según el umbral de intensidad específico para ello definido por el usuario:
     - Se calcula la transformada de Hough y todos los datos relacionados.
     - Se calcula el flujo óptico (Optical Flow).

- **tidyClose**
  - Programa: SparksAnalyzer
  - Clase: ImagePanel
  - Declaración: public void tidyClose()
  - Descripción: Cierra ordenadamente el Processor, el DataSourceHandler y el fichero de salida cuando finaliza el flujo de vídeo.

- **outdataBuffer**
  - Programa: SparksAnalyzer
  - Clase: ImagePanel
  - Declaración: public void outdataBuffer(int[] outpix, byte[] inData)
  - Parámetros:
    - int[] outpix: imagen resultante en forma de matriz de enteros.
    - byte[] inData: buffer del que se van a extraer los datos.
  - Descripción: Función auxiliar que obtiene una imagen en forma de matriz de enteros a partir de los datos recogidos en un Buffer.

- **performBackground y performLoadFile**
  - Programa: SparksSolver

- Clase: CommandPanel
- Declaración: private void performBackground(), private void performLoadFile ()
- Descripción: Son las funciones asociadas a la pulsación de los botones de apertura de imágenes desde archivo, para la imagen de fondo y la imagen fuente a analizar.
  Implementan el diálogo para abrir los ficheros mediante el componente javax.swing.JFileChooser. Cuando se han seleccionado, se envían las rutas a ImagePanel, para abrir las imágenes en las funciones setBGImage y setImage descritas anteriormente.
  performBackground da paso a la posibilidad de seleccionar la imagen fuente, y performLoadFile lanza el procesamiento de la imagen fuente.

- **performLoadVideo**
  - Programa: SparksAnalyzer
  - Clase: CommandPanel
  - Declaración: private void performLoadVideo()
  - Descripción: Función asociada a la pulsación del botón de apertura de vídeo desde archivo.
    Implementa el diálogo para abrir el ficheros mediante el componente javax.swing.JFileChooser. Cuando se ha seleccionado, se envía la ruta a ImagePanel, para abrir el vídeo en la función doAnalyze descrita anteriormente.
    Además se ordena la creación del fichero de salida asociado al vídeo que se va a procesar.

- **prepareRGB y prepareGray**
  - Programa: SparksSolver
  - Clase: CommandPanel
  - Declaración: private void prepareRGB, private void prepareGray.
  - Descripción: Funciones asociadas a la pulsación de los botones para analizar la imagen en color (RGB) y en blanco y negro, respectivamente. Ordenan el comienzo del análisis. Este análisis es el básico, llevado a cabo por la clase DataPanel.
    En el caso de prepareGray, se realiza previamente la conversión de la imagen de RGB a monocroma, mediante los métodos de la clase GrayScale.

- **startOperation**
  - Programa: SparksSolver y SparksAnalyzer
  - Clase: Threshold
  - Declaración: public void startOperation()
  - Tipo de datos de librería utilizado:
    - javax.media.jai.ROI y PlanarImage
  - Funciones de librería utilizadas:
    - PlanarImage javax.media.jai.ROI.getAsImage(), convierte una ROI en una PlanarImage.
  - Descripción: Este método realiza la umbralización en intensidad. El umbral ha sido fijado previamente. La imagen que se está analizando se convierte a monocroma con los métodos de la clase GrayScale. A continuación, la función obtainROI(PlanarImage im, int thrvalue) crea un objeto de tipo ROI (Region Of Interest) mediante la declaración "new ROI(im, value)". Un objeto ROI es una imagen binaria con píxeles a uno correspondientes a aquellos de la imagen original que superen el umbral.
    Una vez realizada la umbralización, se procede a llamar a los métodos encargados de realizar un análisis básico de la imagen resultante (histograma de bandas, valores máximos, medios y mínimos) y de calcular el tamaño de la región de interés. Estos métodos pertenecen a la clase DataPanel y se describirán a continuación.
    En la figura 3.17 se ilustra el funcionamiento de la umbralización en intensidad. Se ha utilizado un valor de umbral igual a 100 sobre la imagen de la figura 3.16. En DataPanel se ha recuadrado en rojo la información obtenida del análisis de la imagen umbralizada.

- **extractMean**
  - Programa: SparksSolver y SparksAnalyzer
  - Clase: DataPanel
  - Declaración: private void extractMean(PlanarImage source, ROI roi)
  - Parámetros:
    - PlanarImage source: la imagen que se va a anlizar.

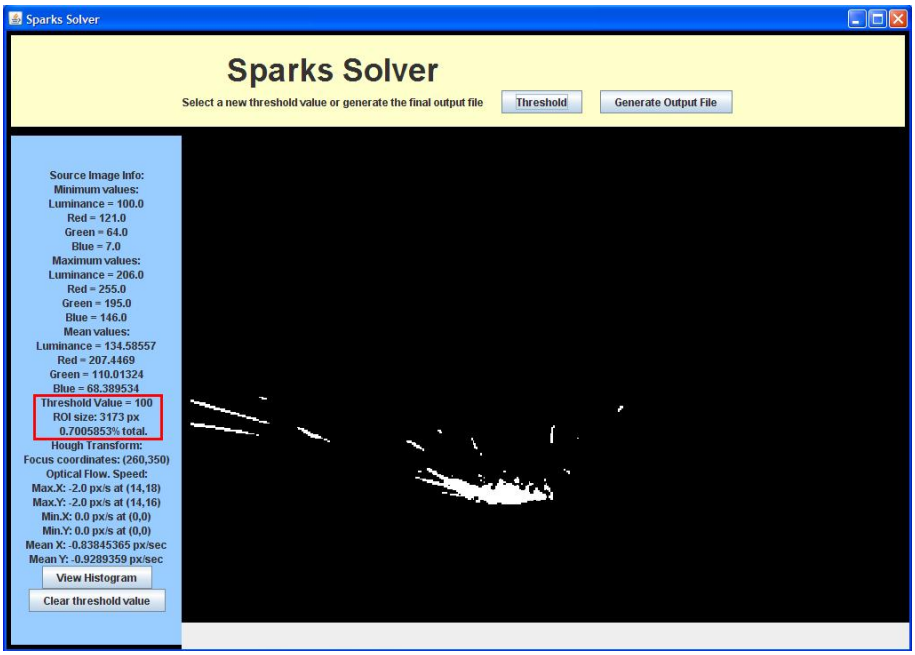

Figura 3.17: Umbralización en intensidad en la función startOperation

- ROI roi: la región de interés dentro de la imagen. Si vale null, se analiza toda la imagen.

- Funciones de librería utilizadas:
  - RenderedOp javax.media.jai.JAI.create(String arg0, ParameterBlock arg1, RenderingHints arg2), en la forma JAI.create("mean", pb, null).
  - Object javax.media.jai.PlanarImage.getProperty(String arg0), con arg0="mean", para obtener los datos de media a partir del RenderedOp resultante de la función anterior.
- Descripción: Esta función obtiene los valores medios de todas las bandas. Contempla la existencia de una región de interés. Los resultados se muestran visualmente en DataPanel.

- **extractExtrema**
  - Programa: SparksSolver y SparksAnalyzer
  - Clase: DataPanel
  - Declaración: private void extractExtrema(PlanarImage source, ROI roi)

- Parámetros:
  - PlanarImage source: la imagen que se va a anlizar.
  - ROI roi: la región de interés dentro de la imagen. Si vale null, se analiza toda la imagen.
- Funciones de librería utilizadas:
  - RenderedOp javax.media.jai.JAI.create(String arg0, ParameterBlock arg1), en la forma JAI.create(.ªextrema", pb).
  - Object javax.media.jai.RenderedOp.getProperty(String arg0), con arg0="extrema", para obtener los valores máximos y mínimos de las bandas de la imagen a partir del RenderedOp resultante de la función anterior.
- Descripción: Extrae los valores extremos de todas las bandas de la imagen. Contempla la existencia de una región de interés. Los resultados se muestran visualmente en DataPanel.

- **getROISize**
  - Programa: SparksSolver y SparksAnalyzer
  - Clase: DataPanel
  - Declaración: private void getROISize(PlanarImage src, ROI roi)
  - Parámetros:
    - PlanarImage src: la imagen a la que pertenece la ROI.
    - ROI roi: región de interés que se pretende medir.
  - Funciones de librería utilizadas:
    - PlanarImage javax.media.jai.ROI.getAsImage(), devuelve una imagen binaria a partir de un objeto ROI.
  - Descripción: Función que calcula el tamaño de la ROI según el valor del umbral de intensidad. El cálculo se fundamenta en el hecho de que la imagen que se obtiene a partir de una ROI con la función de librería ROI.getAsImage() está compuesta por píxeles de valores 0,1. Entonces se cuentan los píxeles a 1 eficientemente con un objeto histogram definido en JAI.
    En el recuadro rojo de la figura 3.17 se muestra un ejemplo de cálculo.

- **extractHistogram**
  - Programa: SparksSolver y SparksAnalyzer

- Clase: DisplayHistogram
- Declaración: public int[][] extractHistogram(PlanarImage source, ROI roi)
- Parámetros:
  - PlanarImage source: imagen de la que se va a obtener el histograma.
  - ROI roi: región de interés.
- Tipo de datos de librería utilizado:
  - javax.media.jai.Histogram
- Funciones de librería utilizadas:
  - RenderedOp javax.media.jai.JAI.create(String arg0, ParameterBlock arg1, RenderingHints arg2), en la forma (PlanarImage)JAI.create("histogram", pb, null).
  - Object javax.media.jai.PlanarImage.getProperty(String arg0), con arg0="histogram".
- Descripción: Esta función obtiene el array con los datos del histograma de las bandas de la imagen utilizando los métodos de la librería JAI.

- **toOutHist**
  - Programa: SparksSolver y SparksAnalyzer
  - Clase: DisplayHistogram
  - Declaración: public void toOutHist()
  - Descripción: Envía los datos del histograma al fichero de salida.

- **plotRGBHist y plotGrayHist**
  - Programa: SparksSolver
  - Clase: DisplayHistogram
  - Declaración: private void plotRGBHist(int[][] data), private void plotGrayHist(int[] grayarray)
  - Parámetros:
    - int[][] data: array con los histogramas de tres bandas.
    - int[] grayarray: array con el histograma de una banda.

- Descripción: Estas dos funciones ordenan la construcción de un gráfico para cada histograma del array que se les pasa como parámetro. Los gráficos se crean con objetos HistPlot, mediante los métodos que se describen seguidamente.

- **paintComponent**

  - Programa: SparksSolver
  - Clase: HistPlot
  - Declaración: public void paintComponent(Graphics g)
  - Parámetros:
    - Graphics g: La clase Graphics es la clase base abstracta para todos los contextos gráficos que permiten a una aplicación dibujar en componentes. Un objeto Graphics encapsula información necesaria para las operaciones de presentación que soporta Java.
  - Descripción: Función que crea un gráfico con la representación del histograma a partir de un vector de datos.

- **OFFrame.processImage**

  - Programa: SparksSolver
  - Clase: OFFrame
  - Declaración: public void processImage()
  - Funciones de librería utilizadas:
    - java.awt.image.PixelGrabber.PixelGrabber(Image img, int x, int y, int w, int h, int[] pix, int off, int scansize) Crea un objeto PixelGrabber para obtener una matriz de una sección rectangular de píxeles (x, y, w, h) desde la imagen especificada al vector dado. Los píxeles se almacenan en el vector en el ColorModel RGB.
  - Descripción: Esta función rige el proceso del cálculo del flujo óptico entre dos imágenes. En primer lugar, las dos imágenes se trasladan a sendas matrices mediante un objeto PixelGrabber. A continuación se pasan los parámetros y se llama a la función de la clase OpticalFlow que calcula el flujo óptico. El resultado, una imagen con los vectores de velocidad, se representa en la ventana que es el propio componente OFFrame. Un ejemplo se muestra en la figura 3.13 en la página 58.

- **OpticalFlow.processImage**
  - Programa: SparksAnalyzer
  - Clase: OpticalFlow
  - Declaración: public void processImage()
  - Igual a SparksSolver.OFFrame.processImage, descrita anteriormente, salvo que no se representa visualmente el flujo óptico.
- **OpticalFlow.process**
  - Programa: SparksSolver y SparksAnalyzer
  - Clase: OpticalFlow
  - Declaración: public int[] process()
  - Descripción: Computa el flujo óptico entre dos imágenes en forma de matriz. Como resultado se crean dos vectores de velocidades horizontales y verticales.
- **speedData**
  - Programa: SparksSolver y SparksAnalyzer
  - Clase: OpticalFlow
  - Declaración: private void speedData()
  - Descripción: A partir de los vectores de velocidades horizontales y verticales obtenidos de OpticalFlow.process(), se obtienen los valores máximos, mínimos y medios de velocidad, mediante las funciones SpeedMax(), SpeedMin() y speedMean(), respectivamente, y las coordenadas de los píxeles en los que se producen.
- **speedMax**
  - Programa: SparksSolver y SparksAnalyzer
  - Clase: OpticalFlow
  - Declaración: private void speedMax()
  - Descripción: Esta función obtiene datos de velocidad máxima. Cuando se ha identificado el píxel de máxima velocidad, se procede a integrar este resultado con el foco obtenido tras la aplicación de la transformada de Hough. Esto permite tener una idea de la exactitud en la estimación del foco. Para ello se llama a todas las funciones necesarias para procesar y presentar los datos.

- **line**
  - Programa: SparksSolver y SparksAnalyzer
  - Clase: OpticalFlow
  - Declaración: public void line(int x0, int y0, int x1, int y1, String color)
  - Parámetros:
    - int x0, int y0: coordenadas del punto origen.
    - int x1, int y1: coordenadas del punto destino.
    - String color: color de la línea.
  - Descripción: Función que permite dibujar una línea entre los puntos origen y destino. Se utiliza para construir un gráfico representativo del flujo óptico entre dos imágenes, a partir de los vectores de velocidades resultantes de aplicar la función OpticalFlow.process() descrita anteriormente. También se utiliza para dibujar una línea que una el punto de mayor velocidad, calculado en OpticalFlow.speedMax(), con el foco de las chispas, calculado en la clase FindFocus.

- **compTimeDif**
  - Programa: SparksAnalyzer
  - Clase: OpticalFlow
  - Declaración: private float compTimeDif()
  - Descripción: Función que computa la diferencia de tiempo entre dos frames a partir de sus marcas temporales. Con este dato se puede expresar la velocidad en las funciones SpeedMax(), SpeedMin() y speedMean() en píxeles por segundo.

- **HoughFrame.processImage**
  - Programa: SparksSolver
  - Clase: HoughFrame
  - Declaración: public void processImage()
  - Funciones de librería utilizadas:
    - java.awt.image.PixelGrabber.PixelGrabber(Image img, int x, int y, int w, int h, int[] pix, int off, int scansize)

- void javax.swing.JLabel.setIcon(Icon icon), permite mostrar una Image en una JLabel.

- Descripción: Esta función es el método principal para el cálculo de la transformada de Hough. Llama a todas las funciones implicadas en el proceso, organizando el paso de parámetros. Los pasos que se siguen son los siguientes:

  1. Detección de bordes mediante el operador de Sobel, con la clase Sobel.
  2. Umbralización con histéresis, con umbrales dinámicos calculados previamente. De esta tarea se encarga la clase HystThreshold.
  3. Cálculo de la transformada de Hough de la imagen binaria, con la clase HoughTransform.
  4. Dibujar los resultados sobre la imagen: las líneas obtenidas y el foco identificado.
  5. Representar la imagen resultante en el JFrame HoughFrame.

  El resultado del algortimo aplicado puede observarse en la figura 3.18. Las líneas obtenidas aparecen en color rojo, y la zona del foco recuadrada en azul.

- **HoughAlgorithm.processImage**

  - Programa: SparksAnalyzer
  - Clase: HoughAlgorithm
  - Declaración: public void processImage()
  - Esta función es igual a HoughFrame.processImage que se acaba de describir, salvo que la imagen resultante de todo el procesamiento mediante la transformada de Hough se presenta en ImagePanel.

- **produceImage**

  - Programa: SparksSolver y SparksAnalyzer
  - Clase: HoughFrame en SparksSolver y HoughAlgorithm en SparksAnalyzer
  - Declaración: private int[] produceImage(int[] input)
  - Parámetros:
    - int[] input: matriz imagen binaria cuyos píxeles blanco son las líneas obtenidas tras la aplicación de la transformada de Hough.

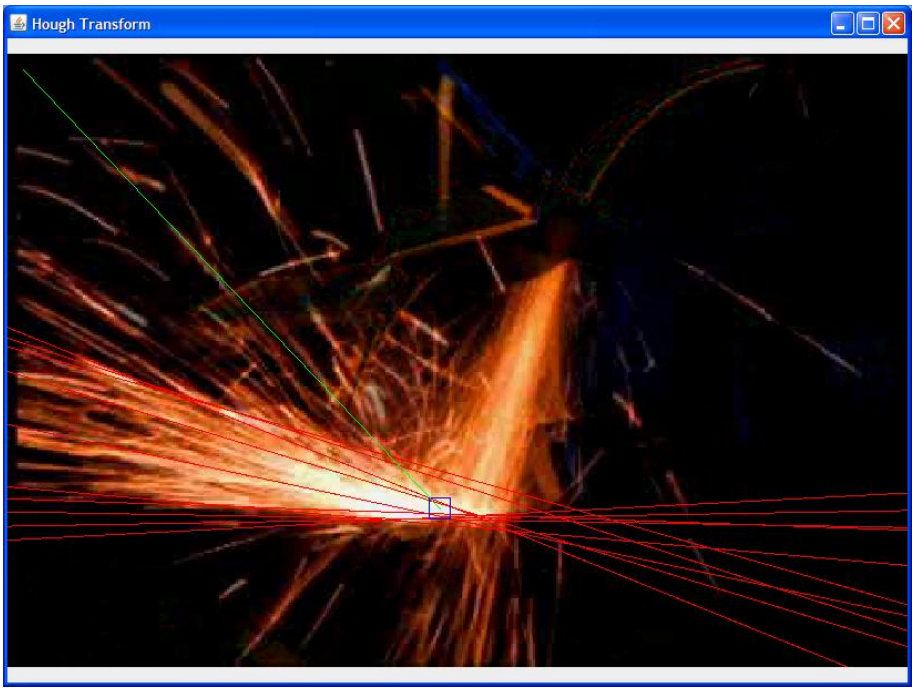

Figura 3.18: Resultado de la transformada de Hough, líneas obtenidas y foco identificado.

- Funciones de librería utilizadas:
  - java.awt.image.PixelGrabber.PixelGrabber(Image img, int x, int y, int w, int h, int[] pix, int off, int scansize)
- Descripción: Esta función dibuja los resultados de la transformada de Hough sobre la imagen fuente original - sin la sustracción del fondo -. Estos resultados son las líneas detectadas en la imagen, obtenidas tras la función processImage que se acaba de describir, y el foco de las chispas, identificado por la clase FindFocus.

- **joinHoughOF**
  - Programa: SparksSolver y SparksAnalyzer
  - Clase: HoughFrame en SparksSolver y HoughAlgorithm en SparksAnalyzer
  - Declaración: public void joinHoughOF(int x0, int y0, int x1, int y1, String color)
  - Parámetros:
    - int x0, int y0: coordenadas del punto origen.
    - int x1, int y1: coordenadas del punto destino.
    - String color: color de la línea que se va a dibujar.
  - Funciones de librería utilizadas:
    - java.awt.image.PixelGrabber.PixelGrabber(Image img, int x, int y, int w, int h, int[] pix, int off, int scansize)
  - Descripción: Esta es la primera de las funciones que integran los resultados de la transformada de Hough y el flujo óptico. Esta función dibuja sobre la imagen creada en produceImage (que se acaba de describir) la línea que une el punto de máxima velocidad, calculado en OpticalFlow.speedMax(), con el foco de las chispas, calculado en la clase FindFocus. Actualiza la imagen resultante y ordena su presentación en el lugar que le corresponde según la aplicación.

- **Sobel.process**
  - Programa: SparksSolver y SparksAnalyzer
  - Clase: Sobel
  - Declaración: public int[] process()

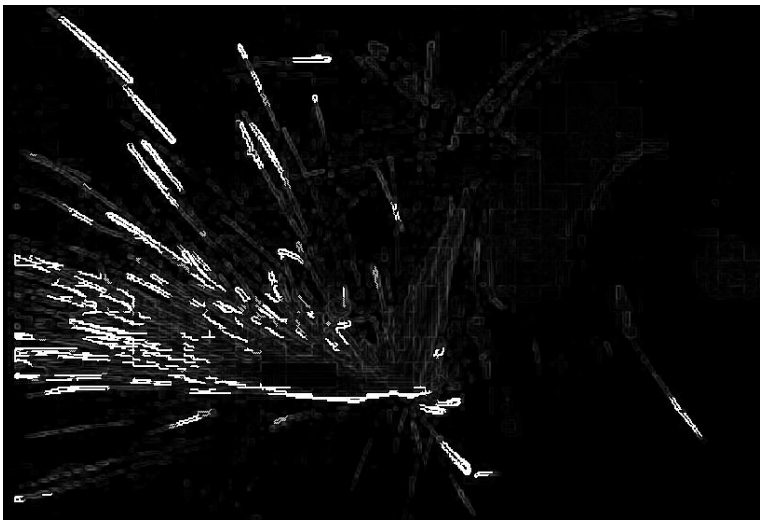

Figura 3.19: Resultado de la detección de bordes mediante el operador de Sobel

- Descripción: Función que realiza la detección de los bordes de una imagen mediante la aplicación del operador de Sobel. Elabora una imagen monocroma con el resultado. En la figura 3.19 se muestra un ejemplo de resultado obtenido de su aplicación.

- **findThresholds**

  - Programa: SparksSolver y SparksAnalyzer
  - Clase: Sobel
  - Declaración: private void findThresholds(int[] histogram)
  - Parámetros:
    - int[] histogram: es el histograma de la imagen monocroma resultado de la función sobel.process().
  - Descripción: Esta función se calcula los umbrales de binarización de la imagen de bordes resultado de Sobel.process() a partir de su histograma. Los umbrales se asignan de forma dinámica mediante una tabla, y se envían a HoughFrame para que los utilice la clase HystThreshold para binarizar la imagen de bordes.

- **HystThreshold.process**

  - Programa: SparksSolver y SparksAnalyzer

- Clase: HystThreshold
- Declaración: public int[] process()
- Descripción: Realiza la binarización con histéresis de umbral de la imagen de bordes generada por Sobel.process() utilizando los umbrales calculados con findThresholds.

- **HoughTransform.process**
  - Programa: SparksSolver y SparksAnalyzer
  - Clase: HoughTransform
  - Declaración: public int[] process()
  - Descripción: Esta función realiza la transformada de Hough de líneas en coordenadas polares y coordina el proceso de la devolución de resultados. Llama a la función auxiliar findMaxima y a finfFocus.doFindFocus. Devuelve una imagen con las líneas obtenidas.

- **findMaxima**
  - Programa: SparksSolver y SparksAnalyzer
  - Clase: HoughTransform
  - Declaración: private void findMaxima()
  - Descripción: Función para obtener las líneas solicitadas por el usuario. A partir del acumulador elaborado en HoughTransform.process se identifican los máximos de incidencia de líneas que pertenecen a clusters independientes. La separación entre clusters se establece con el parámetro 'distance', también configurable.

- **drawPolarLine**
  - Programa: SparksSolver y SparksAnalyzer
  - Clase: HoughTransform
  - Declaración: private void drawPolarLine(int value, int r, int theta)
  - Parámetros:
    - int value: incidencia de la línea de coordenadas (r,theta) en el acumulador de la transformada de Hough.
    - int r: módulo, coordenada polar.
    - int theta: ángulo, coordenada polar.

- Descripción: Método para dibujar líneas según sus coordenadas polares. El resultado es una imagen que sólo contiene las líneas identificadas.
  Se crea un acumulador de incidencias de puntos de las líneas para la identificación posterior del foco de las chispas.

- **checkFocus**
  - Programa: SparksSolver y SparksAnalyzer
  - Clase: HoughTransform
  - Declaración: public boolean checkFocus(int[] polars)
  - Parámetros:
    - int[] polars: coordenadas polares de la línea que une el punto de máxima velocidad, calculado en OpticalFlow.speedMax(), con el foco de las chispas, calculado en la clase FindFocus.
  - Descripción: Esta función participa en la integración de los resultados de la transformada de Hough y el flujo óptico. Comprueba si la identificación del foco de las chispas es correcta, contrastando los parámetros de las líneas obtenidas con la transformada de Hough, con los de la línea que une el punto de máxima velocidad con el foco de las chispas.

- **doFindFocus**
  - Programa: SparksSolver y SparksAnalyzer
  - Clase: FindFocus
  - Declaración: public void doFindFocus(int width,int height)
  - Parámetros:
    - int width, height: dimensiones de la imagen que se está analizando.
  - Descripción: Esta función determina la zona de la imagen en la que se encuentra el foco de las chispas. Para ello parte de un acumulador de incidencias de puntos de líneas, elaborado en la función drawPolarLine.
    Como las líneas de corresponden con las trayectorias que siguen las chispas, la zona por la que pasen más líneas será aquella correspondiente al foco, de ahí que se utilice este acumulador.
    La imagen se divide con una cuadrícula creada según el parámetro de configuración “tamaño de la zona del foco”. La cuadrícula

con mayor incidencia de puntos de líneas será identificada como la región del foco.

- **toGrayScale**
  - Programa: SparksSolver y SparksAnalyzer
  - Clase: GrayScale
  - Declaración: public PlanarImage toGrayScale(PlanarImage im)
  - Parámetros:
    - PlanarImage im: la imagen RGB que se va a convertir en monocroma.
  - Funciones de librería utilizadas:
    - RenderedOp javax.media.jai.JAI.create(String arg0, ParameterBlock arg1, RenderingHints arg2), en la forma (PlanarImage)JAI.create("BandCombine", pb, null)
  - Descripción: Función que convierte una imagen RGB en monocroma. Los pesos utilizados son (R,G,B)=(0.3,0.59,0.11).