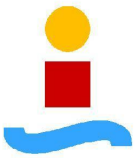

Universidad de Sevilla

Escuela Superior de Ingenieros

Departamento de Ingeniería de Sistemas y Automática

Técnicas para el Rectificado Robotizado de Metales Basadas en el Procesamiento de Imágenes

Proyecto Fin de Carrera
Ingeniería de Telecomunicación

Autora:
Julia de la Torre Lara
Tutor:
Dr. Eduardo Fernández Camacho

Sevilla, Septiembre de 2007

A mi familia, mis amigos y mi tutor

Agradecimientos

Me gustaría agradecer y dedicar esta memoria en primer lugar a mi familia, por el apoyo que me han prestado en todas las etapas del desarrollo de este proyecto. A mis amigos, en especial a Paula, por las veces que no he podido estar con ellos, y por haberse interesado por mi trabajo. A Javier, por su apoyo, sobre todo en la recta final del proyecto.

Agradezco a mi tutor Eduardo Fernández Camacho la oportunidad de desarrollar este proyecto, de aprender de sus conocimientos y que me abriera las puertas del Departamento de Ingeniería de Sistemas y Automática hace ya más de dos años.

Gracias a todos los que me han ayudado con sus consejos, sus críticas constructivas y sus ideas. En particular a mis compañeros del laboratorio en el departamento, que siempre me han echado una mano cuando lo he necesitado, y de los que he aprendido mucho.
A Beatriz Pontes por enseñarme a programar.
A Manuel Gil Ortega por su ayuda.
A los profesores José Antonio Rodríguez Ortiz y Juan Manuel Montes Martos por sus aclaraciones sobre el tratamiento del acero y por proporcionarme vídeos con los que trabajar.

Gracias a todos los que directa o indirectamente han contribuido al desarrollo de este proyecto.

Índice general

Índice de figuras

Capítulo 1

Motivación y Objetivos

En este capítulo se introducirá al lector al contenido proyecto fin de carrera desarrollado, planteando los motivos que llevaron a su realización, y los objetivos que se ha pretendido conseguir.
Seguidamente se explicará el contenido de esta memoria.

1.1. Motivación

El rectificado es el proceso en el que se elimina metal de la superficie de una pieza previamente mecanizada para lograr el acabado deseado. La herramienta con la que se lleva a cabo es la muela abrasiva. En el transcurso del proceso, se desprenden partículas de metal incandescente en forma de chispas (figura 1.1).

El rectificado manual frecuentemente implica un trabajo monótono en un ambiente ruidoso. Los operarios que lo llevan a cabo deben utilizar un equipo protector compuesto por gafas, guantes, etc. Esta circunstancia junto con el hecho de que prácticamente todos los productos que se fabrican están directa o indirectamente relacionados con el rectificado, ha llevado a la automatización de este proceso.

En las primeras aplicaciones industriales, el rectificado llevado a cabo por robots se resolvió utilizando una rectificadora acoplada al efector final del brazo robótico, mediante un mecanismo de amortiguación. La fuerza de contacto deseada entre la herramienta y la superficie de trabajo se obtenía

Figura 1.1: Rectificado de metal, herramienta y formación de chispas.

programando la posición del robot para que estuviera de alguna manera 'debajo'de la superficie de trabajo. De aquí surgió la dificultad de controlar la profundidad del corte y lograr las condiciones óptimas para el proceso, ya que cambios en la geometría de la pieza y el desgaste de la muela de la herramienta producían cambios en las fuerzas de contacto. Además, la geometría de la pieza debía medirse regularmente, lo cual era bastante tedioso. En consecuencia, existía la necesidad de un sistema flexible e inteligente para controlar el robot y ocuparse de los requerimientos del proceso.

En la actualidad el rectificado robotizado de metal se lleva a cabo mediante control de posición, fuerza y velocidad del efector, para poder controlar los parámetros característicos del proceso. Sin embargo esta estrategia tampoco es óptima.
Debido a las peculiaridades del proceso de rectificado, convendría desarrollar una estrategia más eficiente que controlara su resultado directamente. Esto es:

- La cantidad de material que va a ser arrancada.
- El acabado de la superficie.
- El transcurso óptimo del proceso.
- Supervisar que la superficie de trabajo de la pieza no se sobrecalienta.
- El desgaste de la muela.
- Con respecto a las chispas, saber cuándo se está rectificando la superficie deseada y no otra.

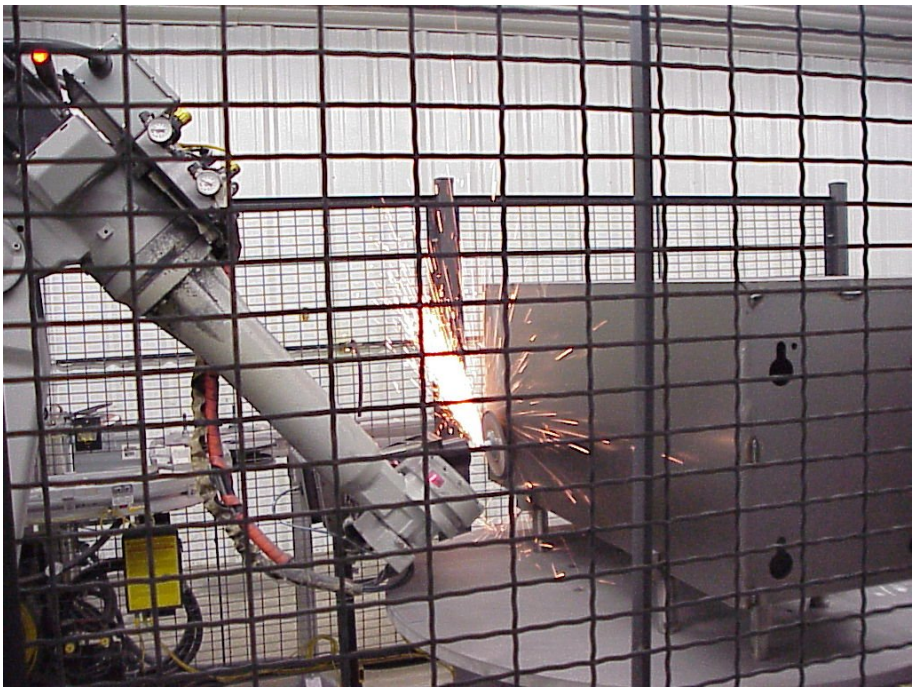

Figura 1.2: Rectificado robotizado de metal.

Conseguir esto equivaldría a la optimización del proceso de rectificado en cuanto a tiempo, desgaste de herramientas, economía y calidad.

En los entornos de trabajo existe incertidumbre en la localización de la superficie de trabajo, en la cantidad de material que se va a eliminar y en el comportamiento de las herramientas - incluso aunque sean iguales -. Debido a esto existe la necesidad de una solución sofisticada que integre diferentes sensores para observar los parámetros críticos. Una vez hecho esto, el siguiente paso es diseñar y generar la actuación del robot en tiempo real para mantener el proceso en condiciones óptimas.

1.2. Objetivos

Los pasos a seguir para optimizar el proceso de rectificado de metal, según lo descrito en el apartado anterior, son los siguientes:

1. Estudiar el proceso de rectificado y sus parámetros críticos.

2. Analizar los diferentes sensores que se pueden aplicar y averiguar qué tipo de información pueden proporcionar.

3. Realizar rectificado manual para recoger datos y aplicar algoritmos para extraer información.

4. Una vez que se tienen todos los datos, desarrollar e implementar una estrategia de control.

El desarrollo de este proyecto cubre los tres primeros pasos desde el campo de la visión artificial, y constituye el inicio de una línea de investigación en la fusión de sensores aplicables al rectificado de metal.
Los objetivos del proyecto, referidos a los pasos descritos anteriormente, son:

1. Estudiar previamente las características que conviene y es posible buscar por medio de la visión.
   A través de un estudio del proceso de rectificado en su conjunto y de numerosas imágenes, se han definido unos objetivos en cuanto a características de interés y forma de observarlas.

2. Analizar la información que puede extraerse de las imágenes que captan el proceso de rectificado.
   A partir de lo aprendido en el paso anterior, se ha estudiado la viabilidad de extraer las características de interés a partir de las imágenes. Para ello se ha desarrollado una aplicación de visión artificial para estudiar las características visuales de estas imágenes.

3. Analizar grandes cantidades de imágenes para recoger datos significativos.
   Una vez estudiadas en detalle imágenes de rectificado, se analizan vídeos completos, para recoger mayores cantidades de datos. Para esto se ha desarrollado una segunda aplicación que procesa vídeos, y cuya funcionalidad mejora si se complementa con la anterior.

El proyecto fin de carrera se ha concebido como una investigación en una línea inexplorada hasta el momento. Se han desarrollado dos herramientas para el estudio del rectificado de metal desde el campo de la visión artificial. Estas herramientas permiten el análisis de todo tipo de imágenes en las que existan chispas de metal incandescente. El objetivo principal no ha sido tanto obtener un prototipo o resultados definitivos, como estudiar la viabilidad de la aplicación de la visión artificial a esta actividad, para conseguir mejorar la calidad de un proceso con tanta importancia en la industria.

1.3. Organización de la memoria

La memoria se ha estructurado en seis capítulos y cuatro apéndices. En el primer capítulo, en el que nos encontramos, se plantean los motivos que han llevado a la realización del proyecto, y los objetivos que se ha pretendido conseguir.

En el segundo capítulo se da una introducción teórica a los conocimientos y las técnicas que se han empleado en la realización del proyecto.

El tercer capítulo recoge el alcance del proyecto. Esto comprende el trabajo que se ha realizado partiendo del conocimiento teórico descrito en el capítulo anterior. Se detallan los componentes del sistema diseñado, como el entorno de programación, así como la implementación en las aplicaciones software desarrolladas de las técnicas estudiadas.

El siguiente capítulo es un manual de usuario. Se explica cómo instalar el entorno de ejecución de las aplicaciones desarrolladas en el proyecto. A continuación se explica paso a paso mediante dos ejemplos el funcionamiento de los dos programas.

En el quinto capítulo se muestran los resultados obtenidos en varios experimentos, y se comentan resultados generales obtenidos a partir del estudio de imágenes desarrollado.

Por último, el sexto capítulo recoge las conclusiones a las que se ha llegado durante el desarrollo de este proyecto. Se proponen también líneas de desarrollo para el futuro.

En los apéndices se proporciona información adicional. En el primero, se explica la instalación del entorno de desarrollo utilizado en el proyecto.

En el segundo se incluye una explicación detallada de las funciones de librería utilizadas.

En el tercero se analiza la estructura de los ficheros de salida de texto que proporcionan las aplicaciones. En ellos se recogen todos los datos de los análisis.

El último recoge el contenido del CD adjunto a esta memoria.

Capítulo 2

Introducción teórica

En este capítulo se va a proporcionar una introducción a los conocimientos teóricos que se han aplicado durante el desarrollo de este proyecto fin de carrera.

Como se va a poner de manifiesto, el proyecto abarca varias materias. En primer lugar se darán nociones del proceso de rectificado de metales llevados a cabo por robots.

Posteriormente nos centraremos en la visión artificial, sus fundamentos y las herramientas que se pueden emplear para desarrollar aplicaciones de procesamiento de imágenes.

A continuación se explicarán las características fundamentales de Java, el lenguaje de programación que se ha utilizado para desarrollar los programas realizados.

Por último se van a describir las técnicas empleadas en la realización del proyecto. Se explicará en detalle el fundamento teórico de los algoritmos implementados en las aplicaciones que se han desarrollado en este proyecto.

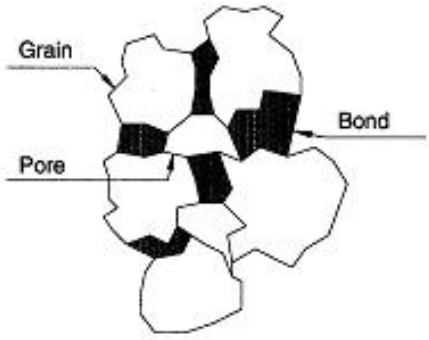

Figura 2.1: Componentes principales de la muela abrasiva.

2.1. Rectificado robotizado del metal

2.1.1. Componentes básicos

Las muelas para el rectificado se fabrican para muchos tipos de aplicaciones. Dependiendo de la pieza sobre la que van a trabajar tienen diferentes formas y tamaños. Las muelas abrasivas están compuestas principalmente de granos, aglutinante y poros (figura 2.1). Los granos son las herramientas cortantes que desprenden material en forma de esquirlas o chispas de la superficie de trabajo. El aglutinante pega y fija los granos, y mantiene unida la herramienta. Los poros proporcionan espacio para las partículas de material en la superficie de contacto entre la herramienta y la pieza.

Aunque se seleccione una herramienta adecuada para la pieza sobre la que se va a trabajar, no existen garantías de obtener un buen resultado. Como se muestra en la figura 2.2, la fuerza de empuje F_y y la velocidad de corte v_w se deben seleccionar dentro de unos límites. Se obtienen buenos resultados dentro de la zona amarilla de la figura. Cabe destacar que la pieza sobre la que se trabaja y la herramienta influyen en este tipo de curvas.

2.1.2. Estrategias para rectificado ligero

En muchas aplicaciones, el volumen de metal que se va a retirar de la pieza por unidad de superficie es comparativamente pequeño. Estas aplicaciones reciben el nombre de “rectificado ligero”. Es característico de estas

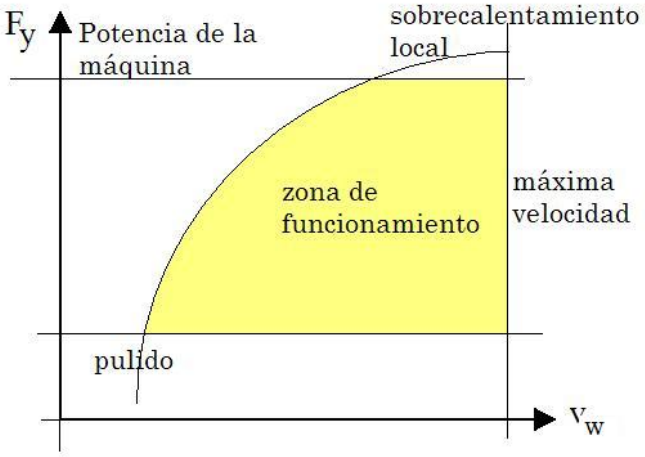

Figura 2.2: Limitaciones de los parámetros del proceso de rectificado

aplicaciones que proceso se lleve a cabo en una sola pasada.

Las fuerzas de contacto son pequeñas, lo que implica tasas de desgaste de herramienta bajas. Dependiendo de los requisitos de la superficie que se pretende obtener, en cuanto a acabado y geometría, y del grosor de la capa de material que se va a retirar, existen cuatro estrategias de operación posibles(figura 2.3):

- Posición programada, P_N, normal a la superficie de trabajo y velocidad de corte v_w constante. (Estrategia de control de posición).
- Fuerza de empuje F_y constante y velocidad de corte v_w constante. (Estrategia de control de fuerza).
- Posición programada P_N normal a la superficie de trabajo y velocidad de corte v_w variable controlada por las fuerzas del proceso. (Estrategia de control de velocidad).
- Posición programada P_N normal a la superficie de trabajo y velocidad de corte v_w variable controlada por las fuerzas del proceso, incluyendo repetición si v_w baja de un cierto umbral. (Estrategia de repetición).

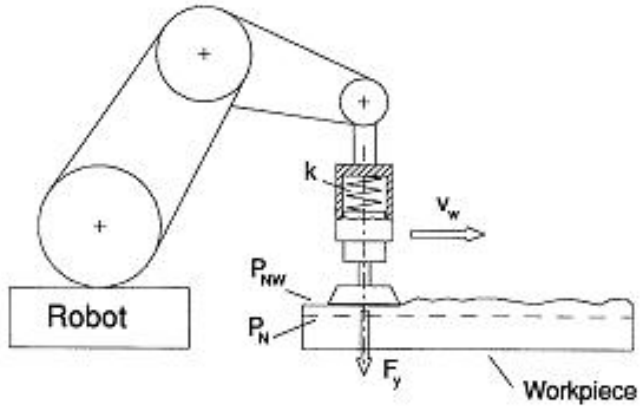

Figura 2.3: Parámetros principales del rectificado ligero.

2.1.3. Estrategias para rectificado grueso

En otras muchas aplicaciones, el volumen de metal que se retira de la pieza por unidad de superficie es grande. Estas aplicaciones sn las que realmente requieren robots industriales. Reciben el nombre de "rectificado grueso". En ellas el rectificado se realiza en varias pasadas. Además, con frecuencia la pieza de trabajo es pesada y difícil de posicionar adecuadamente. Para resolver estos problemas, se requiere un sistema de control más sofisticado.

Una solución apropiada consiste en separar la operación de rectificado en pasadas bastas y finas, y usar una estrategia basada en un modelo empírico del proceso. Las pasadas bastas eliminan la mayor parte del material, mientras que las finas se encargan de conseguir las especificaciones de geometría y acabado de la superficie. En la figura 2.4 se muestran los principales parámetros de este proceso.

2.2. Visión artificial

La Visión artificial, también conocida como Visión por Computador o Visión técnica, es un subcampo de la inteligencia artificial. Es la ciencia y la tecnología de las máquinas que 'ven'.

Como disciplina científica, trata de construir sistemas artificiales que obtengan información de imágenes. Los datos que componen la imagen pueden

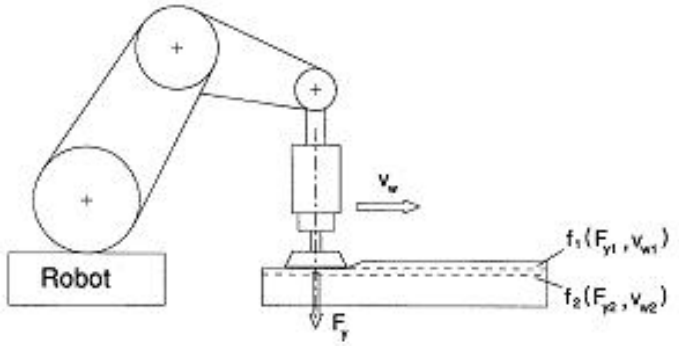

Figura 2.4: Parámetros principales del rectificado grueso.

ser de muchas formas, como una secuencia de vídeo, vistas desde múltiples cámaras, o datos multidimensionales procedentes de un escáner médico.

Como disciplina tecnológica, busca aplicar la teoría y los modelos de la visión por computador a la construcción de sistemas de visión por computador.

Así, el propósito de la visión artificial es programar un computador para que "entienda" una escena o las características de una imagen. La visión artificial consiste en la captación de imágenes y su posterior tratamiento a través técnicas de procesamiento de imagen avanzadas, permitiendo así poder intervenir o sobre un proceso o supervisarlo. Los objetivos típicos de la visión artificial incluyen:

- La detección, segmentación, localización y reconocimiento de ciertos objetos en imágenes (por ejemplo, caras humanas).
- La evaluación de los resultados (por ejemplo, segmentación y registro).
- Registro de diferentes imágenes de una misma escena u objeto, es decir, hacer concordar un mismo objeto en diversas imágenes.
- Seguimiento de un objeto en una secuencia de imágenes.
- Mapeo de una escena para generar un modelo tridimensional de la escena; tal modelo podría ser usado por un robot para navegar por la escena.
- Estimación de las posturas tridimensionales de humanos.

- Búsqueda de imágenes digitales por su contenido.

Existen múltiples aplicaciones de las técnicas de visión artificial, la mayoría ligadas a la automática, y es un campo en constante expansión en la actualidad. Muchos de los métodos y las aplicaciones se encuentran todavía en fase de investigación, pero cada vez hay más productos con salida comercial, como parte de sistemas más complejos que resuelven tareas complicadas. Ejemplos de esto son el tratamiento de imágenes médicas, el control de calidad o las medidas en procesos industriales, como es el caso que nos ocupa en este proyecto.

Un área de la visión por computador se centra en el desarrollo de software y hardware que permita reducir el coste computacional de los algoritmos clásicos para ganar velocidad sin perder calidad.

Procesamiento de imágenes. El procesamiento de imágenes es cualquier forma de procesamiento de información mediante algoritmos en un computador en la cual se analiza una imagen. El resultado no es necesariamente una imagen, puede ser un conjunto de propiedades de la imagen.

Este campo, junto con el procesamiento de señales, constituyen la base de la visión por computador, ya que además de adquirir imágenes, para poder "ver" hay que extraer información de ellas. La mayoría de las técnicas de procesamiento de imágenes implican el tratamiento de la imagen como una señal bidimensional, y la aplicación de técnicas de procesamiento de señales. Además, muchos métodos se basan en el estudio matemático de las propiedades de la imagen, como la geometría o la extracción de estadísticas.
Sin embargo, debido a la naturaleza específica de las imágenes existen muchos métodos desarrollados en el ámbito de la visión por computador que no tienen contrapartida en el procesamiento de señales monovariable. Estos métodos se caracterizan porque no son lineales, lo cual junto con la multidimensionalidad de la señal, define un área del procesamiento de señales como parte de la visión por computador.

Figura 2.5: Logotipo de Java.

2.3. Java

Java es un lenguaje de programación de gran expansión en la actualidad, sobre todo a raíz de la omnipresencia de internet en todos los aspectos de la vida laboral y cotidiana.

Es un lenguaje de programación orientado a objetos, desarrollado por James Gosling y sus compañeros de Sun Microsystems Inc. en 1991. Su lanzamiento se produjo en 1995. El lenguaje en sí mismo toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel como punteros.
A diferencia de los lenguajes de programación convencionales, que generalmente están diseñados para ser compilados a código nativo, Java se compila en un "bytecode" que se ejecuta (usando normalmente un compilador JIT, Just In Time) en una máquina virtual Java (Java Virtual Machine, JVM). Estas características técnicas se traducen en las dos principales propiedades de Java: portabilidad o independencia de la plataforma y seguridad.

Portabilidad o independencia de la plataforma La primera característica, la independencia de la plataforma, supone que programas escritos en el lenguaje Java pueden ejecutarse de igual forma en cualquier tipo de hardware y en múltiples sistemas operativos. Esto permite la difusión de programas por Internet y la ejecución en línea.

Se escribe un programa una vez y puede ejecutarse en cualquier dispositivo, tal como reza el axioma de Java, "Write once, run everywhere". Para ello, se compila el código fuente escrito en lenguaje Java, para generar un código conocido como "bytecode" (en concreto: Java bytecode). Este código se compone de instrucciones de lenguaje máquina simplificadas, específicas de la plataforma Java. Estas instrucciones están a medio camino entre el código fuente y el código máquina que entiende el dispositivo de destino.
El bytecode es ejecutado entonces en la máquina virtual (VM). La máquina virtual es un programa escrito en código nativo de la plataforma destino (que es el que entiende su hardware), que interpreta y ejecuta el código. Se debe tener presente que, aunque hay una etapa explícita de compilación, el bytecode generado es interpretado o convertido a instrucciones máquina del código nativo por el compilador JIT (Just In Time).

El concepto de independencia de la plataforma de Java cuenta con un gran éxito en las aplicaciones en el entorno del servidor, como los Servicios Web, los Servlets, los Java Beans, así como en sistemas empotrados basados en OSGi, usando entornos Java empotrados.

Seguridad La segunda característica, la seguridad, se refiere a que cuando se utiliza un navegador de internet compatibe con Java, es posible transferir de forma segura las applets (tipo de aplicación) de Java sin temor de ser infectados por un virus o a recibir intentos de acceso no autorizados. Java consigue esto limitando el programa Java a un entorno de ejecución propio, impidiendo que acceda a otras partes del equipo.

Gestión de la memoria En Java, el programador no tiene que preocuparse de liberar la memoria utilizada, ya que existe el llamado recolector automático de basura (o automatic garbage collector). El programador determina cuándo se crean los objetos, es el entorno en tiempo de ejecución de Java (Java runtime) el responsable de gestionar el ciclo de vida de los mismos. El programa, u otros objetos pueden tener localizado un objeto mediante una referencia (que desde un punto de vista de bajo nivel es una dirección de memoria). Cuando no quedan referencias a un objeto, el recolector de basura de Java lo borra, liberando así la memoria que ocupaba previniendo posibles fugas (o "memory leaks").

Aún así, es posible que se produzcan fugas de memoria si el código almacena referencias a objetos que ya no son necesarios. Esto puede dar problemas en tiempo de ejecución. La solución es sencilla: aunque el programador no

tiene la necesidad de gestionar la memoria, puede hacerlo. Si se tiene la certeza de que un objeto no se va a utilizar más, es posible acelerar su recogida asignándolo a 'null'.

El sistema de recogida de basura automática facilita la labor del programador, pero no le resta flexibilidad, ya que es posible realizarla de forma no automática cuando se requiera.

Rendimiento El rendimiento de una aplicación está determinado por multitud de factores, por lo que no es fácil hacer una comparación que resulte totalmente objetiva. En tiempo de ejecución, el rendimiento de una aplicación Java depende más de la eficiencia del compilador, o la JVM, que de las propiedades intrínsecas del lenguaje. El bytecode de Java puede ser interpretado en tiempo de ejecución por la máquina virtual, o bien compilado al cargarse el programa, o durante la propia ejecución, para generar código nativo que se ejecuta directamente sobre el hardware. Si es interpretado, será más lento que usando el código máquina intrínseco de la plataforma destino. Si es compilado, durante la carga inicial o la ejecución, la penalización está en el tiempo necesario para llevar a cabo la compilación.

El rendimiento de un compilador JIT y los compiladores nativos puede ser parecido, aunque la distinción no está clara en este punto. La compilación mediante el JIT puede consumir un tiempo apreciable, lo cual supone un inconveniente principalmente para aplicaciones de corta duración o con gran cantidad de código. Sin embargo, una vez compilado, el rendimiento del programa puede ser comparable al que consiguen compiladores nativos de la plataforma destino.

Evolución En los últimos años, el desarrollo de la tecnología Java ha sido enorme, y en la actualidad es una de las tecnologías más demandadas. Las deficiencias en su funcionamiento derivadas de ser un lenguaje interpretado en lugar de compilado, se están solucionando rápidamente, de forma que actualmente las prestaciones que ofrece son prácticamente iguales a las de otros lenguajes. A día de hoy, el lenguaje de programación Java ha sido totalmente mejorado, ampliado y probado por una comunidad activa de unos cuatro millones de desarrolladores de software.

2.4. Técnicas empleadas

2.4.1. Utilización de Java

A la hora de elegir el lenguaje de programación con el que se iban a desarrollar las aplicaciones del proyecto, se optó por Java. Las razones de esta elección son:

- El auge de la tecnología Java en numerosos sectores de la ingeniería. Esto hacía interesante profundizar en esta tecnología, ya que cada vez tiene más aplicaciones.
- La filosofía de orientación a objetos. Suponía un aspecto que no se había tratado hasta el momento durante la carrera.
- La portabilidad o independencia de la plataforma, que garantiza que las aplicaciones se pueden ejecutar en cualquier equipo.
- El procesamiento de imágenes en Java estaba menos desarrollado que en otros lenguajes. Esto suponía un reto y un aliciente.
- El rendimiento de Java mejora rápidamente, lo cual disminuía las reticencias respecto a su potencia.

Librerías de procesamiento de imágenes de Java

Las APIs (Aplication Programming Interface) de Java para el procesamiento de imágenes que se han utilizado son Java Advanced Imaging (JAI) y Java Media Framework (JMF). En el capítulo 3 se explica su funcionamiento, y el el Apéndice A se proporcionan las instrucciones para su instalación.

Las librerías proporcionan las funciones básicas para las aplicaciones de procesamiento de imágenes, y mejoran el rendimiento de los programas, ya que optimizan la estructura del lenguaje. La utilización de librerías se hace necesaria principalmente en operaciones de bajo nivel, que de otra forma consumirían muchos recursos y aumentarían el tiempo de ejecución.

2.4.2. Sustracción de imagen de fondo

En las aplicaciones que se han desarrollado, se ha implementado un algoritmo de sustracción del fondo de la imagen, como paso previo a algoritmos más complejos.

El funcionamiento del algoritmo es el siguiente: dadas la imagen a analizar y una imagen de fondo, se restan los píxeles de las dos imágenes, uno a uno y banda a banda. La sustracción de imágenes es una operación aritmética.

Por ejemplo, para imágenes RGB de 24 bits, cuyos píxeles son ternas de valores entre 0 y 255 tendríamos:

$$(\ 230,\ 200,\ 120) - (\ 210,\ 210,\ 50) = (\ 20,\ 0,\ 70)$$

Si el resultado es negativo en alguna banda, se satura a cero. Si a una imagen se le resta una imagen negra, se obtiene la imagen original.

Este paso previo a la aplicación del resto de los algoritmos diseñados mejora el funcionamiento de estos. Si la sustracción se lleva a cabo correctamente, es decir, con una imagen que recoja el fondo estático, se reduce enormemente la cantidad de elementos de la imagen. Sólo queda el elemento de interés, en nuestro caso las chispas, y no procesamos datos innecesarios. En la figura 2.6 se muestra un ejemplo de funcionamiento.

En caso de no disponer de una imagen de fondo, se puede obviar este paso seleccionando como fondo una imagen negra.

2.4.3. Estadísticas

A la hora de extraer patrones de comportamiento, resulta de utilidad disponer de estadísticas de la distribución de ciertas propiedades de las imágenes. Esto se lleva a cabo mediante operaciones estadísticas, que son:

- Obtener los valores medios de las bandas de una imagen: Se recorre cada banda y se calcula la media de los valores de los píxeles.
- Obtener los valores extremos de las bandas de una imagen: Se recorre cada banda buscando los valores máximos y mínimos.

Figura 2.6: Sustracción de dos imágenes.

- Obtener los histogramas de las bandas de una imagen. Para cada valor de intensidad de cada bande (de 0 a 255) se hace un recuento de la incidenca de píxeles. El resultado se representa en forma de gráfico, que contiene la distribución de los píxeles en los colores de la imagen.

Con estos datos, se tiene información completa de las distribuciones de color de la imagen.

Además de calcular todos estos datos para las bandas de una imagen en color, se pueden calcular para la luminancia transformando la imagen a blanco y negro (monocroma y con una sola banda).

2.4.4. Transformación de color a luminancia

Dada una imagen en color (normalmente de tres bandas), se trata de convertirla a una imagen de una sola banda. En esa banda estará la información de luminancia.

La conversión habitual de una imagen de tres bandas RGB (Red, Green, Blue) a luminancia se lleva a cabo aplicando la fórmula:

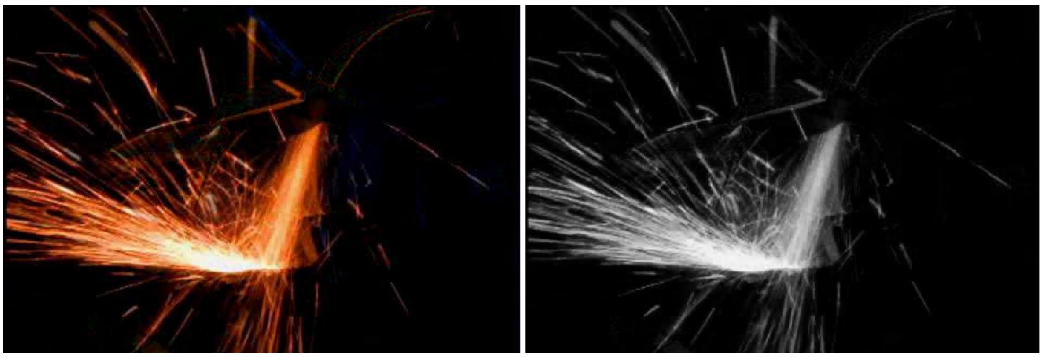

Figura 2.7: Conversión de color a blanco y negro.

$$Y = 0.3 \cdot R + 0.59 \cdot G + 0.11 \cdot B$$

El proceso consiste en aplicar esta fórmula a cada píxel de cada banda, y almacenar el resultado en un píxel, formando así la imagen en blanco y negro.

El resultado se muestra en la figura 2.7.

2.4.5. Umbralización en intensidad

En imágenes que presentan gran contraste de intensidad luminosa entre el objeto de interés y el fondo, es posible segmentar el objeto mediante la umbralización en intensidad.

Este algoritmo consiste en separar el objeto del fondo según la propiedad de intensidad. Se parte de la imagen en blanco y negro, resultado del algoritmo anterior. Para cada píxel, se realiza lo siguiente:

- Si el valor de intensidad de ese píxel es mayor que un cierto umbral, se pone el píxel en blanco. Es decir, se le asigna el valor 255 de intensidad.
- Si el valor es inferior al umbral, se pone el píxel en negro, a cero.

En nuestro caso concreto, estamos trabajando con imágenes que contienen chispas. Éstas se caracterizan por un nivel muy elevado de intensidad. Por tanto, si se elige adecuadamente el valor del umbral, en teoría es posible separarlas del fondo.

De la segmentación de las chispas se puede extraer un dato adicional: el tamaño del área de las chispas. Este cómputo del área se fundamenta en contar los píxeles que han quedado en blanco, y evaluar el tamaño con el resto de la imagen.
Se puede obtener el área en píxeles y en tanto por ciento de la imagen.
Así se conoce la cantidad de chispas que se están produciendo en un momento dado.

Umbralización dinámica

La umbralización dinámica es una variante de la anterior, en la que el umbral de intensidad no es fijo, sino que depende de las características de la imagen. En concreto depende de la distribución de intensidad, que se obtiene con el histograma.

La asignación del umbral se realiza por medio de una tabla, cuya entrada es la característica de distribución de intensidad, y la salida el valor del umbral.

Umbralización con histéresis

Esta técnica de umbralización puede emplear umbrales fijos o dinámicos. En las aplicaciones desarrolladas se ha implementado con umbrales dinámicos.

La umbralización con histéresis consiste en trabajar con dos umbrales, uno bastante elevado, y otro a una distancia relativamente significativa. La técnica es la siguiente:

1. Se comienza teniendo en cuenta sólo el umbral superior, y se va recorriendo la imagen.
2. Si el valor de intensidad de un píxel es mayor que el umbral superior, se pone el píxel en blanco, y se procede a evaluar a sus vecinos.
    - Si el valor de intensidad del píxel vecino es mayor que el umbral inferior ahora, se pone el píxel en blanco, y se repite este proceso iterativamente con sus vecinos.
    - Si el valor es inferior al umbral inferior, se pone el píxel en negro.

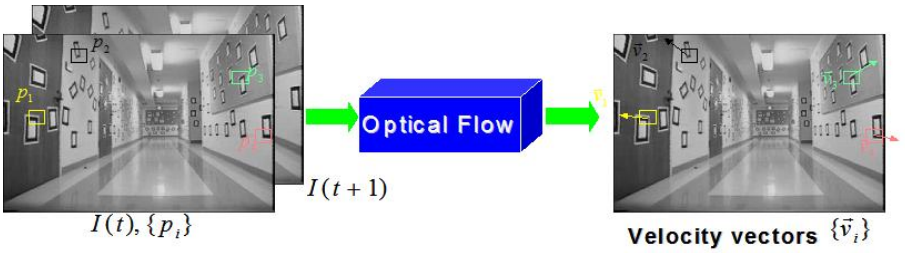

Figura 2.8: Introducción al cálculo del flujo óptico.

3. Si el valor es inferior al umbral, se pone el píxel en negro, y se vuelve a (1).

Con este método se reduce el ruido de la imagen, ya que el umbral superior actúa como primera criba muy restrictiva, y los puntos aislados cuya intensidad no es elevada no la pasan.

Este algoritmo de umbralización con histéresis se ha utilizado en el cálculo de la transformada de Hough, que se explica en la sección 2.4.7.

2.4.6. Flujo óptico

teoría, ecuaciones y forma de implementarlo y lo que se pretende.

El flujo óptico (optical flow) es el movimiento aparente de los patrones de intensidad luminosa o brillo de los objetos cuando la cámara que los observa se mueve en relación a ellos.

Su cálculo es una técnica diferencial que permite determinar el movimiento de los objetos de una escena a partir de un conjunto de imágenes variantes en el tiempo. Es una aproximación al movimiento local de una imagen basada en las derivadas locales en una secuencia de imágenes. Esto es, en 2D especifica cuánto se mueve cada píxel entre imágenes adyacentes (figura 5.12), mientras que en 3D especifica cuánto se mueve un volumen entre volúmenes adyacentes. En ambos casos, los movimientos provocan variaciones en la intensidad luminosa de la imagen. Nos centraremos en el algoritmo 2D, que es el que se ha implementado en las aplicaciones del proyecto.

Se puede extraer gran cantidad de información de imágenes variantes en el tiempo. De hecho, algunos datos son más fáciles de obtener de una secuencia

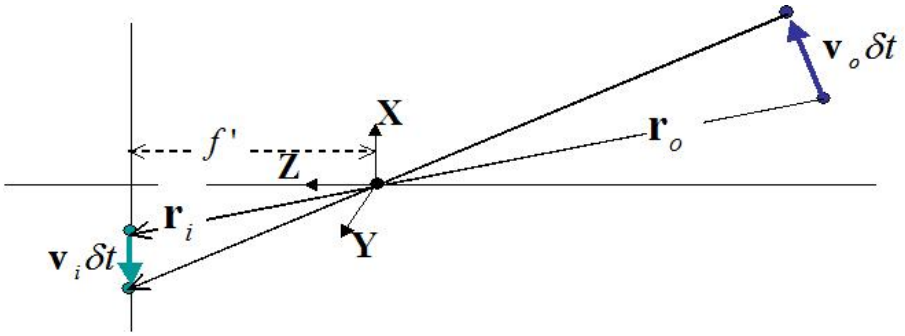

Figura 2.9: Proyección perspectiva de un punto de un objeto en una imagen.

que de una imagen aislada.

El campo de movimiento

Cuando los objetos se mueven frente a una cámara, o la cámara por un ambiente estático, se producen cambios en las imágenes. Estos cambios se pueden utilizar para recuperar los movimientos relativos, así como la forma de los objetos.

Definimos en primer lugar el campo de movimiento, que asigna un vector de velocidad a cada punto de la imagen. En un determinado instante de tiempo, un punto P_i en la imagen se corresponde con un punto P_o en la superficie de un objeto. Los dos están relacionados por la ecuación de proyección. El sistema de referencia se muestra en la figura 5.13.

Sea v_o la velocidad de P_o relativa a la cámara. Ésta induce un movimiento de velocidad v_i en el punto P_i. Los puntos se mueven $v \cdot \delta t$ en un intervalo de tiempo δt. Por tanto tenemos:

$$v_o = \frac{d\mathbf{r_o}}{dt} \qquad \text{y} \qquad v_i = \frac{d\mathbf{r_i}}{dt}, \tag{2.1}$$

donde r_o y r_i están relacionados por la ecuación de proyección perspectiva:

$$\frac{1}{f'}\mathbf{r_i} = \frac{1}{\mathbf{r_o} \cdot \hat{\mathbf{z}}}\mathbf{r_o}. \tag{2.2}$$

Y derivando esta ecuación de perspectiva obtenemos la ecuación del campo de movimiento

$$\frac{1}{f'}\mathbf{v_i} = \frac{(\mathbf{r_o} \cdot \hat{\mathbf{z}})\mathbf{v_o} - (\mathbf{v_o} \cdot \hat{\mathbf{z}})\mathbf{r_o}}{(\mathbf{r_o} \cdot \hat{\mathbf{z}})^2} = \frac{(\mathbf{r_o} \times \mathbf{v_o}) \times \hat{\mathbf{z}}}{(\mathbf{r_o} \cdot \hat{\mathbf{z}})^2}. \tag{2.3}$$

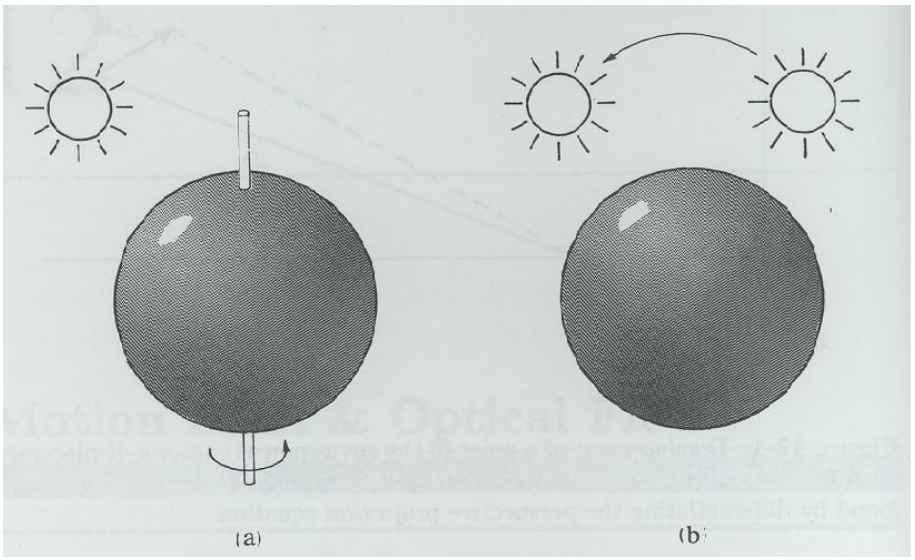

Figura 2.10: El flujo óptico no siempre es igual al campo de movimiento.

Por tanto se puede asignar un vector a cada punto de la imagen. Estos vectores constituyen el campo de movimiento.

Los puntos vecinos en un objeto tienen velocidades similares. Se espera por tanto que el campo de movimiento sea continuo en la mayor parte de la imagen. Las excepciones son las siluetas de los objetos, donde existen discontinuidades.

El flujo óptico

La distribución de brillo se una imagen se mueve según los objetos que la crean se mueven. El flujo óptico es el movimiento aparente de esa distribución. Idealmente el flujo óptico se corresponde con el campo de movimiento, pero como se muestra en la figura 5.14, esto no tiene por qué cumplirse. En (a) una esfera lisa rota bajo iluminación constante, la imagen no cambia, por lo que el flujo óptico es nulo, pero el campo de movimiento es distinto de cero. En (b) una esfera quieta se ilumina con una fuente móvil, la sombra de la imagen cambia, con lo cual el flujo óptico es distinto de cero, pero el campo de movimiento es cero.

La propiedad que nosotros podemos determinar visualmente es el flujo óptico, no el campo de movimiento. Aunque como en la mayoría de los casos

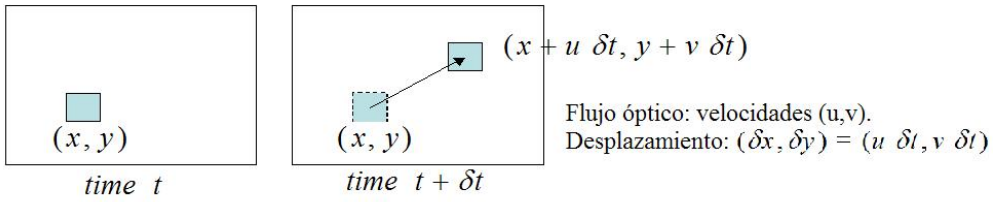

Figura 2.11: Desplazamientos y velocidades en el cálculo del flujo óptico.

coinciden, podremos utilizar el primero para averiguar el segundo. Hablamos entonces de *movimiento aparente del brillo.*

Sea $E(x, y, t)$ la intensidad, luminancia o brillo en un punto (x, y) en un instante t. Entonces $u(x, y)$ y $v(x, y)$ son las componentes del vector de flujo óptico en ese punto (figura 2.11). Asumimos que el brillo en el punto $(x + u\delta t, y + v\delta t)$ en el instante $t + \delta t$ es el mismo. Esto es,

$$E(x + u\delta t, y + v\delta t, t + \delta t) = E(x, y, t), \tag{2.4}$$

en un pequeño intervalo δt. Esta restricción no es suficiente para determinar u y v unívocamente. Asumimos entonces que la intensidad varía suavemente, con lo cual podemos expandir en serie de Taylor el miembro izquierdo de la ecuación anterior y obtener

$$E(x, y, t) + \delta x\frac{\delta E}{\delta x} + \delta y\frac{\delta E}{\delta y} + \delta z\frac{\delta E}{\delta z} + e = E(x, y, t), \tag{2.5}$$

donde e representa los términos de orden superior a uno. Cancelando $E(x, y, t)$, dividiendo por δt y haciéndola tender a cero, obtenemos

$$\frac{\delta E}{\delta x}\frac{\delta x}{\delta t} + \frac{\delta E}{\delta y}\frac{\delta y}{\delta t} + \frac{\delta E}{\delta t} = 0, \tag{2.6}$$

que es la expansión de la ecuación

$$\frac{dE}{dt} = 0. \tag{2.7}$$

Utilizando las abreviaturas

$$u = \frac{dx}{dt}, \qquad v = \frac{dy}{dt}, \qquad E_x = \frac{\delta E}{\delta x}, \qquad E_y = \frac{\delta E}{\delta y}, \qquad E_t = \frac{\delta E}{\delta t}, \tag{2.8}$$

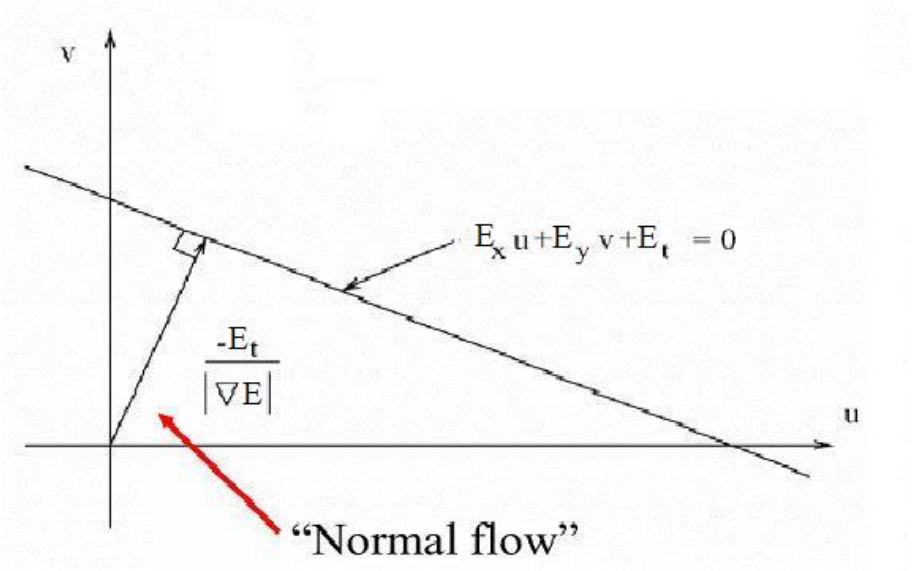

Figura 2.12: Espacio de velocidad.

obtenemos la Ecuación de Restricción del flujo óptico:

$$E_x u + E_y v + E_t = 0. \tag{2.9}$$

Si consideramos el plano de la figura 2.12, al que podemos llamar "espacio de velocidad", tenemos que los valores de (u, v) que satisfacen la ecuación de restricción se encuentran sobre una línea. Las medidas locales de intensidad sólo pueden identificar esa línea, con lo cual nos encontramos ante el denominado "problema de apertura", ya que no podemos encontrar los valores concretos de (u, v) con la información que tenemos hasta el momento. Esto hace que el movimiento a lo largo de un eje no se aprecie.

Podemos determinar la componente del flujo óptico normal a la línea, que viene dada por

$$\frac{E_t}{\sqrt{E_x^2 + E_y^2}}, \tag{2.10}$$

pero no la componente sobre la línea. El problema de apertura se ilustra en la figura 2.13, y en la figura 2.14 puede verse un ejemplo del flujo óptico entre dos imágenes.

Para resolver el problema de apertura, necesitaremos aplicar otras restricciones. Los distintos algoritmos existentes en la actualidad para el cálculo

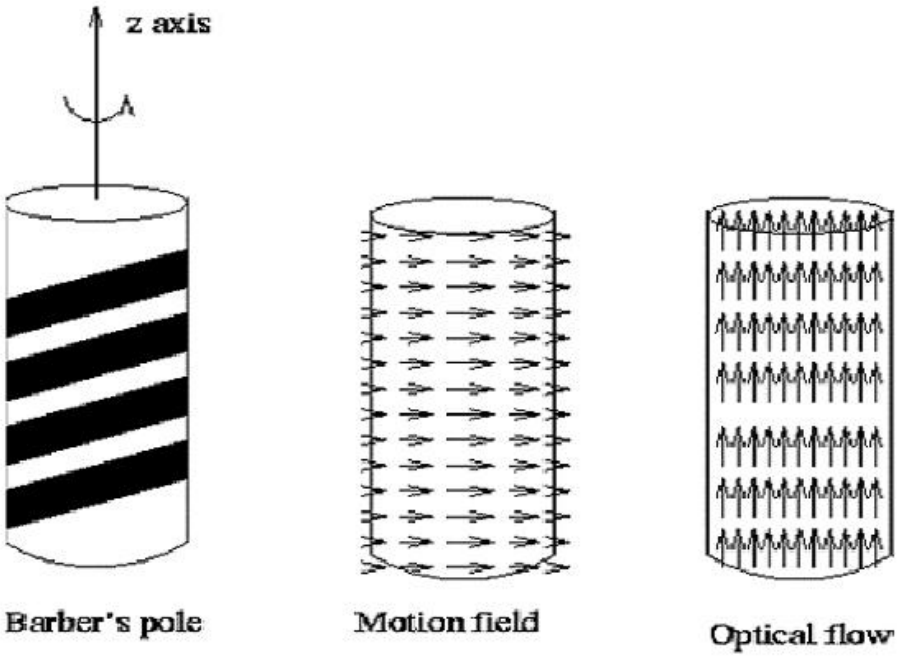

Figura 2.13: Problema de apertura.

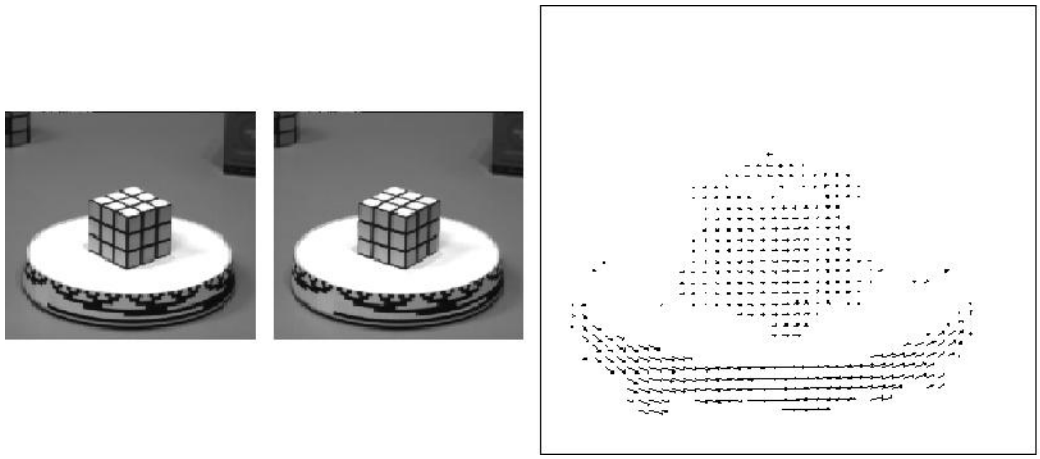

Figura 2.14: Ejemplo de flujo óptico.

del flujo óptico se diferencian entre sí en base a las restricciones extras que aplican a la ecuación anterior para darle solución.

Técnicas para el cálculo del flujo óptico

Existen varios tipos de técnicas para el cálculo del flujo óptico:

- Basadas en el desplazamiento de bloques (Block Matching) o de correlación de área.
- Basadas en la ecuación del flujo óptico (técnicas diferenciales). Resultan de la aplicación directa de la ecuación que define el flujo óptico. La más conocida es el algoritmo de Lucas-Kanade.
- Basadas en el acoplamiento de rasgos entre imágenes.

En este proyecto, se ha implementado el algoritmo del flujo óptico con una técnica basada en el desplazamiento de bloques. Esta es la técnica utilizada por ejemplo en la compresión de imágenes y vídeo, como se muestra en la figura 2.15. Consiste en coger bloques de píxeles de la imagen origen, de tamaño definido, y compararlos con bloques de la imagen destino. En la figura 2.16 se ilustra el funcionamiento. Los bloques de la imagen destino con los que se comparan resultan de desplazar la ventana que forma el bloque una cantidad de píxeles en todas direcciones, hasta un valor máximo definido.
Para cada desplazamiento se calcula la correlación entre los bloques. El máximo de correlación dará la coincidencia. Cuando se encuentra coincidencia, se computa el desplazamiento y se obtienen las velocidades horizontal y vertical del píxel del centro del bloque.

Esta técnica es conceptualmente sencilla e intuitiva. Su implementación en lenguaje de programación es también relativamente simple, ya que se basa en operaciones aritméticas sencillas. Esto ha llevado a su implantación en muchas aplicaciones. Sin embargo una mala elección de los parámetros 'tamaño de ventana'y 'desplazamiento permitido', puede elevar su coste computacional, al incrementar considerablemente el número de operaciones aritméticas que se deben realizar.

Con la implementación del flujo óptico en las aplicaciones desarrolladas se pretende obtener información de velocidad y movimiento de las chispas en las imágenes de rectificado de metales. Además se desea complementar

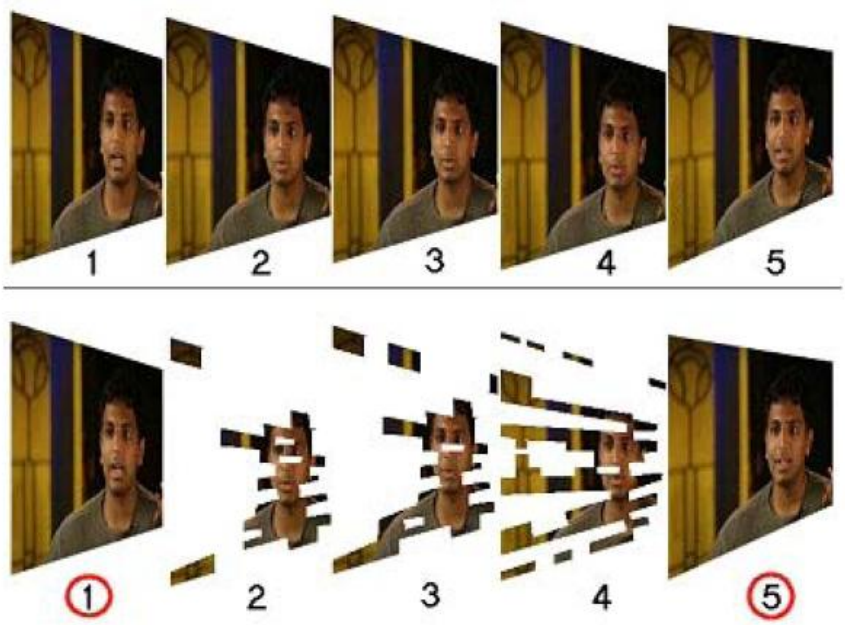

Figura 2.15: Compresión de imágenes mediante estimación de movimiento.

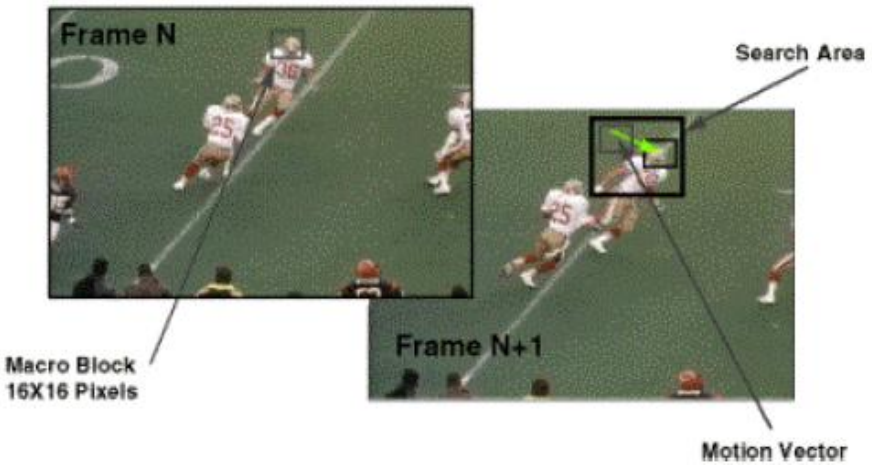

Figura 2.16: Búsqueda de bloques en la estimación del movimiento.

la información obtenida del algoritmo de la transformada de Hough, que se describe a continuación.

2.4.7. Transformada de Hough

La transformada de Hough es una técnica muy potente que se utiliza para aislar características de forma particular dentro de una imagen. Permite detectar curvas que puedan ser parametrizadas, como líneas rectas, polinomios y círculos. Los objetos de interés pueden estar parcialmente ocultos y tener cualquier tamaño.

La transformación consiste en pasar del plano (x, y) de la imagen a un espacio N-dimensional definido por los parámetros de la curva. En nuestras aplicaciones pretendemos detectar rectas correspondientes a las trayectorias de las chispas del proceso de rectificado. En este caso, las rectas se pueden representar con dos parámetros: pendiente y ordenada en el origen.

En el espacio transformado:

- Cada punto representa una posible recta en la imagen original, especificada por su pendiente y ordenada en el origen.
- Todas las posibles rectas que pasan por un punto en la imagen original se representan por una recta en la imagen transformada

En la figura 2.17 se muestra el paso del plano (x, y) al transformado.

El problema de esta representación para realizarla en un computador es que la pendiente puede tomar valor el valor infinito. Por eso se prefiere utilizar la parametrización de las rectas en coordenadas polares. Esto es:

$$\rho = x \cos\theta + y \sin\theta, \tag{2.11}$$

donde ρ es la longitud de una normal desde el origen hasta la línea, y θ es el ángulo de ρ respecto al eje x. Este plano transformado tiene las siguientes propiedades:

- Cualquier línea recta en una imagen es representada en un punto simple.

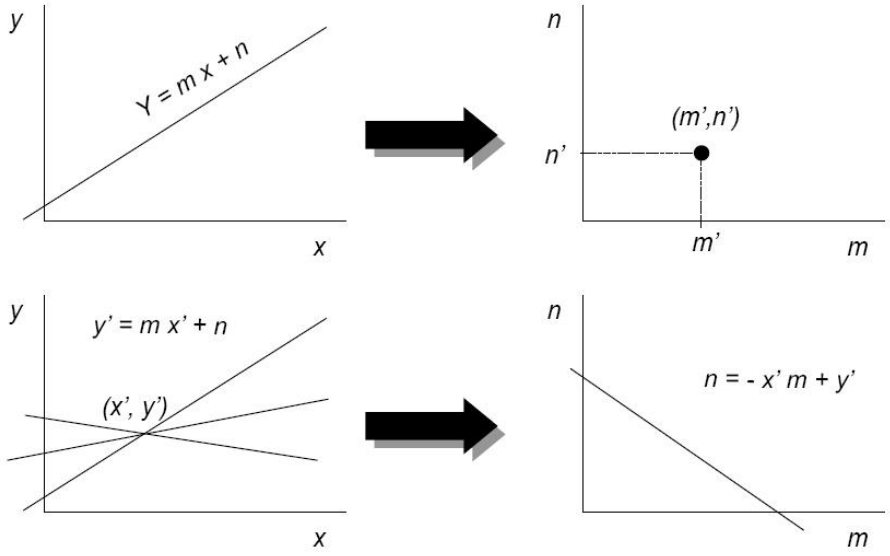

Figura 2.17: Transformada de Hough de rectas, plano (m, n).

- N puntos colineales son transformados en N curvas sinusoidales en el plano (ρ, θ). Los puntos de intersección de las curvas en el espacio paramétrico, corresponden a los parámetros de las posibles rectas que se encuentran en la imagen.

En la figura 2.18 se muestra el paso del plano (x, y) al plano (ρ, θ).

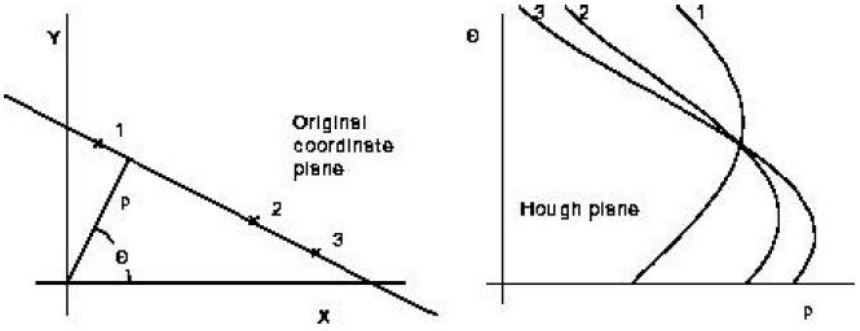

Figura 2.18: Transformada de Hough de rectas, plano (ρ, θ).

Implementación del cálculo de la transformada de Hough

El algoritmo para el cálculo digital de la transformada de Hough de imagen se basa en la construcción de un espacio acumulador. El acumulador es una matriz, cuyos elementos se corresponden con todos los valores posibles de (ρ, θ). Estos valores son valores discretos para poder implementar la transformada digitalmente. Por ejemplo θ varía entre 0 y 180 grados, y ρ es la distancia en píxeles enteros al origen de la imagen. Así, se tiene el espacio transformado discretizado en una matriz de enteros. Esto se ilustra en la figura 2.19, en ese caso se permiten valores negativos de los parámetros. Es más conveniente utilizar valores enteros positivos.

Algoritmo de Hough Sea E una imagen binaria de tamaño MxN, en la cual cada píxel toma valor '1'ó '0'.

1. Discretizar el espacio paramétrico en una matriz acumuladora de elementos (r,theta).
2. Recorrer la imagen. Para cada punto (x,y):
   - Para cada valor de theta, calcular: $\rho = x \cos\theta + y \sin\theta$.
   - Redondear al valor ρ obtenido al r más próximo.
   - Incrementar en una unidad el valor del elemento (r,theta) del acumulador.

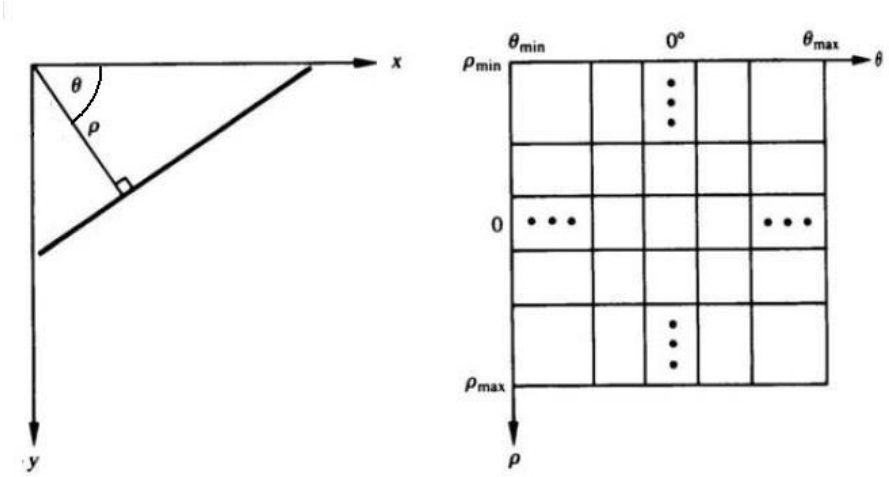

Figura 2.19: Formación del espacio acumulador.

3. Una vez construido el acumulador, reducir en el mismo las nubes de puntos que corresponden a una misma línea. Este fenómeno se debe a los redondeos en la construcción del acumulador. Las líneas del plano (x, y) no se corresponden con un único punto del acumulador. Se tienen nubes de puntos que se corresponden a una misma línea. Para evitar la replicación de resultados erróneos hay que reducir estas nubes a un punto. Para ello el método utilizado es:

   - Recorrer el acumulador buscando el valor máximo.
   - Eliminar los puntos que se encuentran a menos de una cierta distancia de él.
   - Repetir el proceso hasta obtener el número de líneas deseado.

4. Registrar las coordenadas polares de las líneas identificadas.

Tratamiento previo

La transformada de Hough no se puede aplicar directamente sobre una imagen, eso no obtendría resultados satisfactorios. Hay que realizar un procesamiento previo de la imagen para resaltar los elementos que se pretenden detectar, en el caso que nos ocupa las líneas, concretamente aquellas que se corresponden a las trayectorias de las chispas.

El algoritmo completo de la transformada de Hough consiste en los siguientes pasos:

1. Convertir la imagen en color a blanco y negro. Este procedimiento se ha explicado en la sección 2.4.4. El objetivo es trabajar con una sola banda de intensidad.

2. Detectar los bordes de la imagen, con un operador como el de Sobel. De esta forma se resaltan los bordes de los elementos de la imagen, y se obtienen líneas nítidas.

3. Binarizar la imagen de bordes. Una vez se tiene los bordes, la información que interesa de los píxeles es si pertenecen o no a un borde. Para ello se reduce el rango de datos creando una imagen binaria (blanco o negro) mediante una umbralización de intensidad. Se pueden aplicar cualquiera de las técnicas descritas en el apartado 2.4.5. En este proyecto se ha optado por la umbralización con histéresis y umbrales dinámicos.

4. Aplicar el algoritmo de Hough.

5. Representar las líneas obtenidas. El último paso es representar los resultados de la transformada, una forma de hacerlo es dibujar las líneas obtenidas sobre la imagen original.

En la figura 2.20 se ilustran los pasos anteriores. Se parte de una imagen en blanco y negro, se detectan los bordes y se binariza, a continuación observamos el aspecto del espacio acumulador, y por último se dibujan las líneas obtenidas. En este ejemplo no se han reducido las nubes de puntos, y por eso aparecen líneas repetidas que difieren entre sí ligeramente.

Detección de bordes mediante el operador de Sobel

El operador de Sobel extrae todos los bordes de una imagen, independientemente de su dirección. La imagen resultante aparece como un resalte omnidireccional de los objetos de la imagen original.

El operador calcula el gradiente de intensidad de la imagen en cada punto, proporcionando la dirección de máximo crecimiento de menos a más intensidad, y el módulo de este crecimiento. Por tanto el resultado indica la suavidad

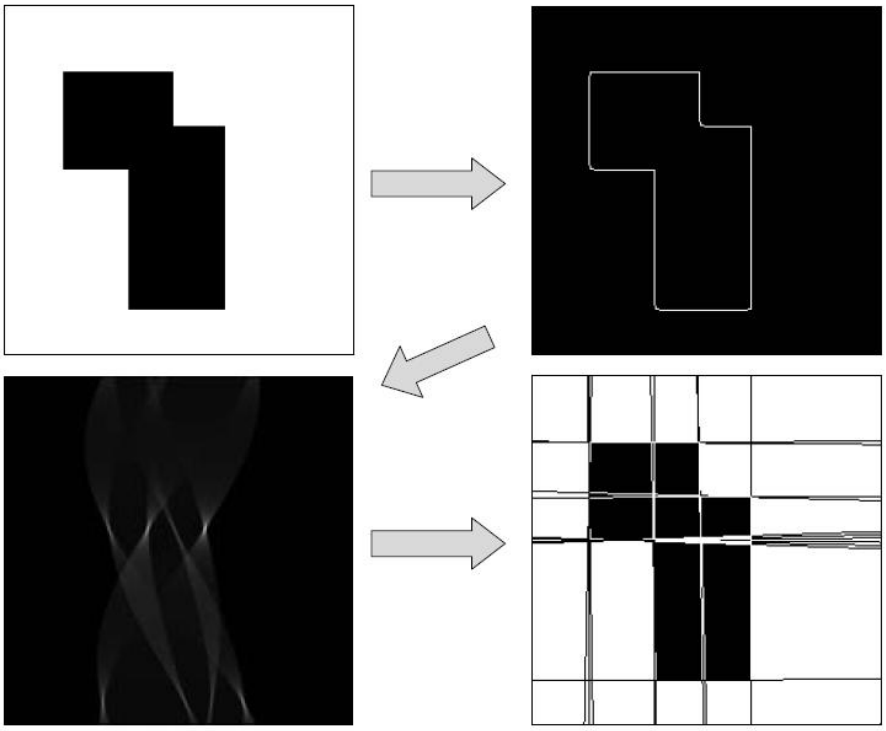

Figura 2.20: Ejemplo de funcionamiento del algoritmo completo de la transformada de Hough.

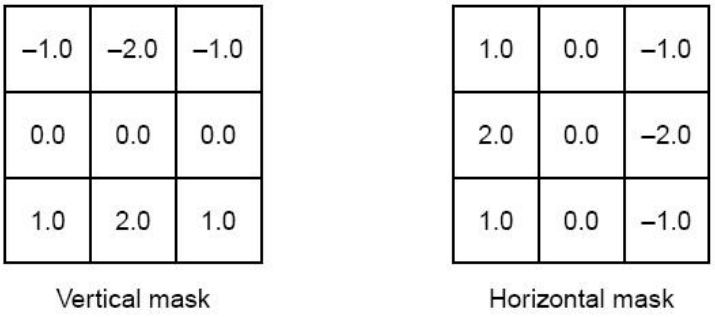

Figura 2.21: Máscaras de Sobel para la detección de bordes.

o la brusquedad del cambio de la imagen en ese punto, y por tanto la correspondencia de esa zona de la imagen con un borde y la orientación del mismo. En la práctica, el cálculo del módulo (similitud con un borde) es más fiable y fácil de interpretar que el cálculo de la dirección.

El gradiente de una función de dos variables, como la intensidad en la imagen, es un vector de dos componentes en las direcciones x e y, dadas por las derivadas de la función en esas direcciones. En cada punto de la imagen, el vector gradiente apunta en la dirección de máximo crecimiento posible de la intensidad. La magnitud de este crecimiento viene dada por el módulo del vector.
Esto implica que el resultado del operador de Sobel en un punto que pertenezca a una región de intensidad constante será un vector nulo. En un punto de borde, el vector apuntará en dirección normal al borde, de los puntos oscuros a los claros.

La implementación de la operación se lleva a cabo realizando la convolución espacial de la imagen con unas máscaras que son matrices de 3x3 píxeles (figura 2.21). De esta forma se realizan aproximaciones de las derivadas en cada dirección.

En cada punto de la imagen, el resultado de las aproximaciones del gradiente se puede combinar para obtener el módulo del gradiente con

$$|g| = \sqrt{g_x^2 + g_y^2} \tag{2.12}$$

Utilizando esta información, se puede calcular la dirección del gradiente con

$$\measuredangle g = \arctan \frac{g_y}{g_x} \tag{2.13}$$

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.

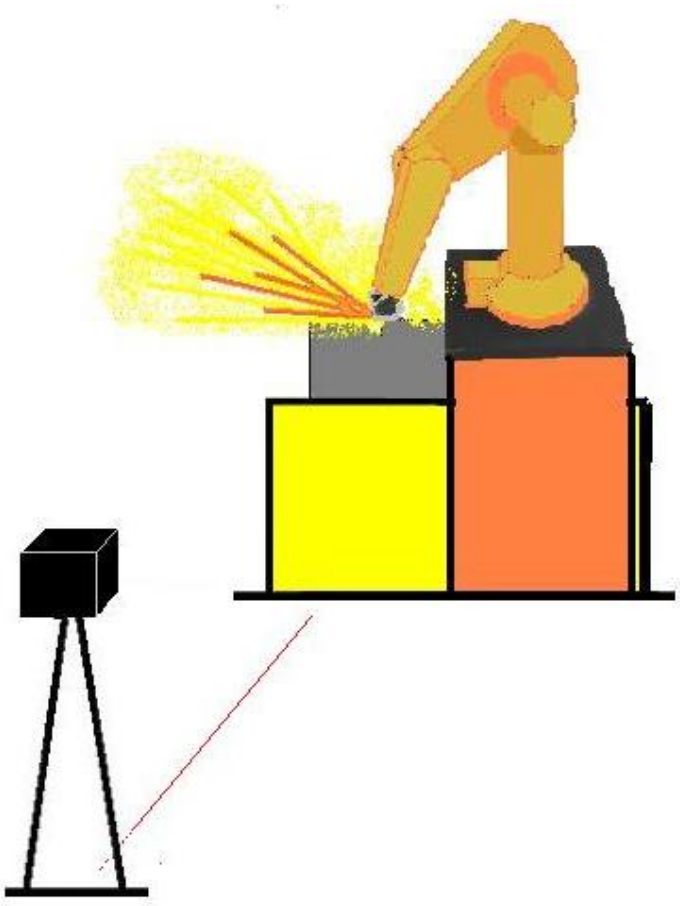

Figura 3.1: Esquema de instalación de trabajo

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

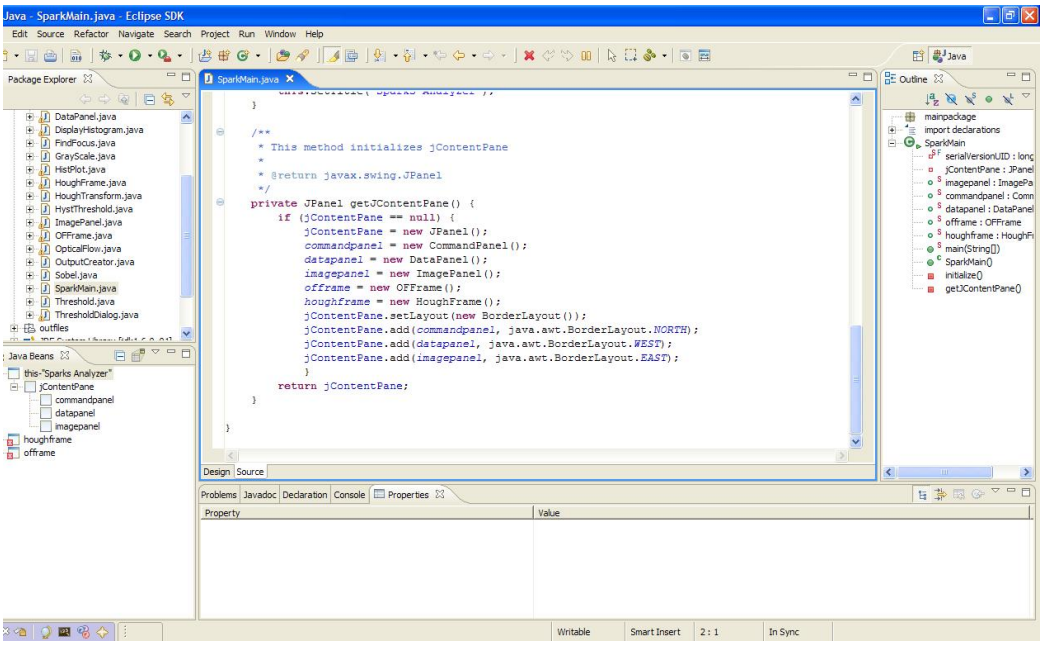

Figura 3.2: Entorno de trabajo o Workbench

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.

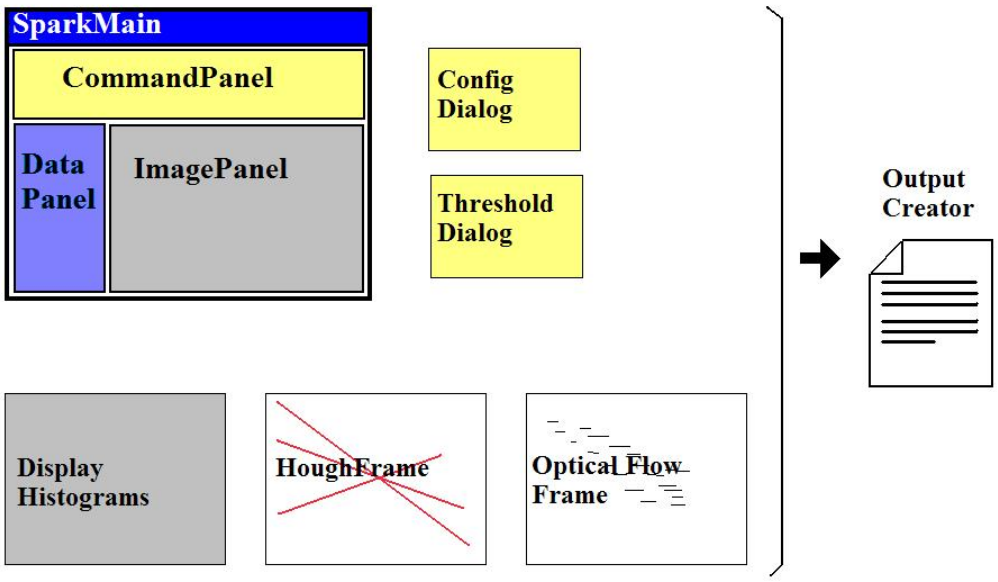

Figura 3.3: Estructura en bloques funcionales de SparksSolver

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.

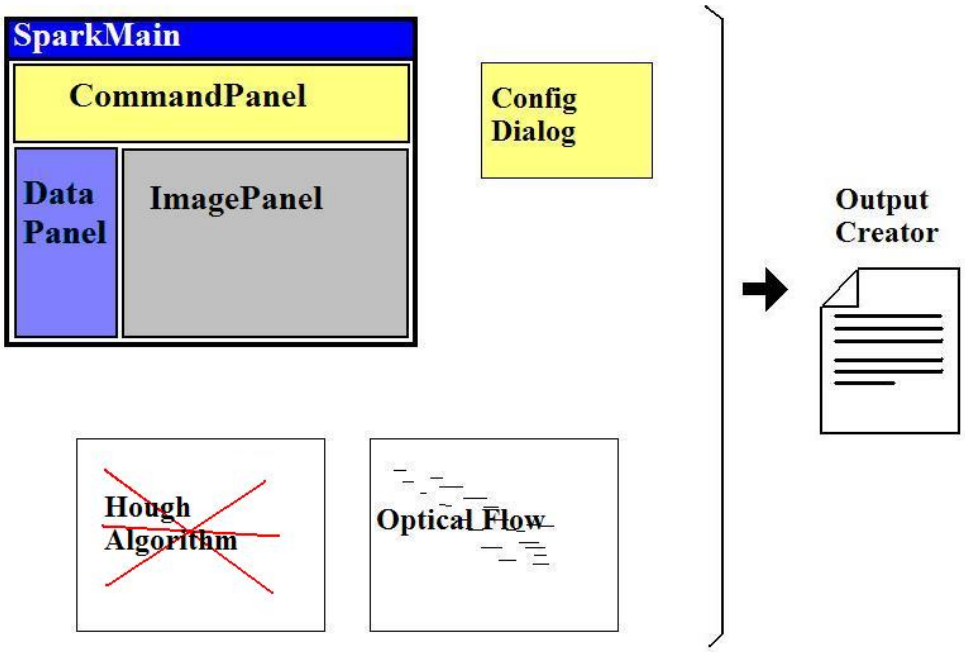

Figura 3.4: Estructura en bloques funcionales de SparksAnalyzer

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.

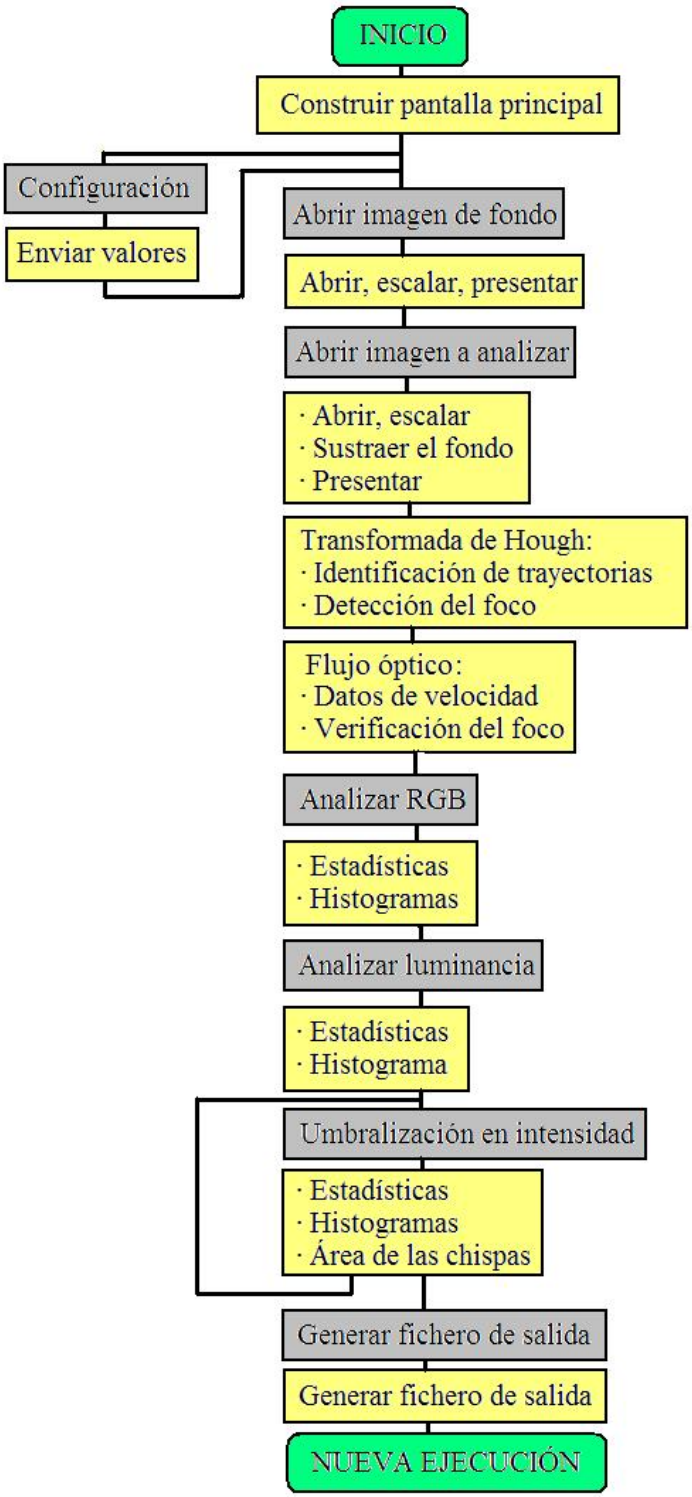

Figura 3.5: Diagrama de flujo de SparksSolver.

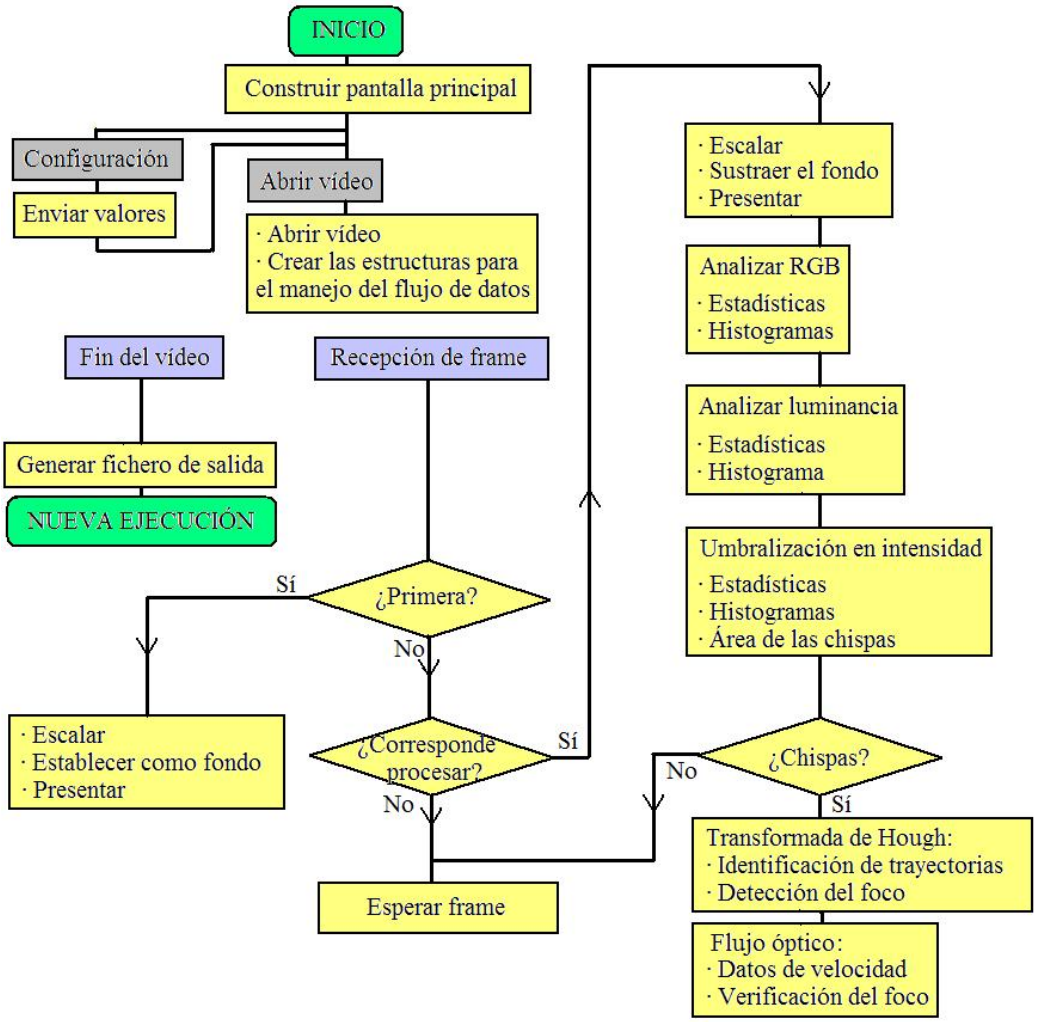

Figura 3.6: Diagrama de flujo de SparksAnalyzer.

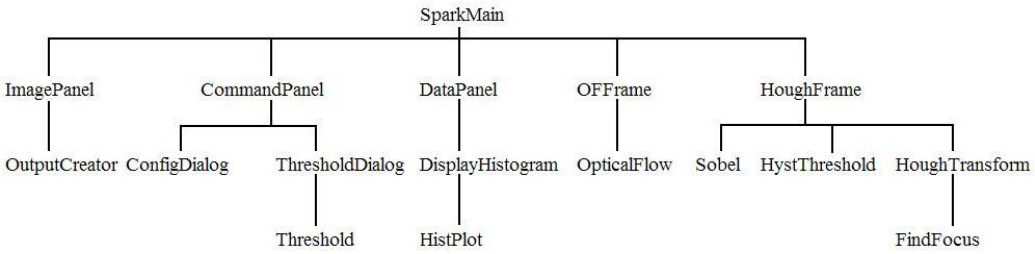

Figura 3.7: Estructura en clases de SparksSolver

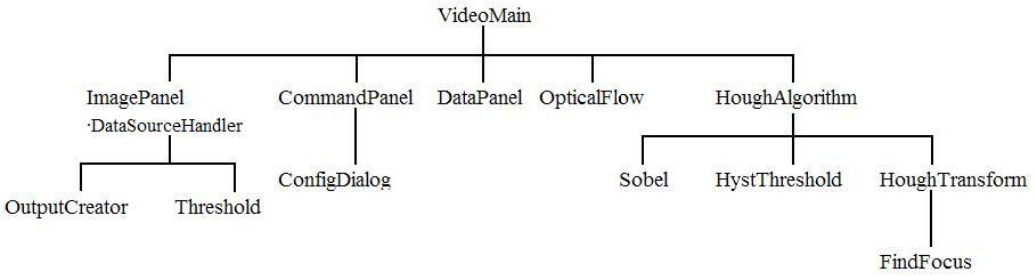

Figura 3.8: Estructura en clases de SparksAnalyzer

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.

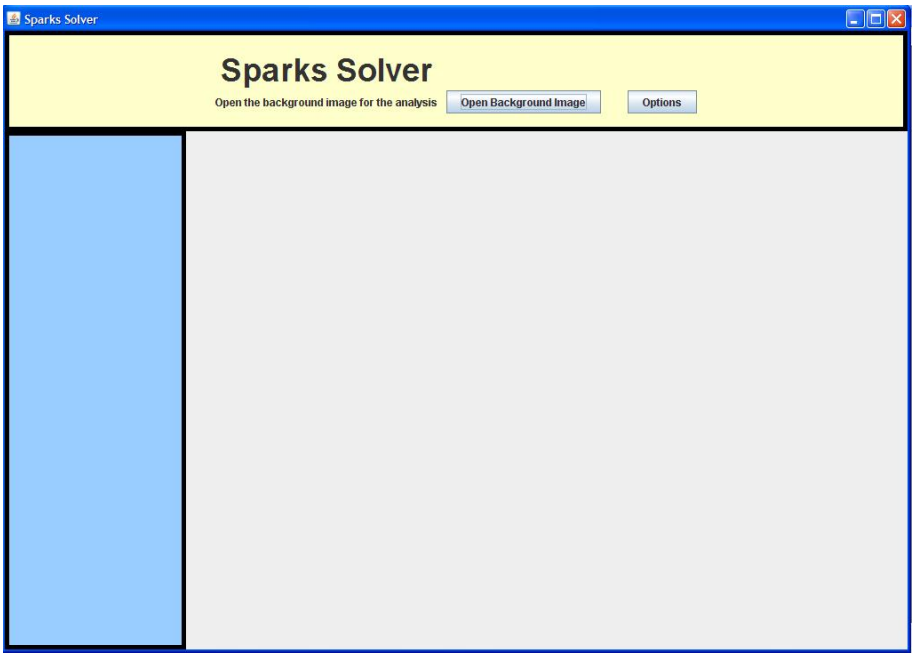

Figura 3.9: Aspecto de la pantalla principal de SparksSolver

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.

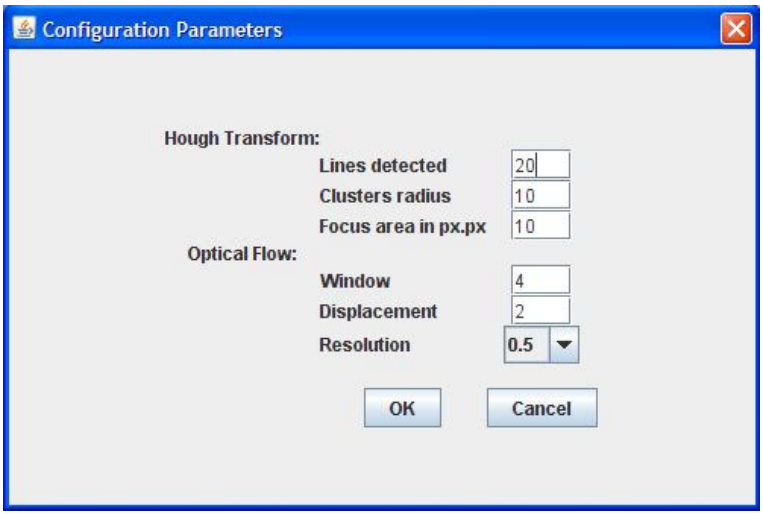

Figura 3.10: Aspecto del cuadro de configuración de SparksSolver

- 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.

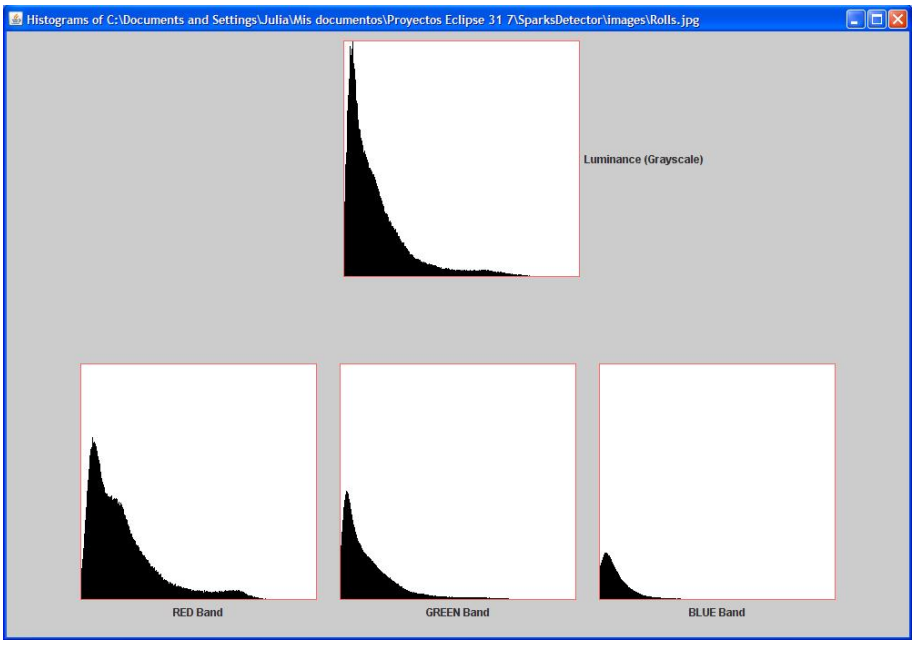

Figura 3.12: Ventana de representación de histogramas de SparksSolver

- 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.

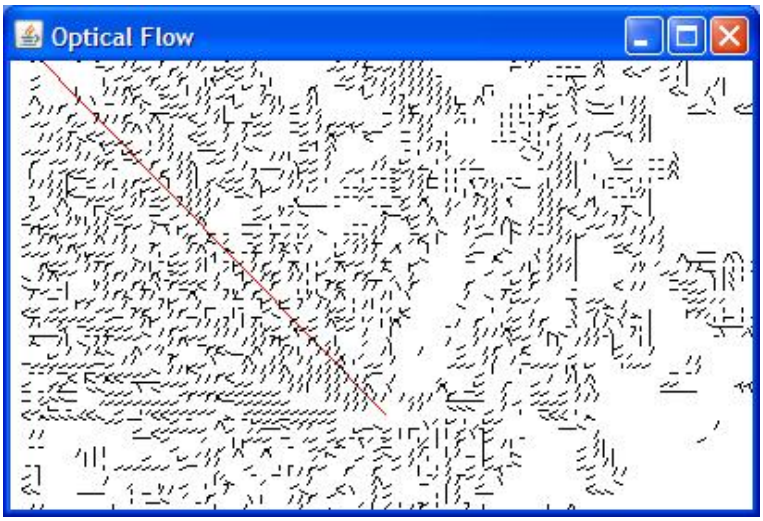

Figura 3.13: Ejemplo de representación del flujo óptico en SparksSolver

- 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

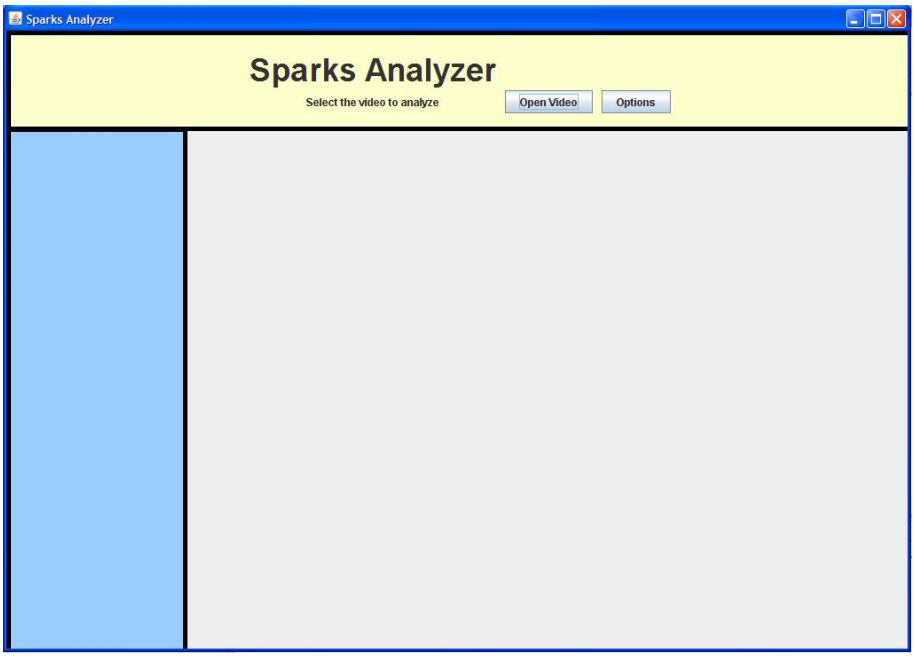

Figura 3.14: Aspecto de la pantalla principal de SparksAnalyzer

- 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.

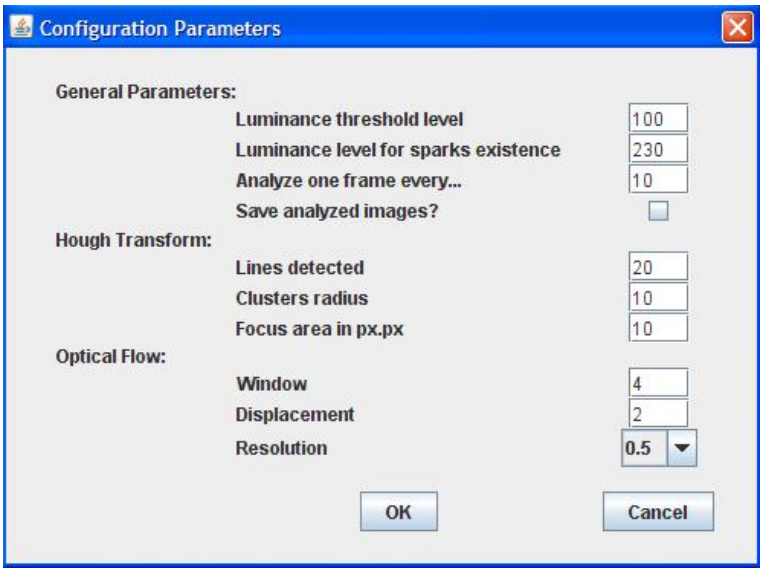

Figura 3.15: Aspecto del cuadro de configuración de SparksAnalyzer

- 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:

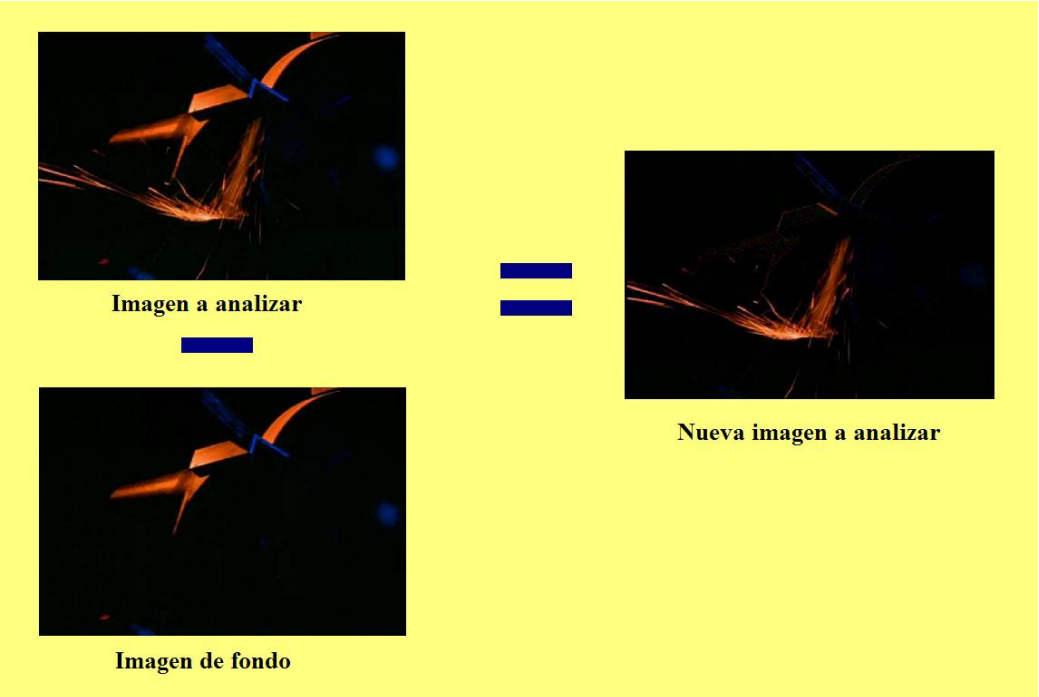

Figura 3.16: Operación de sustracción en la función subtract

  - 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.

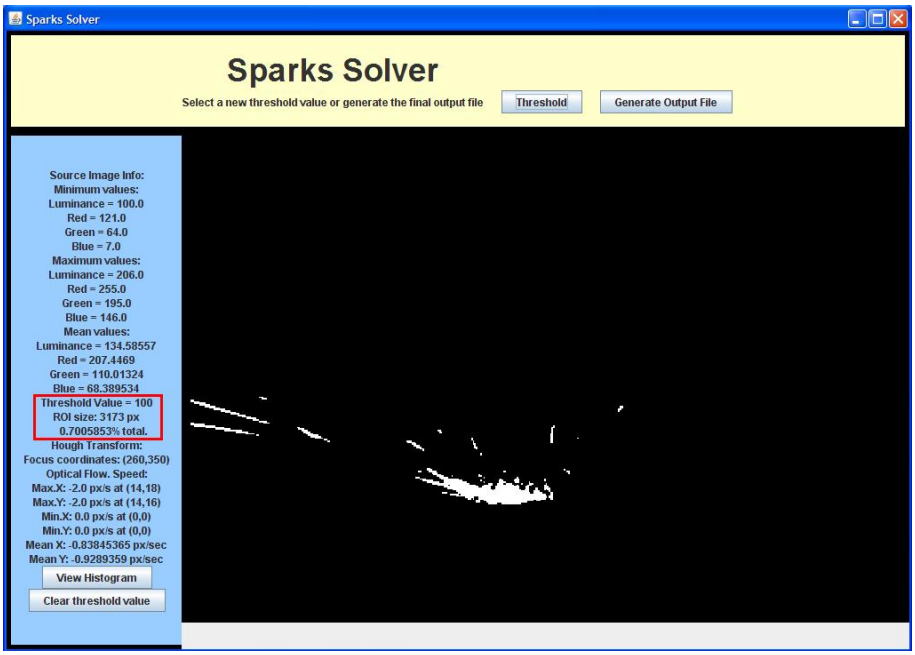

Figura 3.17: Umbralización en intensidad en la función startOperation

- 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.

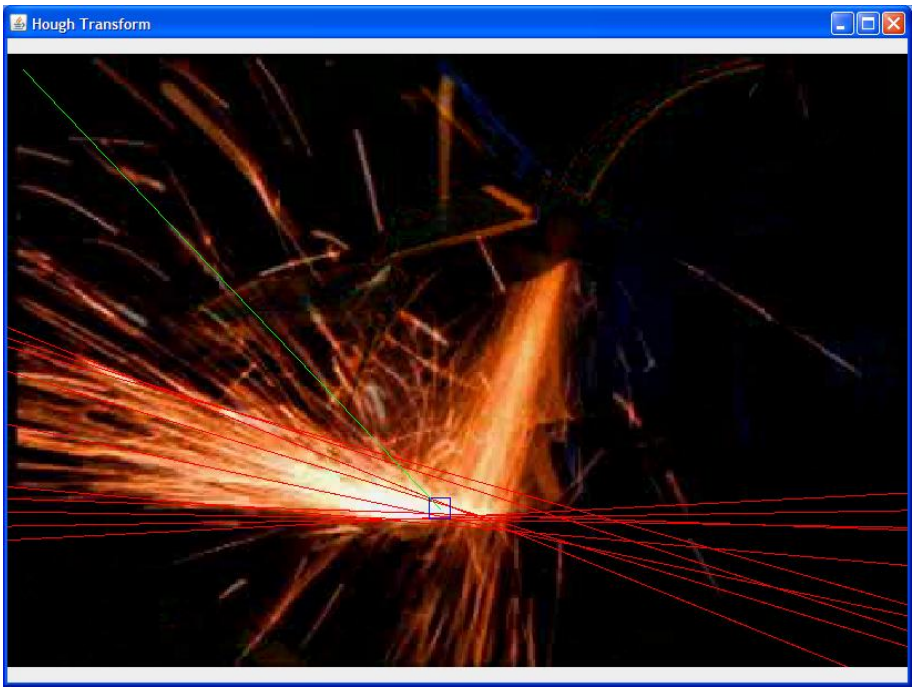

Figura 3.18: Resultado de la transformada de Hough, líneas obtenidas y foco identificado.

- 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()

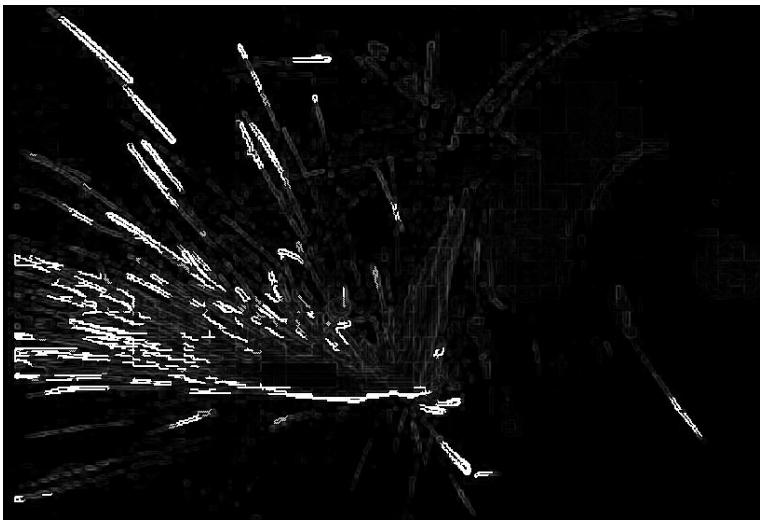

Figura 3.19: Resultado de la detección de bordes mediante el operador de Sobel

- 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).

Capítulo 4

Manual de usuario

En este capítulo se van a explicar los pasos necesarios para poner en funcionamiento las dos aplicaciones diseñadas en el proyecto fin de carrera.

En la primera parte se detallará la instalación de las aplicaciones desde el punto de vista del usuario. Para la instalación del entorno de desarrollo Eclipse y las librerías JAI y JMF remitimos al lector al Apéndice A.

En la segunda parte se explicará el funcionamiento de la aplicación SparksSolver, paso a paso, introduciendo ejemplos ilustrativos.
En la tercera parte se hará lo mismo para SparksAnalyzer.

4.1. Instalación

Para ejecutar aplicaciones escritas en lenguaje Java, es necesario descargar e instalar el Software de Java. Esto no es más que instalar la máquina virtual Java, que permite ejecutar un mismo programa en cualquier plataforma, dentro del denominado Java Runtime Environmet (JRE).

El software de Java se puede descargar gratuitamente en Java.com, en concreto en la página www.java.com/es/download/, como se muestra en la figura 4.1.

Al pulsar en **Descargar ahora**, la aplicación web identifica nuestro sistema operativo, y nos ofrece el software específico para nuestra plataforma (figura 4.2. Entonces debemos pulsar en **Iniciar descarga**, y seguir los pasos

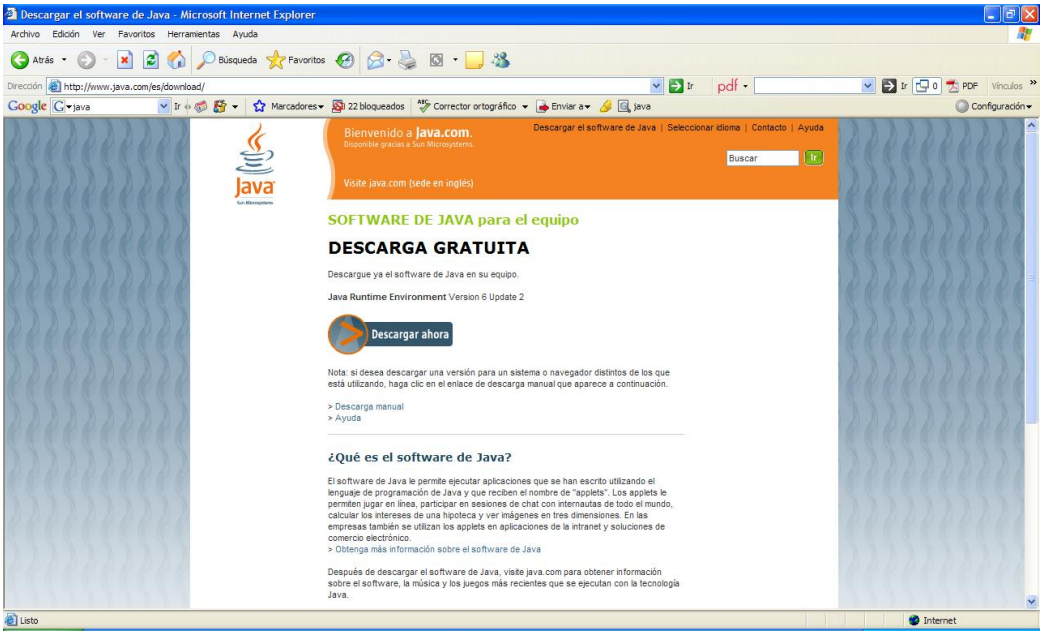

Figura 4.1: Página de descarga del software Java

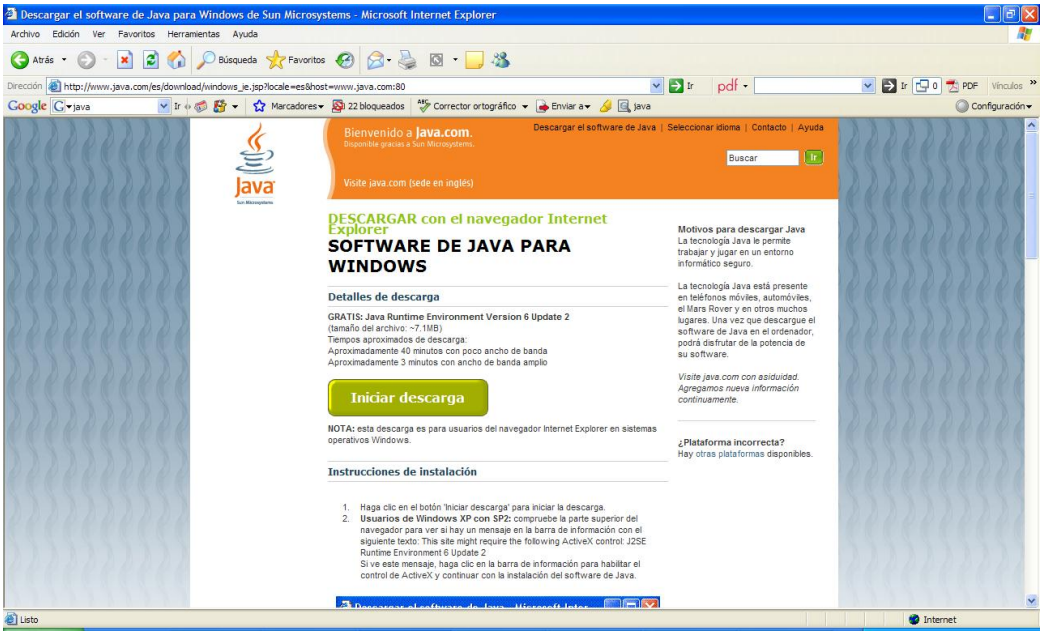

Figura 4.2: Página de descarga del software Java: identificación de plataforma

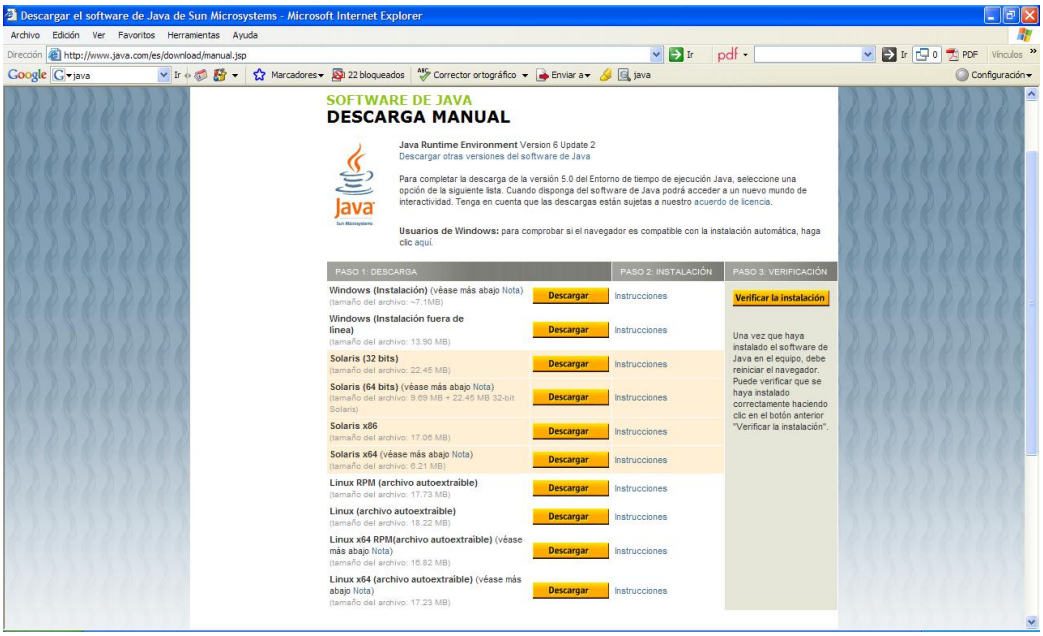

Figura 4.3: Página de descarga del software Java: descarga manual

de instalación específicos que se detallan justo debajo de ese botón.

Si no se ha identificado correctamente la plataforma, o se prefiere realizar la descarga manual, es posible hacerlo pulsando, en la parte derecha de la página, sobre **¿Plataforma incorrecta? Hay otras plataformas disponibles**. Esto da paso a la página de descarga manual que se muestra en la figura 4.3. Como puede verse, aparecen todos los sistemas operativos en los que puede ejecutarse un mismo programa escrito en lenguaje Java, y se proporcionan las instrucciones de instalación para cada uno de ellos. Es importante que, tanto si se ha descargado el software automáticamente como manualmente, se verifique que la instalación se ha realizado con éxito. En el caso automático, se abrirá un navegador que realizará la verificación. En el caso manual, se debe pulsar el botón de la parte derecha de la página de la figura 4.3.

Una vez que está instalado el software de Java en el equipo, es posible ejecutar las aplicaciones desarrolladas.

En el CD que se adjunta con esta memoria se proporcionan los programas SparksSolver y SparksAnalyzer, en sendas carpetas con los mismos nombres. Dentro de cada carpeta se encuentra el archivo ejecutable, de nombre

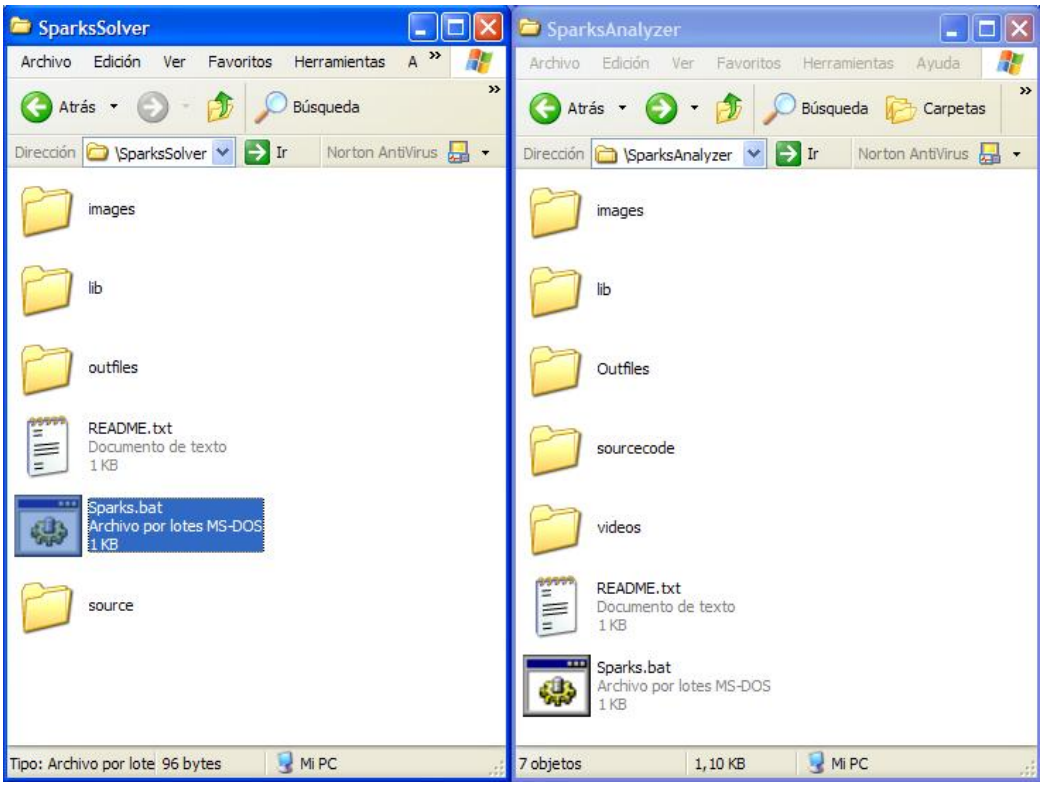

Figura 4.4: Carpetas donde se encuentran las aplicaciones

Sparks.bat en ambos casos (figura 4.4). Además se proporciona un archivo README con las instrucciones básicas de ejecución.
. En las carpetas 'lib'se encuentran las librerías externas utilizadas en cada programa, esto es, las librerías de JAI en SparksSolver y JAI y JMF en SparksAnalyer.
En las carpetas 'source'y 'sourcecode'se encuentra el código fuente.
En las carpetas 'outfiles'se guardan los ficheros de salida de texto de cada programa.
En SparksSolver, 'images'contiene imágenes para su análisis, y es el directorio inicial en la selección de imágenes desde archivo.
Algo análogo ocurre con la carpeta 'videosén SparksAnalyzer, en este caso contiene vídeos para analizar.
Adicionalmente, en SparksAnalyzer la carpeta 'imagesés aquella en la que se guardan las frames del vídeo analizadas, si se selecciona esa opción.

Para ejecutar las aplicaciones, basta hace doble click en su correspondiente archivo Spars.bat. A continuación aparecerá una ventana como la de la

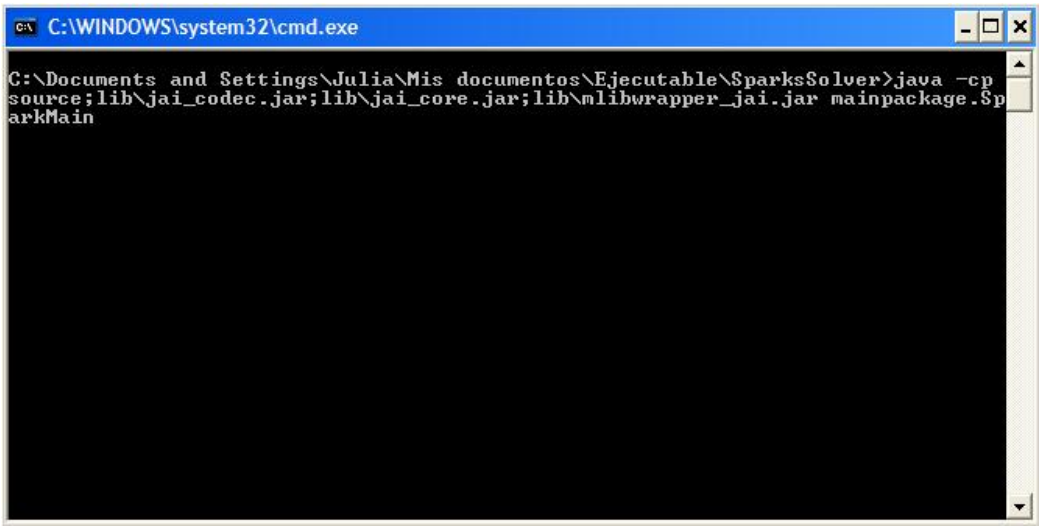

Figura 4.5: Ventana que aparece al ejecutar SparksSolver

figura 4.5 para SparksSolver, y como la de la figura 4.6 para SparksAnalyzer y seguidamente la pantalla principal de la aplicación. En las siguientes secciones se explica detalladamente el funcionamiento de ambos programas.

Cabe mencionar que si al ejecutar los archivos .bat de las aplicaciones, aparecen las ventanas de las figuras 4.5 y 4.6, pero no aparecen las pantallas principales de las aplicaciones, es probable que se haya producido un error en la instalación del software de Java. Se recomienda en este caso repetir la instalación.

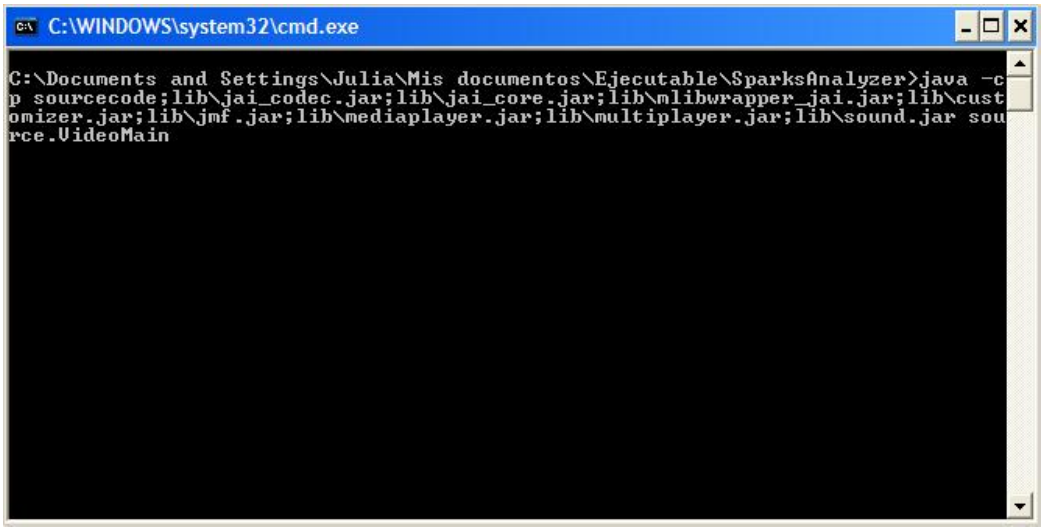

Figura 4.6: Ventana que aparece al ejecutar SparksAnalyzer

4.2. Explicación de uso de SparksSolver

Partimos de la pantalla principal de la aplicación (figura 4.7).

Desde esta pantalla podemos abrir directamente la imagen de fondo para el análisis o ajustar los parámetros de configuración de los algoritmos del programa. Estos parámetros son:

- Para la transformada de Hough:
  - Lines detected: número de líneas que va a detectar el algoritmo. Por defecto 20.
  - Clusters radius: radio en unidades de las nubes de puntos en r y theta que corresponden a una misma línea. Por defecto 10.
  - Focus area in px·px: área de la región que será identificada como foco de las chispas. Por defecto 10 (10x10).
- Para el cálculo del flujo óptico:
  - Window: tamaño de ventana de la imagen de la que se busca similitud. Por defecto 4 (4px·4px)
  - Displacement: desplazamiento máximo permitido en la búsqueda. Por defecto 2 px.

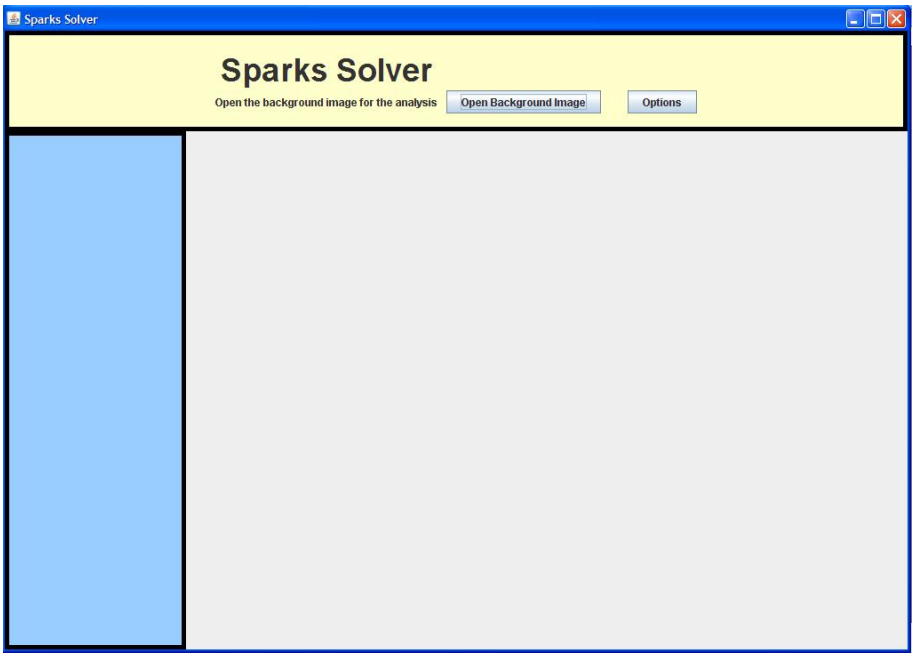

Figura 4.7: Pantalla principal de SparksSolver

- Resolution: escalado previo a las imágenes a analizar en el algoritmo, para reducir el coste computacional a costa de perder precisión. Por defecto 0.5 (las dos dimensiones de la imagen se reducen a la mitad).

En caso de no realizar ninguna modificación en la configuración, se utilizarán los valores por defecto.

Veamos un ejemplo en el que pulsamos el botón **Options** para modificar previamente esos valores (figura 4.8). Los valores seleccionados son:

- Lines detected: 10.
- Clusters radius: 10.
- Focus area in px·px: 20.
- Window: 4.
- Displacement: 2.

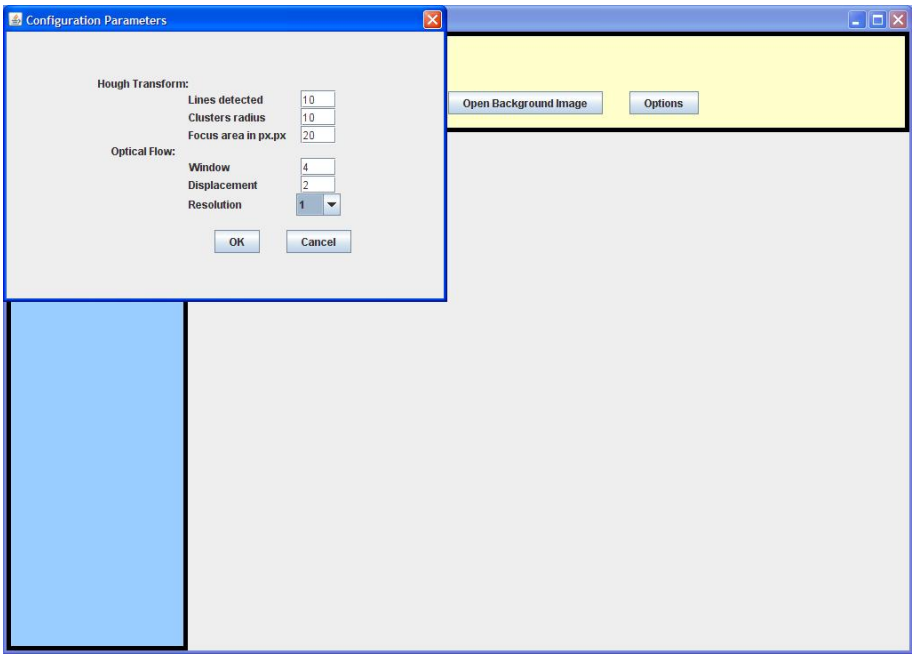

Figura 4.8: Pantalla principal de SparksSolver

- Resolution: 1 (no se realiza escalado adicional)

Pulsamos el botón **OK** para hacer efectivos los cambios.

Ahora procedemos a seleccionar la imagen de fondo para el análisis. Pulsamos el botón **Open Background Image**, aparecerá un cuadro de diálogo para seleccionar el archivo, el directorio inicial es 'images', en la carpeta de la aplicación (figura 4.9).

En este directorio se proporcionan algunas imágenes para analizarlas. Seleccionamos por ejemplo background2.jpg, ya que es una imagen didáctica que ilustra muy bien el funcionamiento de los algoritmos. La imagen aparece en la pantalla principal (figura 4.10).

Hecho esto seguimos las instrucciones de la propia aplicación y pulsamos el botón **Open File** para abrir la imagen que vamos a analizar y a la que se va a sustraer el fondo. Elegimos por ejemplo image_2316876890.jpg.
Automáticamente se sustrae el fondo y se ejecutan los algoritmos de la transformada de Hough y el flujo óptico. En la pantalla principal de la aplicación aparecerá la imagen seleccionada con el fondo sustraído. A la izquierda en es-

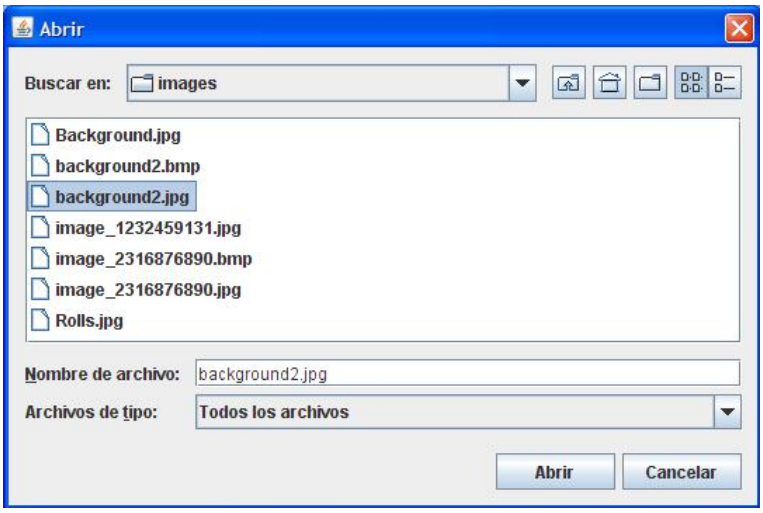

Figura 4.9: Cuadro de diálogo para seleccionar la imagen de fondo.

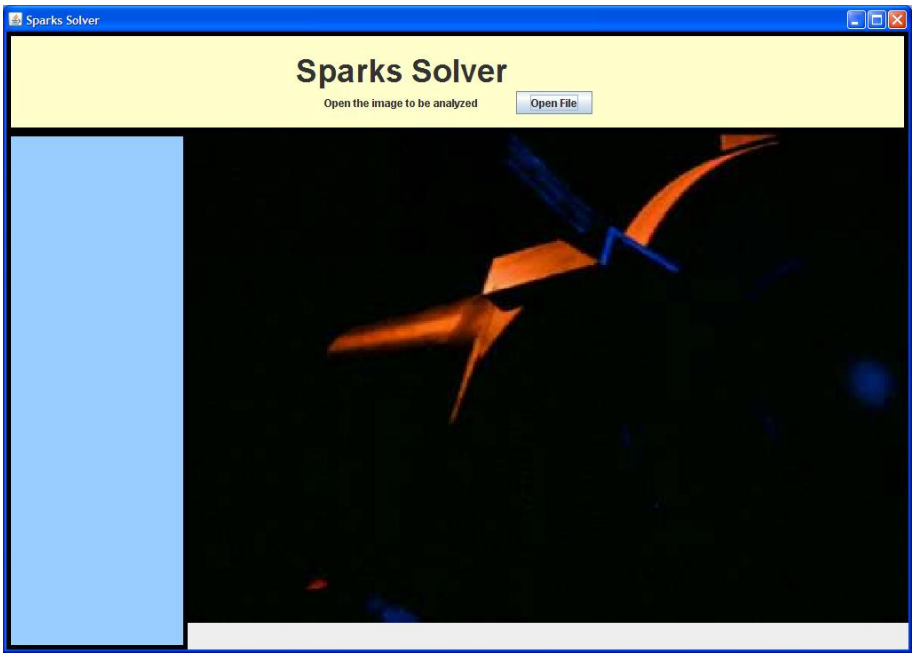

Figura 4.10: Imagen de fondo seleccionada.

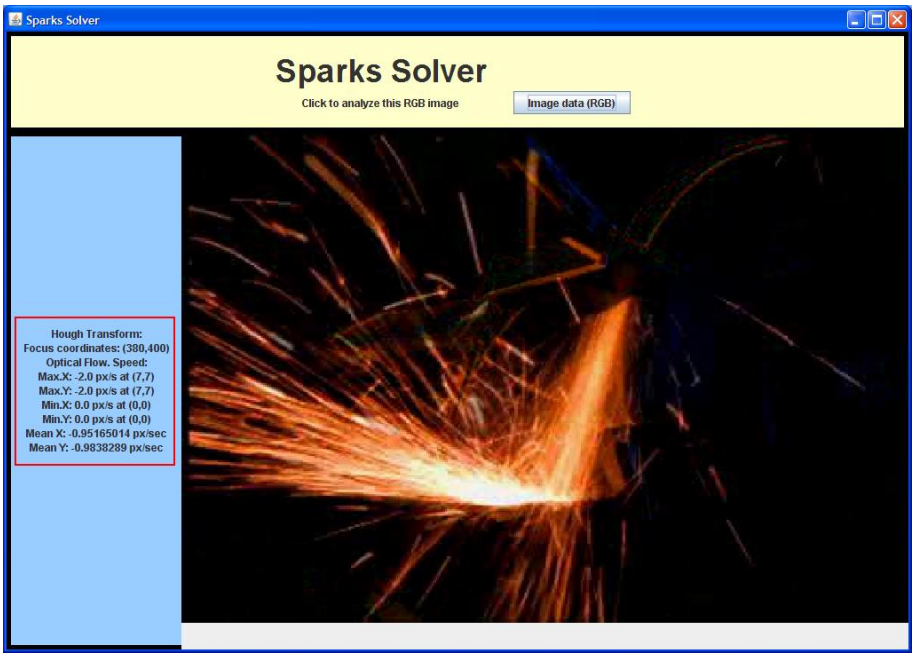

Figura 4.11: Imagen fuente con el fondo sustraído.

ta pantalla aparecen los datos calculados en los algoritmos anteriores (figura 4.11).

Aparecen dos nuevas ventanas, una para cada algoritmo aplicado. En la ventana **Optical flow** tenemos el gráfico resultante del cálculo del flujo óptico entre la imagen fuente y el fondo (figura 4.12). La línea roja une el punto de máxima velocidad con el foco de las chispas identificado.

En la ventana **Hough Transform** tenemos el resultado del procesamiento a partir de la transformada de Hough: las líneas encontradas correspondientes a la trayectoria de las chispas pintadas de rojo, y la zona identificada como foco de las chispas, recuadrada en color azul (figura 4.13). De la integración del foco calculado con el flujo óptico surge la línea verde, que une el punto de máxima velocidad con el foco.

Siguiendo las instrucciones que aparecen en la propia ventana de la aplicación, pulsamos el botón **Image data (RGB)**, y eso da paso al análisis de las bandas de la imagen RGB. Este análisis comprende la determinación de los valores máximo, mínimo y medio de cada banda de la imagen, y la elaboración y representación de los histogramas. Los datos calculados aparecen en la parte izquierda de la ventana de la aplicación (ver figura 4.14).

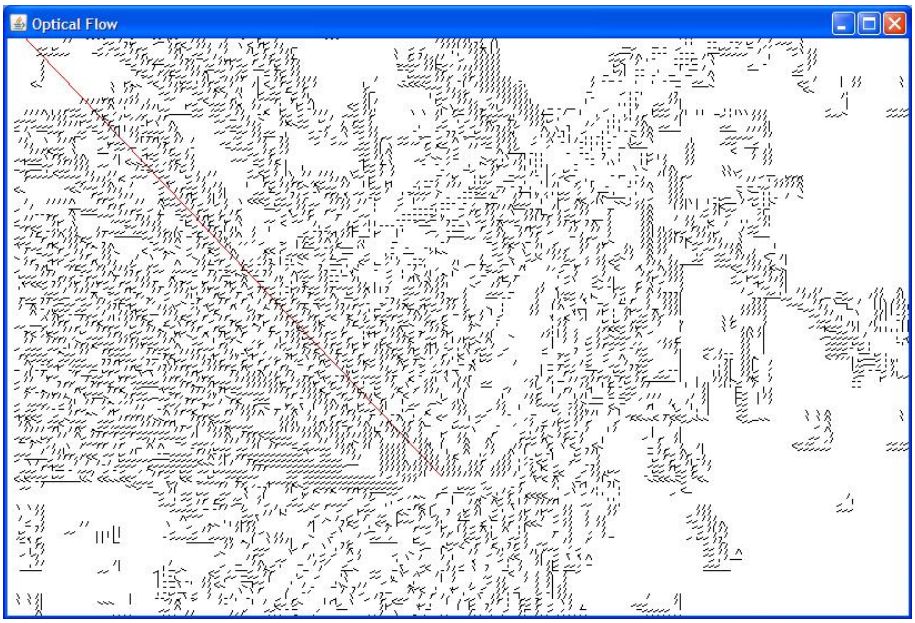

Figura 4.12: Resultado del cálculo del flujo óptico.

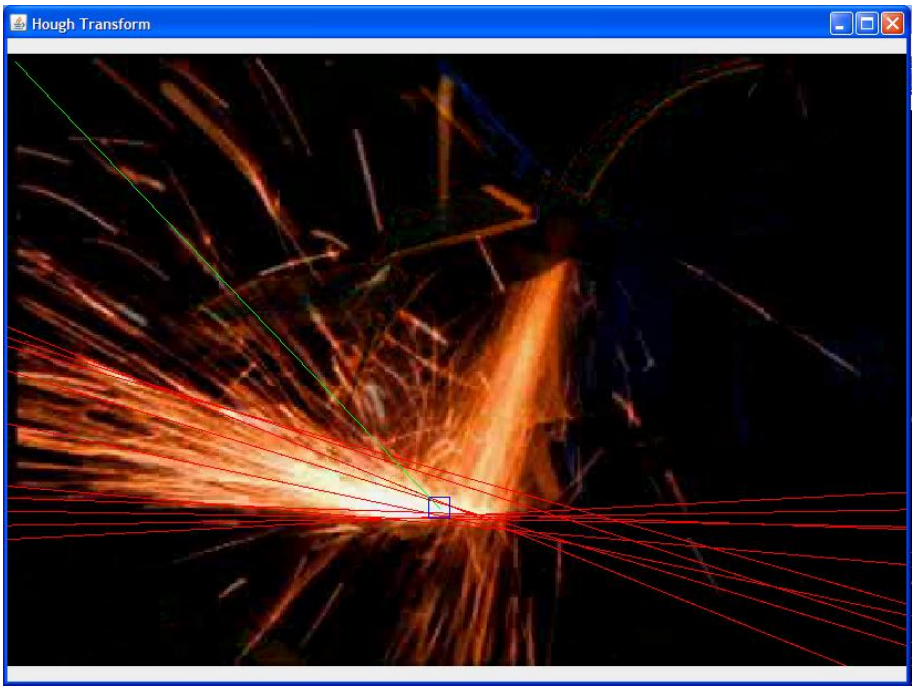

Figura 4.13: Resultado del cálculo de la transformada de Hough

Figura 4.14: Aparición de los datos calculados a las bandas RGB.

Se puede continuar con el análisis, o ver los histogramas calculados. En caso de pulsar el botón **View Histogram** de la parte izquierda de la ventana, aparece otra ventana con los gráficos de los histogramas (figura 4.15). Los datos numéricos de los histogramas se encuentran en el fichero de salida de texto, en la carpeta 'outfiles'. En este caso, el elevado número de píxeles con valor '0'provoca que los histogramas tengan ese aspecto. Hay que destacar el hecho de que la altura de los gráficos está normalizada al valor máximo de la ordenada.

Para continuar, se pulsa el botón **Image Data (GrayScale)** y se procede al análisis de la imagen en cuanto a luminancia. Para ello la imagen se transforma a blanco y negro, presentando el resultado en la ventana de la aplicación. Se determinan los valores máximo, mínimo y medio de luminancia, y se elabora y representa el histograma. Los datos calculados aparecen en la parte izquierda de la ventana de la aplicación (figura 4.16).

De nuevo pueden verse los histogramas calculados (figura 4.17).

Tras esto, el programa nos insta a seleccionar un valor de intensidad con el cual umbralizar la imagen. Se calculará el tamaño de la zona que se encuentre por encima de dicho umbral. La idea de este paso es computar el área de la

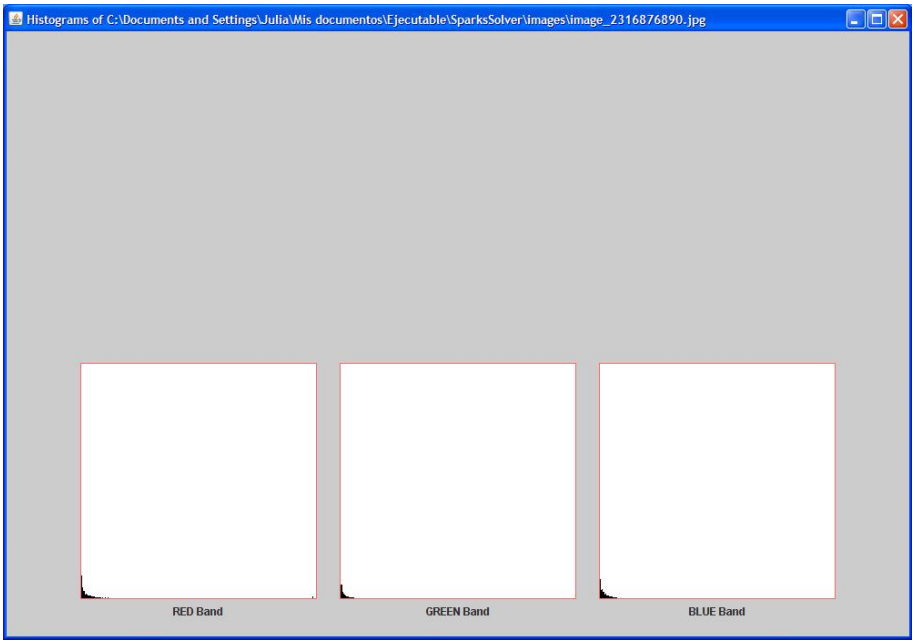

Figura 4.15: Ventana con los histogramas de las bandas RGB.

Figura 4.16: Aparición de la imagen en blanco y negro y los datos calculados.

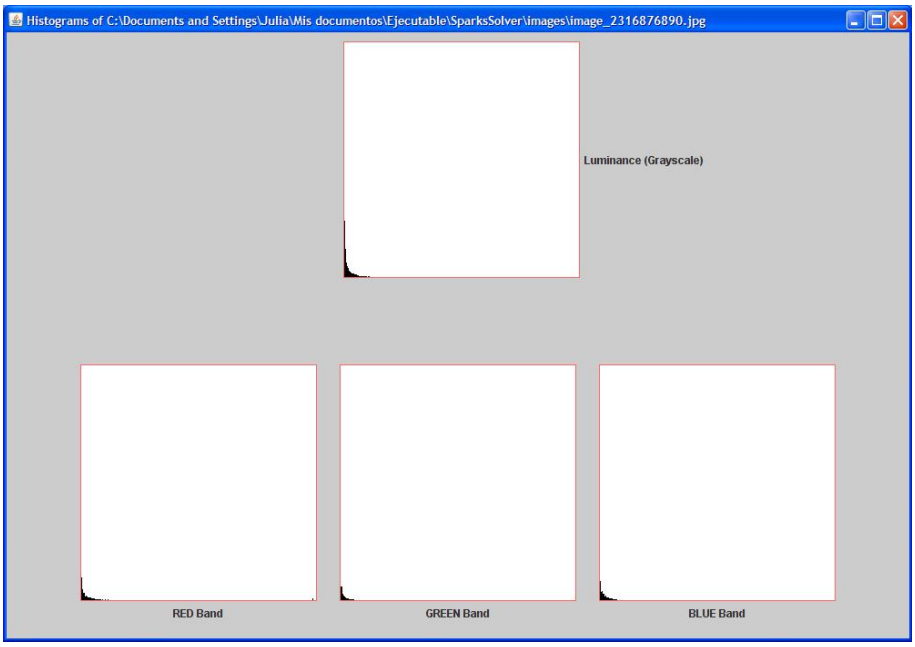

Figura 4.17: Ventana con los histogramas de luminancia y RGB.

zona de chispas en la imagen, considerando chispas aquellos píxeles con un nivel de intensidad superior al umbral.
Pulsamos el botón **Threshold)** y aparece un cuadro de diálogo en el que podemos seleccionar el umbral. Si elegimos por ejemplo 150, obtenemos el resultado de la figura 4.18.

Los nuevos histogramas se pueden ver en la figura 4.19. En este caso, al estar normalizados en altura respecto al valor máximo, se aprecia más claramente su forma. De nuevo se remite al usuario al fichero de salida para obtener los datos numéricos.

Este paso se puede repetir pulsando de nuevo el botón **Threshold)** para introducir un nuevo valor de umbral. También se puede volver a tener en pantalla la imagen que se está analizando pulsando el botón **Clear threshold value)**.

Por último, al pulsar **Generate Output File)** se genera el fichero de salida de texto con todos los datos calculados. Este fichero se salva en la carpeta 'outfiles', dentro del directorio del programa. Su nombre tiene el formato "nombre_de_imagen+data.txt. En el ejemplo que nos ocupa, es image_2316876890data.txt. La estructura del fichero de salida se explica detalla-

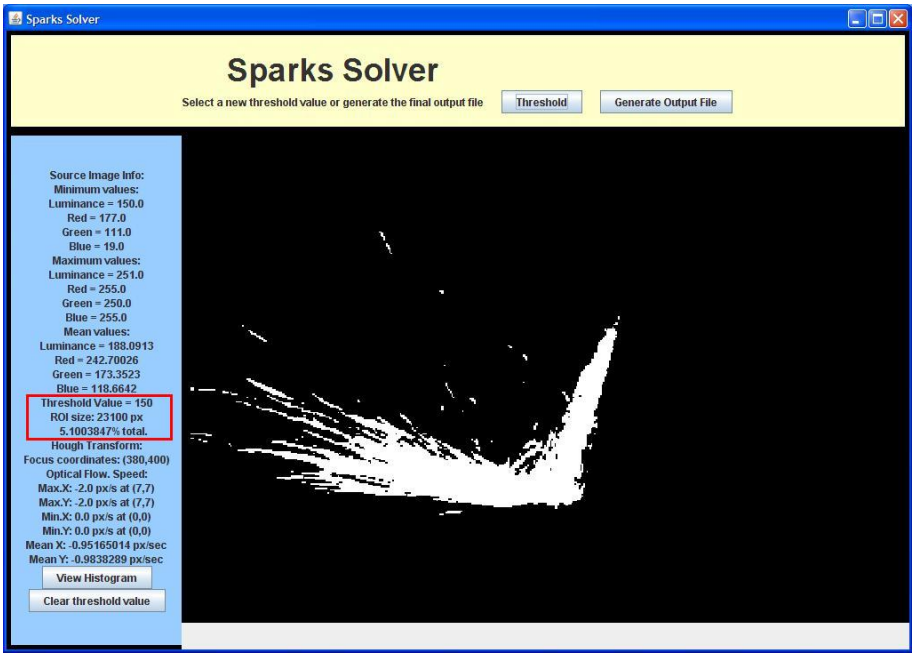

Figura 4.18: Umbralización en intensidad y datos calculados.

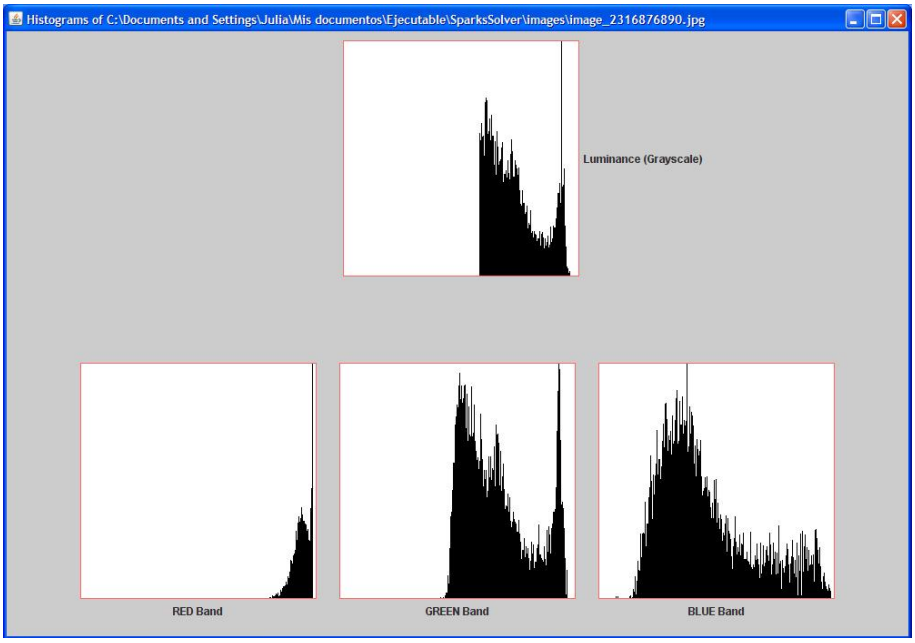

Figura 4.19: Histogramas tras la umbralización en intensidad.

damente en el Apéndice C.

Una vez generado el fichero de salida, la aplicación vuelve al estado inicial, como en la figura 4.7 en la página 89, y se puede analizar otro par de imágenes. También se pueden reconfigurar los parámetros.

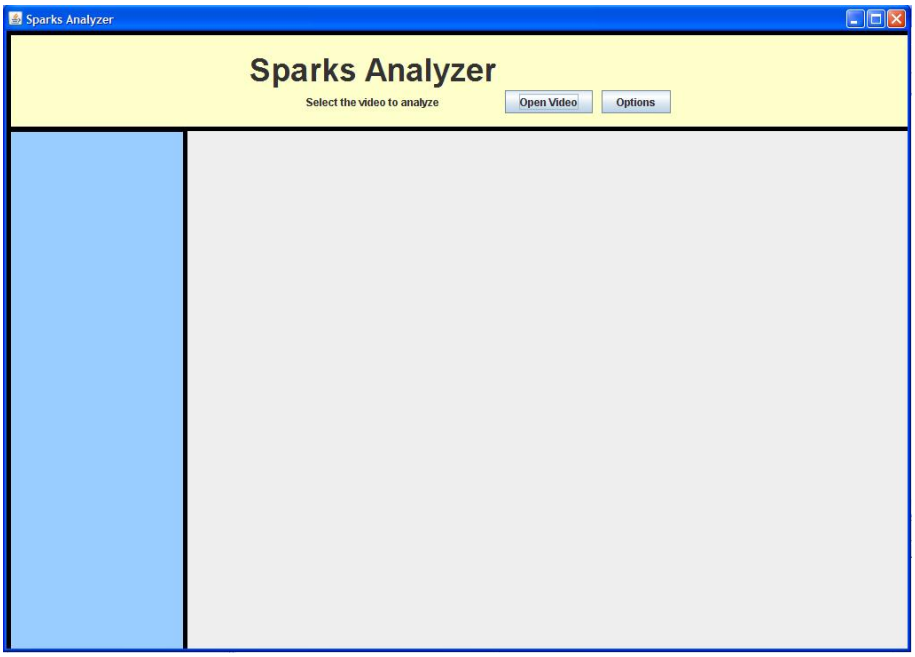

Figura 4.20: Pantalla principal de SparksAnalyzer

4.3. Explicación de uso de SparksAnalyzer

La pantalla de la aplicación SparksAnalyzer se muestra en la figura 4.20.

Podemos abrir directamente el vídeo que se va a analizar o ajustar los parámetros de configuración de los algoritmos del programa. En caso de no realizar ninguna modificación en la configuración, se utilizarán los valores por defecto. Los parámetros de funcionamiento de SparksAnalyzer son:

- Parámetros generales:
  - Luminance threshold level: Nivel para la umbralización en intensidad que se utiliza en la evaluación del tamaño de la zona de chispas. Por defecto 100.
  - 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. Por defecto 230.

  - Analyze one frame every...: Tasa de análisis de las frames recibidas, para conseguir mayor velocidad. Por defecto una de cada... 10.
  - Save analyzed images?: Permite seleccionar si se guardan o no las imágenes analizadas en la carpeta 'images'. Por defecto está deshabilitado.

- Para la transformada de Hough:
  - Lines detected: número de líneas que va a detectar el algoritmo. Por defecto 20.
  - Clusters radius: radio en unidades de las nubes de puntos en r y theta que corresponden a una misma línea. Por defecto 10.
  - Focus area in px·px: área de la región que será identificada como foco de las chispas. Por defecto 10 (10x10 px).

- Para el cálculo del flujo óptico:
  - Window: tamaño de la ventana de la imagen de la que se busca similitud (en px·px). Por defecto 4.
  - Displacement: desplazamiento máximo permitido en la búsqueda. Por defecto 2.
  - Resolution: escalado previo a las imágenes a analizar en el algoritmo, para reducir el coste computacional a costa de perder precisión. Por defecto 0.5.

El funcionamiento óptimo de SparksAnalyzer se consigue introduciendo en estos parámetros los parámetros de SparksSolver con los que se hayan obtenido buenos resultados. En el apartado anterior, cuando explicábamos el funcionamiento de SparksSolver se utilizaron:

- Lines detected: 10.
- Clusters radius: 10.
- Focus area in px·px: 20.
- Window: 4.
- Displacement: 2.

- Resolution: 1 (no se realiza escalado adicional)

Así que para ilustrar el funcionamiento de SparksAnalyzer vamos a utilizar algunos de ellos. Pulsando el botón **Options** introducimos:

- Luminance threshold level: 150
- Luminance level for sparks existence: 240.
- Analyze one frame every. . . : 10.
- Save analyzed images?: Sí.
- Lines detected: 10.
- Clusters radius: 10.
- Focus area in px·px: 20.
- Window: 4.
- Displacement: 2.
- Resolution: 0.5 (escalado a la mitad para disminuir el coste computacional)

Esto se muestra en la figura 4.21.

Una vez configurada la aplicación procedemos a seleccionar el vídeo que se va a analizar, pulsando el botón **Open Video**. Aparecerá un cuadro de diálogo para seleccionar el archivo, el directorio inicial es 'videos', en la carpeta de la aplicación (figura 4.22).

De entre los vídeos proporcionados, seleccionamos por ejemplo sparks.mpeg. De este vídeo proceden las imágenes utilizadas para explicar el funcionamiento de SparksAnalyzer en el apartado anterior.

Al abrir el vídeo, automáticamente el programa empieza a procesar las frames que va recibiendo del flujo de vídeo. Las frames procesadas se van mostrando en la pantalla principal de la aplicación, así como el resultado de la transformada de Hough. De este modo se van observando las trayectorias y el foco de las chispas identificados. Los datos obtenidos de todos los algoritmos aparecen en la parte izquierda de la pantalla.

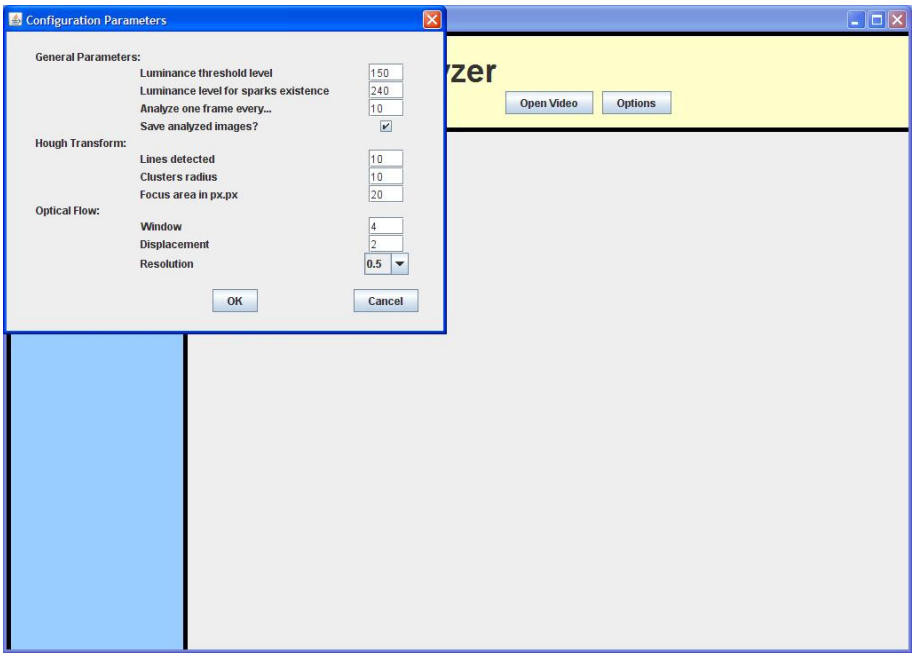

Figura 4.21: Configuración de SparksAnalyzer

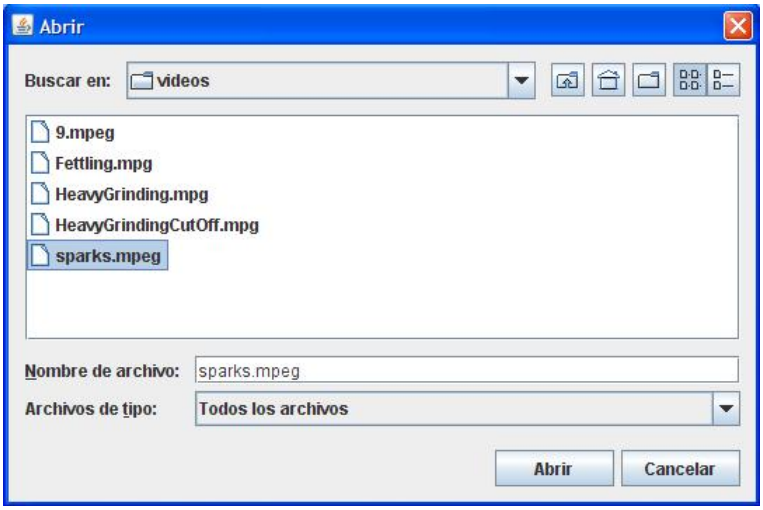

Figura 4.22: Cuadro de diálogo para seleccionar el vídeo que se va a analizar.

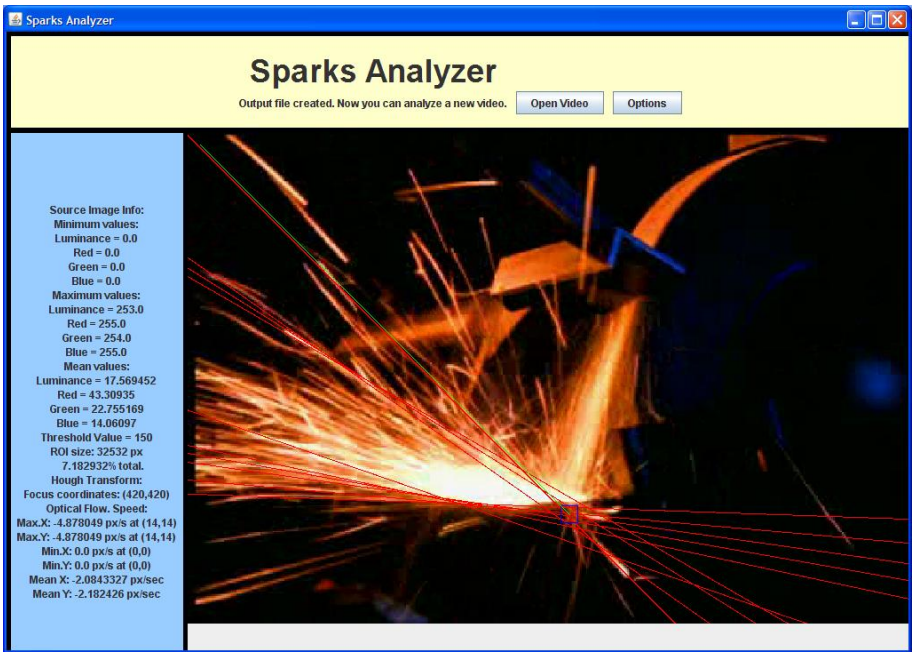

Figura 4.23: Fin del análisis del vídeo.

Cuando el vídeo termina, el programa nos informa de que se ha creado el fichero de salida, y nos ofrece analizar un nuevo vídeo. Esto se muestra en la figura 4.23. La estructura del fichero de salida se explica detalladamente en el Apéndice C. El fichero se guarda en la carpeta 'Outfiles', en el directorio del programa, con nombre "nombre_del_vídeo+data.txt. En este caso, sparksdata.txt.

Podemos acudir entonces al directorio 'images'y comprobar que se han salvado las frames analizadas, en formato JPEG (figura 4.24).

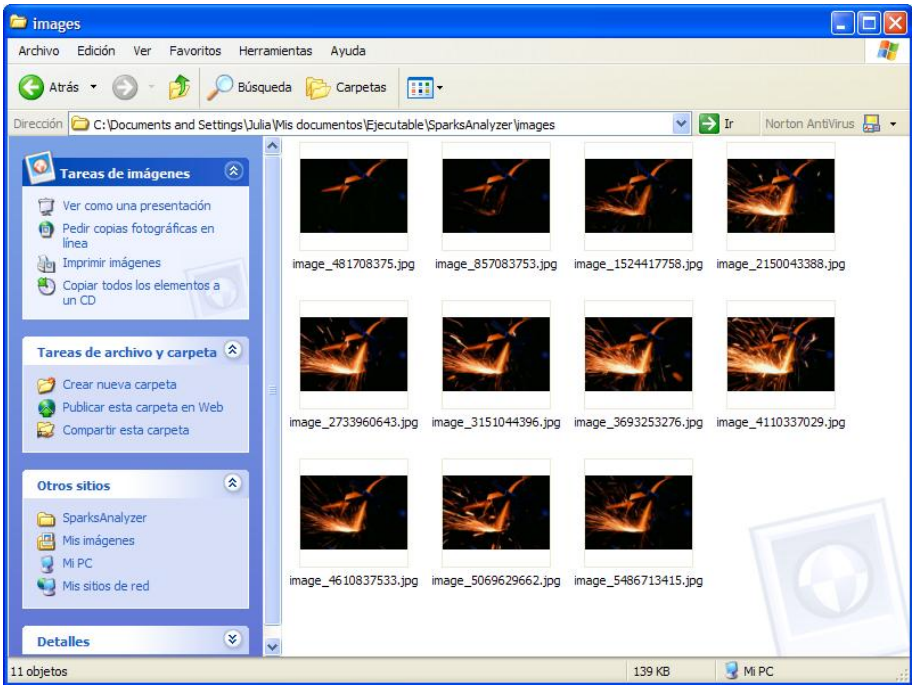

Figura 4.24: Frames salvadas

Capítulo 5

Experimentos y resultados

En este capítulo se va a probar el funcionamiento de los algoritmos diseñados para las aplicaciones que se han desarrollado. Para ello se analizarán distintas imágenes y vídeos en diversas condiciones, y con varias posibilidades de configuración.
Se comentarán los resultados obtenidos y se sacarán conclusiones respecto al funcionamiento de los algoritmos y de los parámetros de configuración.

Con las dos aplicaciones desarrolladas se pueden analizar gran cantidad de imágenes y vídeos, ese es su objetivo. En este capítulo se recogen algunos experimentos, para ilustrar la conveniencia de la elección de ciertos valores en los parámetros de configuración, pero las posibilidades son mucho más amplias.

5.1. Sustracción del fondo de la imagen

Tanto SparksSolver como SparksAnalyzer sustraen una imagen de fondo a la imagen que se está analizando. SparksSolver permite ver el resultado en pantalla.

En la figura 5.1 se muestra el funcionamiento de la operación. En este caso el resultado es satisfactorio, sólo quedan las chispas y un leve ruido donde antes estaba la muela. El motivo de este buen funcionamiento es que la cámara ha permanecido inmóvil entre ambas imágenes.

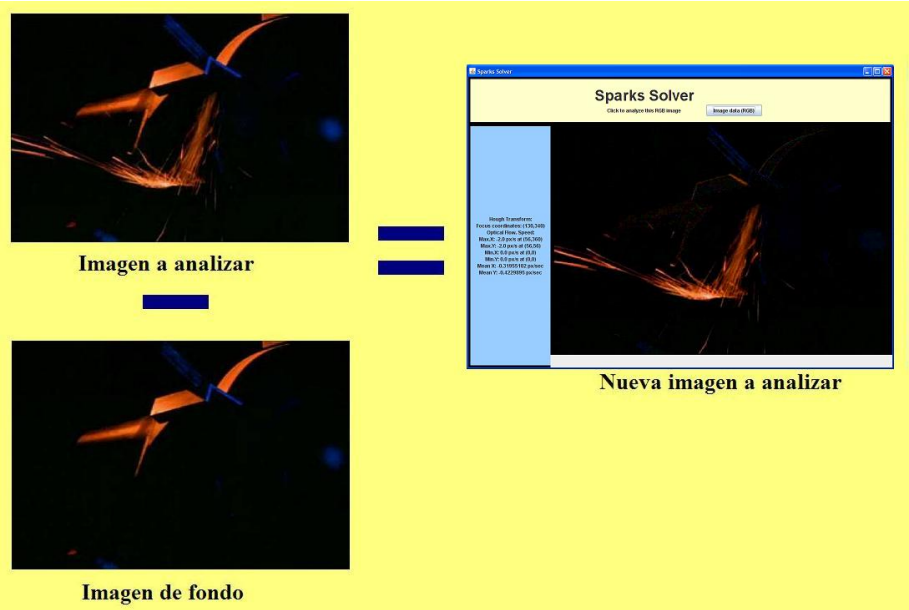

Figura 5.1: Sustracción correcta.

Si se aplica el algoritmo a las imágenes Rolls.jpg y RollsBG.jpg, se obtiene lo que aparece en la figura 5.2. La imagen de fondo contenía chispas, y además hay un pequeño desplazamiento de la cámara entre las dos imágenes, ya que se obtuvieron con la cámara en mano. Esto provoca que la sustracción no sea totalmente efectiva, ya que queda ruido en la imagen resultante. Esto a su vez influye de forma significativa en el funcionamiento de los algoritmos que se le aplican. En particular, el del flujo óptico, ya que hay más diferencia entre las imágenes que la que hay entre las escenas reales. El gráfico del flujo óptico aparece en la figura 5.3, y se aprecia mucho más movimiento en él del que realmente existe.
Pero este ejemplo ilustra un resultado importante: se pueden eliminar fuentes de iluminación estáticas que aparezcan en la imagen. El foco de la parte superior izquierda desaparece. De este modo no afectaría a la discriminación de las chispas del resto de la imagen, que se lleva a cabo por intensidad luminosa.

Otro ejemplo de sustracción correcta por la adquisición de imágenes en condiciones adecuadas se muestra en la figura 5.4. Las imágenes utilizadas son heavygrindingBG.jpg y heavygrinding480000000.jpg, ambas proporcionadas con SparksSolver.

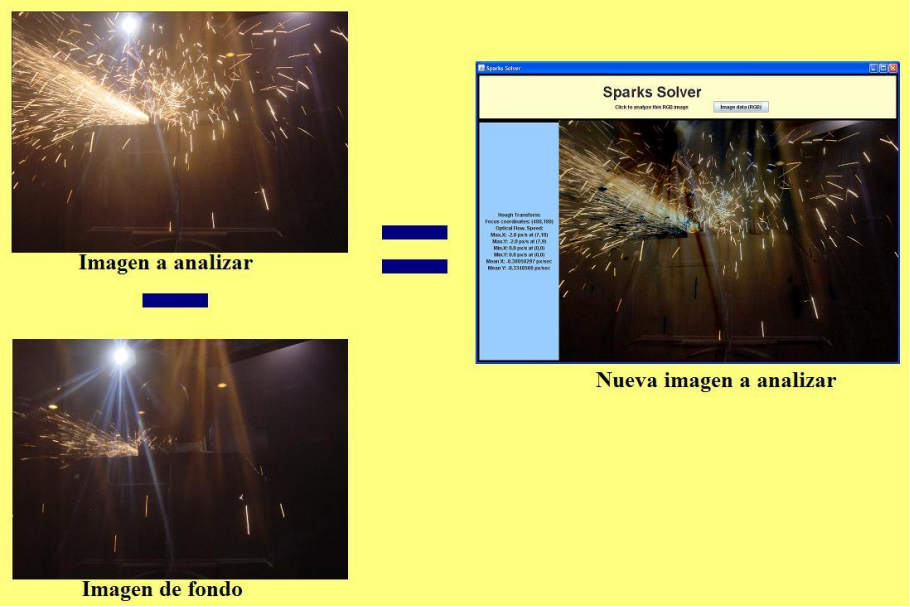

Figura 5.2: Sustracción con desplazamiento de la cámara.

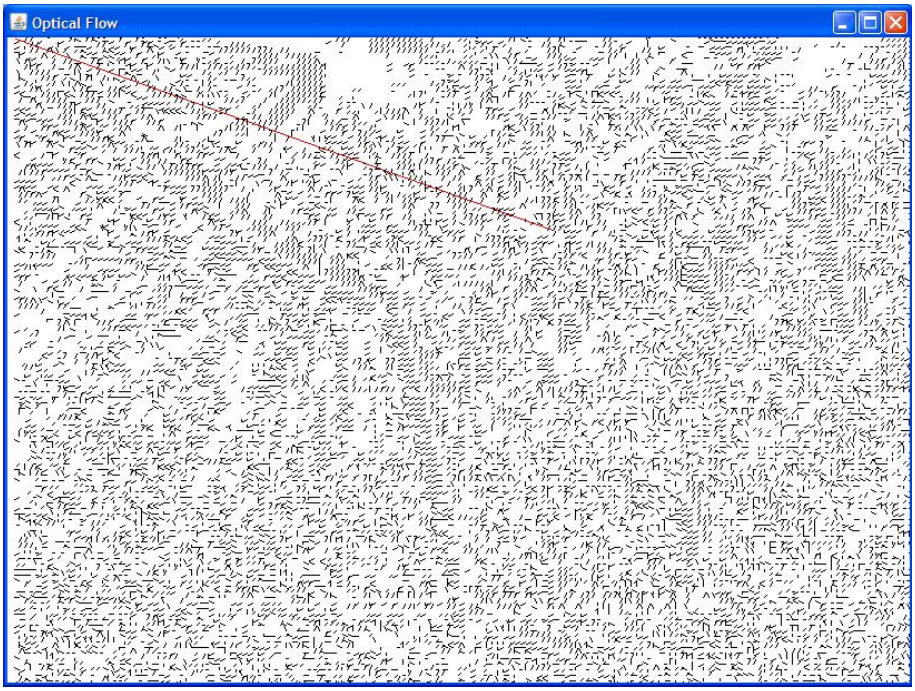

Figura 5.3: Flujo óptico tras una sustracción inadecuada.

Figura 5.4: Sustracción correcta.

Este método tiene ventajas e inconvenientes. Por una parte facilita la aplicación de algoritmos complejos y reduce su coste computacional. Además elimina elementos que no son de interés, o que estorban al análisis. A cambio exige ciertas condiciones en la imagen de fondo. Ésta debe ser lo más parecida posible a una imagen de trabajo, sobre todo en lo que se refiere a la posición de la cámara y del brazo robótico. Esto limita la libertad de actuación, con lo cual hay que llegar a un compromiso entre calidad del análisis y libertad de movimientos.
Por otra parte, para facilitar la labor de estos programas de visión, convendría que el lugar de trabajo fuera lo más uniforme posible en cuanto a color. Lo ideal sería que tanto el lugar como el robot fueran negros. De este modo los resultados serían mejores ya que al reducir los elementos que no tienen interés, la sustracción los eliminaría casi totalmente.

5.2. Cómputo del área de chispas

Para evaluar el tamaño de las chispas de una imagen, se ha optado por umbralizar las imágenes en intensidad. Así, se considera que pertenecen a chispas aquellos píxeles con más de un cierto nivel de luminancia. Dadas las características del proceso de rectificado, cuando hay chispas, éstas se corresponden con los niveles más altos de intensidad de la imagen, de manera que para segmentarlas del resto de la imagen basta con escoger adecuadamente el valor del umbral.

Las dos aplicaciones desarrolladas permiten al usuario seleccionar un valor de umbral para computar el área de las chispas. Cuando no se sabe qué valor elegir, el primer paso es realizar varios análisis probando distintos valores con SparksSolver. A continuación se podrán analizar las numerosas imágenes que componen un vídeo configurando en SparksAnalyzer el valor obtenido anteriormente.

En el caso de las imágenes de la figura 5.4, se han probado varios valores: 50, 100, 150 y 200. Cabe recordar en este momento que en Java los píxeles de las imágenes con bandas RGB son números enteros compuestos por ternas de enteros entre 0 y 255. Una imagen en blanco y negro tiene una sola banda con valores de intensidad comprendidos entre 0 y 255. Los resultados obtenidos con los valores de umbral probados se muestran en la figura 5.5. Parece entonces que el valor más adecuado se encuentra entre 50 y 100. Las áreas calculadas para cada zona aparecen en el fichero de salida:

Umbral=50,

```
ROI size: 23773 px
5.162432% total.
```

Umbral=150,

```
ROI size: 4733 px
1.0277959% total.
```

Umbral=100,

```
ROI size: 11442 px
2.4846907% total.
```

Umbral=200,

```
ROI size: 855 px
0.18566775% total.
```

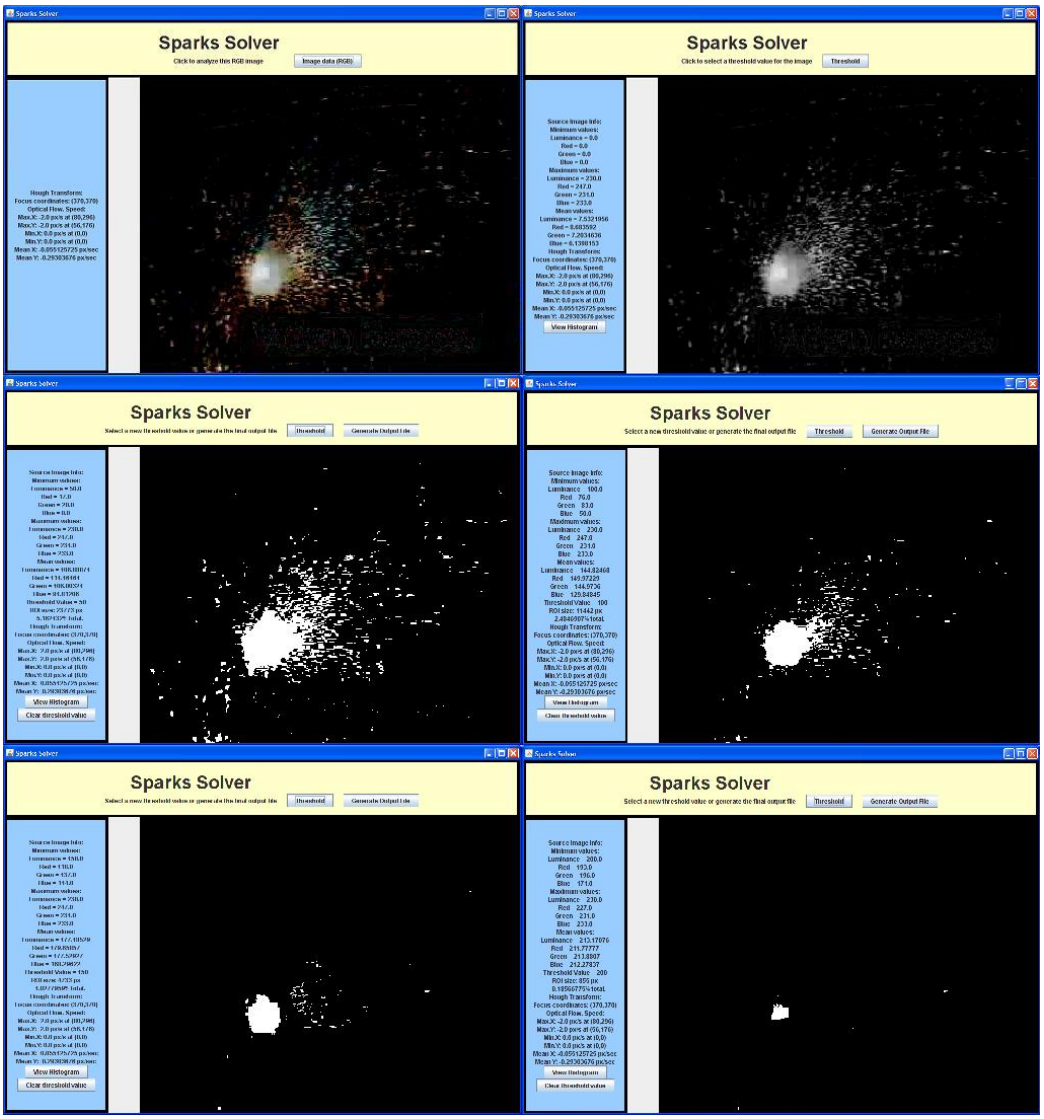

Figura 5.5: Umbralización con distintos niveles de intensidad. De izquierda a derecha y de arriba a abajo: Imagen a analizar, imagen en blanco y negro, umbral 50, umbral 100, umbral 150, umbral 200.

5.3. Transformada de Hough

Como se explicó en el Capítulo 2, el algoritmo de la transformada de Hough consiste básicamente en los siguientes pasos:

- Convertir la imagen en color a blanco y negro.
- Detectar los bordes de la imagen, con un operador como el de Sobel.
- Binarizar la imagen de bordes.
- Aplicar la transformada de Hough.
- Reducir las nubes de puntos que corresponden a una misma línea.
- Representar las líneas obtenidas.

El algoritmo implementado en las aplicaciones desarrolladas introduce variaciones para mejorar su funcionamiento con imágenes como las que nos ocupan. Estas mejoras se describen a continuación.

5.3.1. Mejora del algoritmo clásico

La primera mejora es que la binarización se realiza con histéresis y con umbrales dinámicos. Se aplica en primer lugar un umbral elevado, y a los vecinos de los puntos que lo superen, un umbral de menor valor.
Los valores de los umbrales no son fijos. Para mayor versatilidad del algoritmo, se asignan de forma dinámica con una tabla, según el histograma de la imagen.
De esta forma , al aplicar primero un umbral elevado, se reduce la presencia de puntos aislados, reduciendo por consiguiente el ruido.

Además se ha modificado el primer paso del algoritmo. En lugar de realizar la detección de bordes a la imagen en blanco y negro, se ha realizado sobre la banda azul de la imagen en color. Este cambio se ha basado en el estudio de las características de las imágenes con chispas. Se ha descubierto que la banda azul contiene menos elementos que no sean chispas. Las bandas roja y verde presentan el ruido que queda tras la sustracción de la imagen de fondo a la imagen a analizar. La comparativa se muestra en la figura 5.6.
Como se ha señalado en la sección 5.1, los elementos que quedan en la imagen

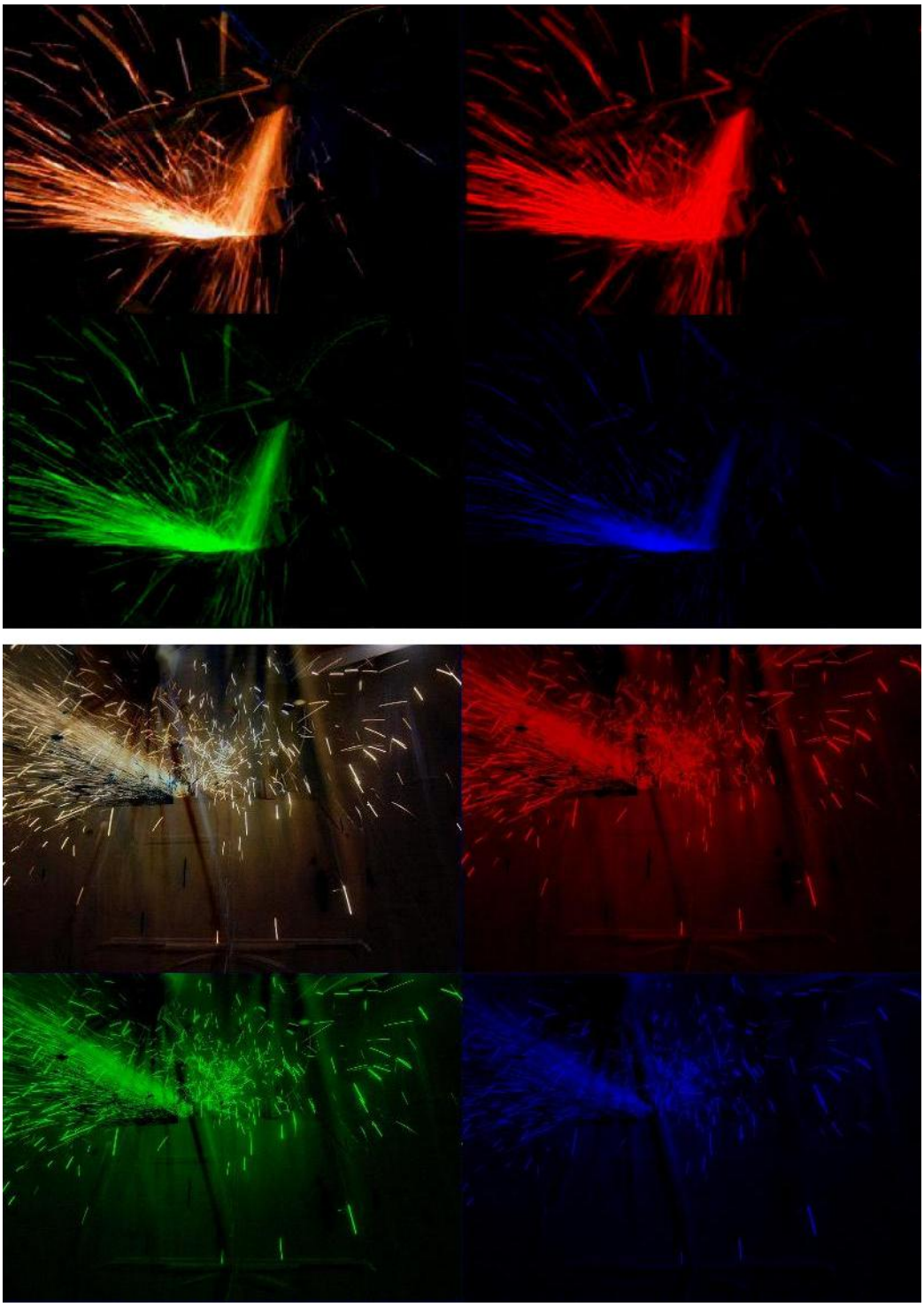

Figura 5.6: Comparación entre el contenido de las bandas de la imagen en color, con sustracción de buena calidad (arriba) y mala (abajo).

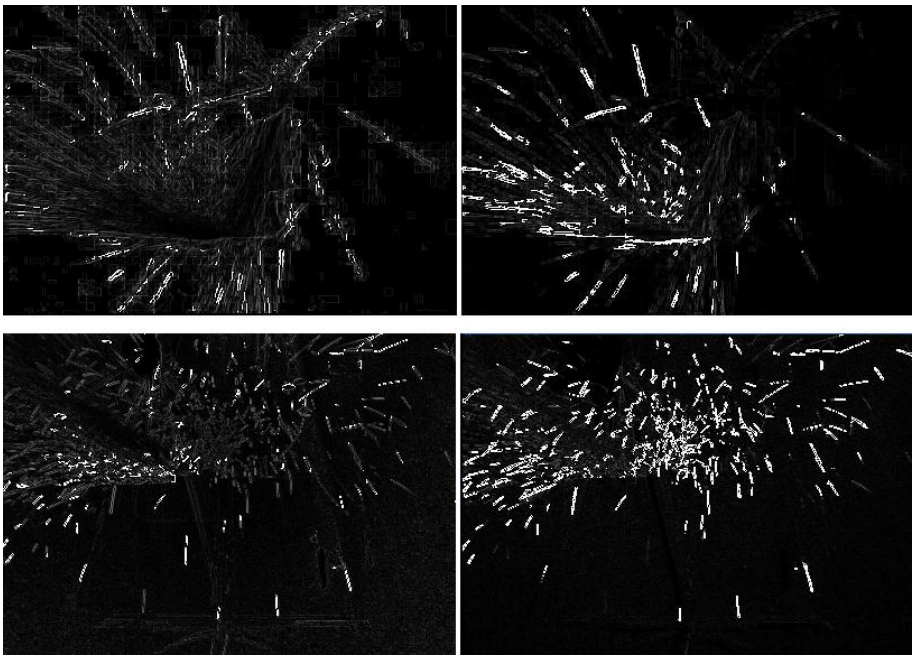

Figura 5.7: Comparación entre la detección de bordes de una imagen en blanco y negro (izquierda) frente a la banda azul de una imagen en color (derecha).

tras la sustracción del fondo son prácticamente inapreciables a nuestros ojos, pero provocan cierta cantidad de ruido en la imagen. Podría pensarse que este ruido no afecta al funcionamiento de la detección de bordes mediante el operador de Sobel, pero se hecho se detectan elementos del fondo. Esto es más significativo cuanto peor haya sido la sustracción del fondo. En la figuras 5.7 puede apreciarse la diferencia en la detección de bordes si se utiliza la banda de luminancia o la banda azul, para las imágenes anteriores.

Al eliminar ruido y elementos que no son de interés, se logra una gran mejora la aplicación del algoritmo de la transformada de Hough para la detección de la trayectoria de las chispas.

5.3.2. Número de líneas detectadas

En el algoritmo de la transformada de Hough es posible configurar el número de líneas que se va a detectar en la imagen. Se muestran las líneas con mayor número de puntos. El número de líneas se debe elegir lo suficientemente grande como para obtener las trayectorias significativas, pero no tanto como para detectar líneas que no se corresponden a chispas.

En la figura 5.8 se muestran los resultados de varios análisis con SparksSolver, con tamaño de la zona del foco 20px·20px y radios de los clusters para una misma línea igual a 10. Se ha probado a detectar 1, 5, 10 y 15 líneas. De los resultados se deduce que cinco líneas son insuficientes, y 15 son demasiadas ya que aparecen líneas que no se corresponden a trayectorias de las chispas. Con 10 líneas se identifican prácticamente todas las trayectorias, por tanto para esta imagen se deberían solicitar entre 5 y 10 líneas.

En la figura 5.9 se ha analizado otra imagen, con tamaño de la zona del foco 20px·20px y radios de los clusters para una misma línea igual a 10. Se ha probado a detectar 1, 5, 10, 15 y 20 líneas. Como puede verse, una línea es insuficiente, y 15 son demasiadas ya que aparecen líneas que no se corresponden a trayectorias de las chispas. Con 10 líneas quedan algunas trayectorias por identificar, con lo cual lo ideal en estas condiciones sería solicitar un número entre 10 y 15.

5.3.3. Radio de los clusters

Además las líneas detectadas dependen del radio de los clusters configurado. En la figura 5.10 se muestra una comparativa entre los resultados análisis, con tamaño de la zona del foco 20px·20px, 10 y 15 líneas, y radios de los clusters para una misma línea 5, 10 y 20. Puede apreciarse que con radios 5 y 20, el foco no se detecta correctamente, mientras que con 10 el resultado es satisfactorio. En cuanto a las trayectorias detectadas, con 10 líneas y radio 10, el resultado es mejor que con el mismo radio y 15 líneas. Con 15 líneas y radio 10 se obtiene prácticamente el mismo resultado que con 10 líneas y radio 20. En este caso por tanto, el mejor resultado se obtiene con 10 líneas y radio 10. Y este sería el valor de los parámetros que se debería introducir en los análisis con SparksAnalyzer.

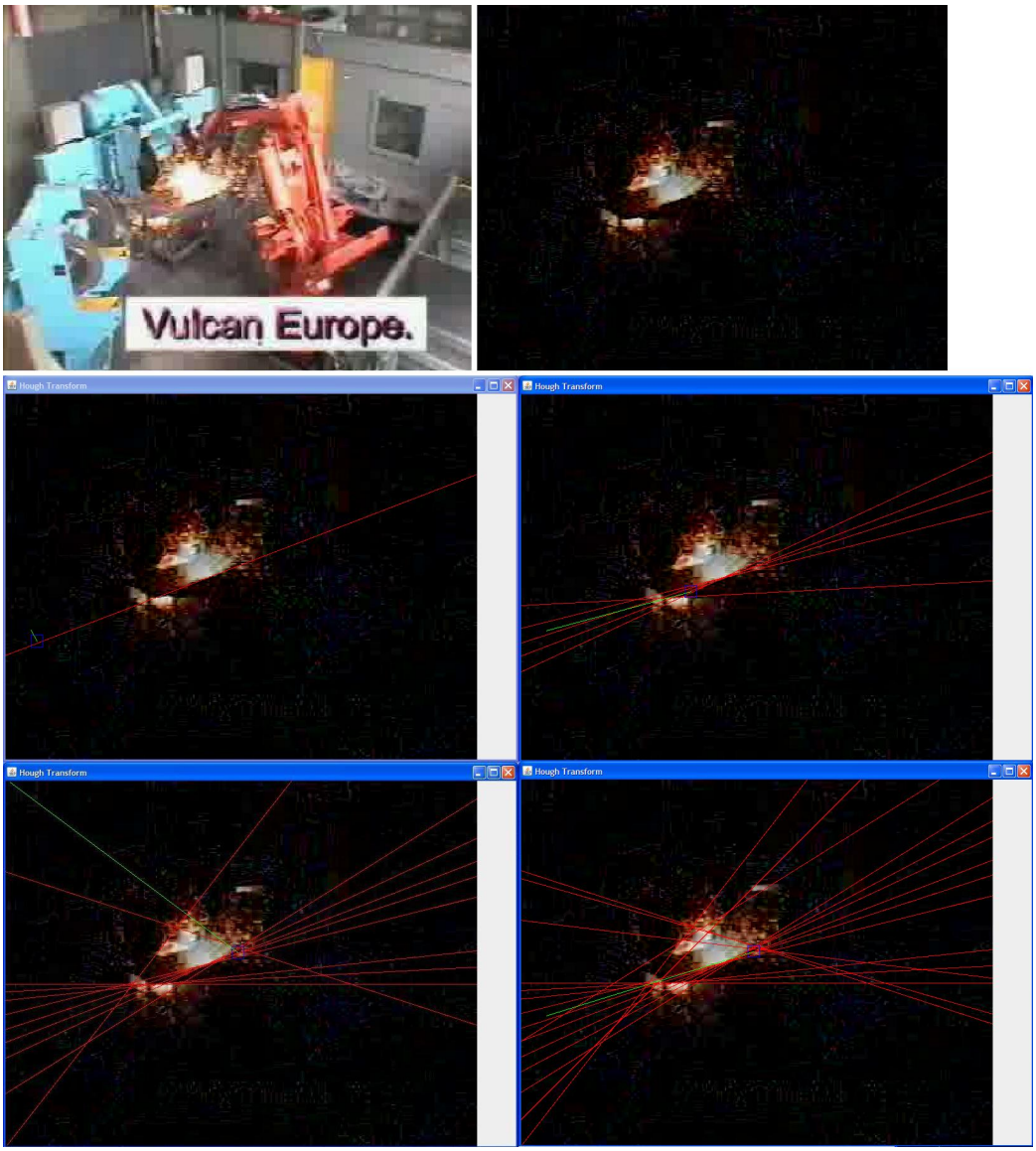

Figura 5.8: Comparativa entre las trayectorias detectadas según el número solicitado. De izquierda a derecha y de arriba a abajo: imagen original, imagen sin fondo, y 1, 5, 10 y 15 líneas detectadas.

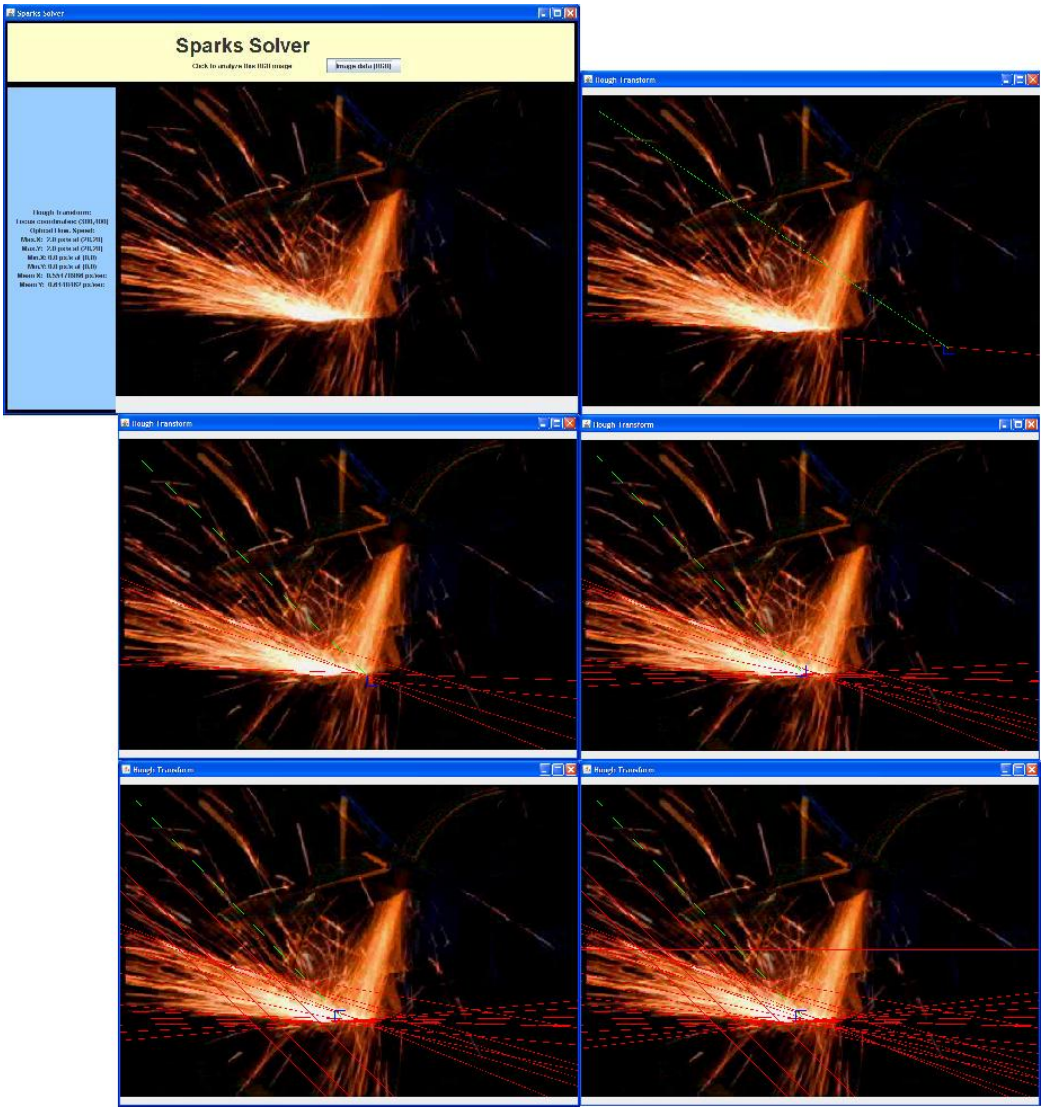

Figura 5.9: Comparativa entre las trayectorias detectadas según el número solicitado. De izquierda a derecha y de arriba a abajo: imagen original, y 1, 5, 10, 15 y 20 líneas detectadas.

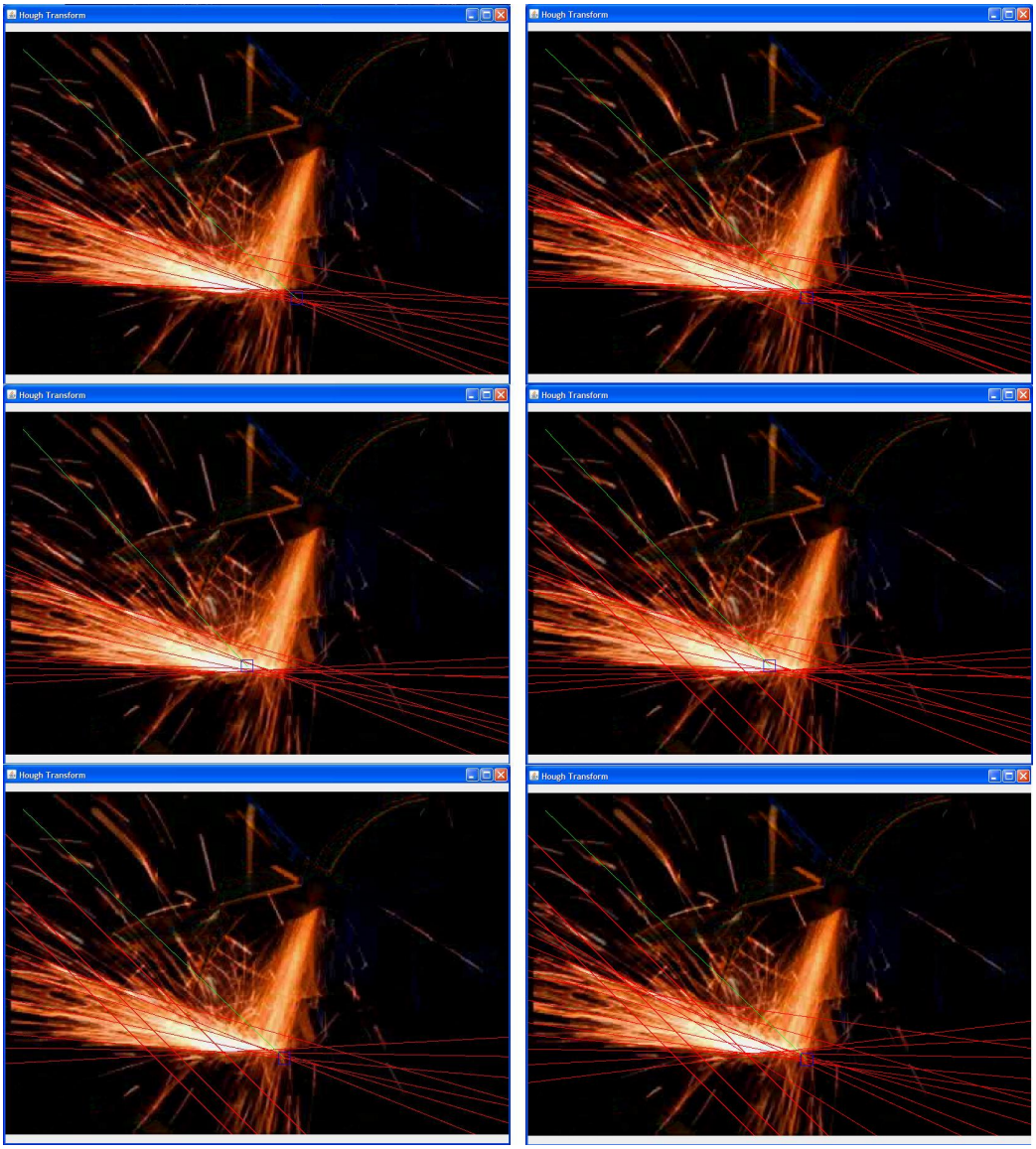

Figura 5.10: Comparativa entre las de trayectorias detectadas según el radio de las nubes de puntos que corresponden a una sola línea. Izquierda: 10 líneas y radios 5, 10 y 20. Derecha: 15 líneas y radios 5, 10, 20.

5.4. Identificación del foco de las chispas

5.4.1. Precisión

La identificación del foco de las chispas se lleva a cabo a partir de los resultados de la transformada de Hough. Según las trayectorias obtenidas, se identifica la zona de la imagen por la que pasan más líneas, y esa se identifica como la región del foco. Por todo esto, el resultado depende de la calidad del funcionamiento de la transformada de Hough. Si se detectan líneas que no se corresponden con trayectorias de chispas, se puede falsear el resultado de la identificación.

En la figura 5.10 se aprecia la dependencia de la identificación del foco con los resultados de la transformada de Hough. Se muestran análisis con tamaño de la zona del foco 20px·20px, 10 y 15 líneas, y radios de los clusters para una misma línea 5, 10 y 20. Puede apreciarse que con radios 5 y 20, el foco no se ha identificado correctamente debido a que las trayectorias detectadas confluyen en otra zona.

5.4.2. Tamaño de la región del foco

Además de la dependencia del número de líneas detectadas y del radio de las nubes de puntos que corresponden a una misma línea, existe otro parámetro que influye en la identificación: el tamaño de la región del foco.

En primer lugar, según el tamaño de la región y su localización incidirán en ella unas determinadas líneas. Con lo cual la zona con mayor número de puntos de líneas puede variar de una ejecución a otra según el valor del parámetro 'Focus zone area in px·px". En la figura 5.11 se muestran los resultados de identificaciones con tamaños 10x10, 20x20, 30x30 y 40x40. Los resultados son mejores con los tamaños 10x10 y 20x20.

En segundo lugar hay que tener en cuenta también el verdadero tamaño de la zona del foco en la imagen. Es decir, para una imagen tomada desde lejos, en la que la zona de trabajo tenga una dimensión de 100x100 píxeles, habrá que elegir un tamaño de la región menor que para otra de 400x400. Así los resultados serán más precisos.

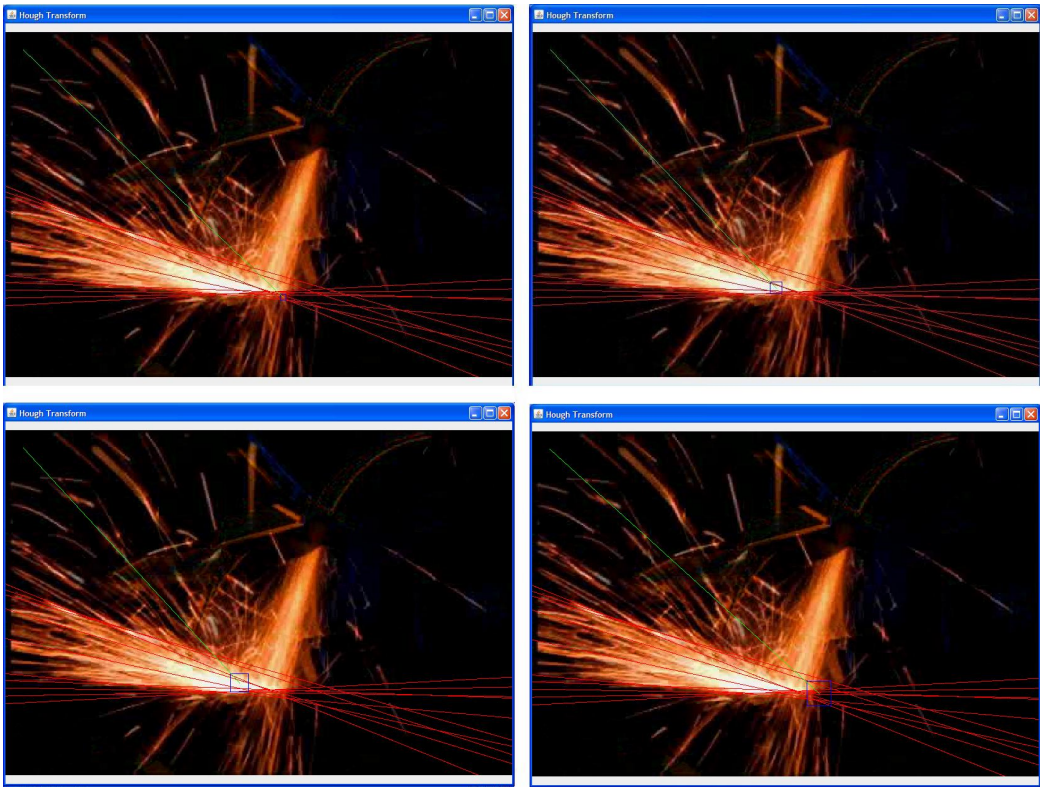

Figura 5.11: Identificación de la región del foco según el tamaño de la región. Tamaños: 10x10, 20x20, 30x30 y 40x40 píxeles.

5.5. Flujo óptico

El funcionamiento del algoritmo para el cálculo del flujo óptico es igual en las dos aplicaciones desarrolladas. Sin embrago varía la presentación de los resultados. En SparksSolver se presenta en una ventana el gráfico resultante del cálculo de las velocidades de los puntos de la imagen y se integran los resultados con los de la transformada de Hough. En SparksAnalyzer, se utiliza la información obtenida para comprobar la corrección del cálculo del foco de las chispas, pero no se presenta el gráfico para disminuir el coste computacional.

En ambas aplicaciones se puede configurar la resolución del análisis escalando o no las imágenes analizadas. Esto se selecciona con el parámetro 'Resolution'. Se pueden multiplicar las dimensiones de la imagen por 1, 0.5, 0.25 y 0.125. De nuevo la idea es ajustar en SparksSolver el menor valor para el cual los resultados son aceptables, e introducir este valor en la configuración de SparksAnalyzer.

En la figura 5.12 se muestran los resultados del algoritmo del flujo óptico para valores de 'Resolution'0.125, 0.25, 0.50 y 1, respectivamente. La línea roja une el punto de máxima velocidad con el foco de las chispas. Como puede observarse, el punto de máxima velocidad apenas varía con el cambio de resolución, con lo que se consigue disminuir apreciablemente el coste computacional sin sacrificar apenas precisión.

5.5.1. Parámetros

En cuanto a los parámetros de configuración, se puede modificar el tamaño de la ventana de la imagen de la que se busca coincidencia, y el desplazamiento permitido en la búsqueda.

Se van a ilustrar varios experimentos variando estos parámetros. En la figura 5.13 se muestran los gráficos obtenidos con resolución 0.5, tamaños de ventana de 2x2, 4x4 y 6x6, de izquierda a derecha; y con desplazamientos 2, 4 y 6 de arriba a abajo.

La velocidad de ejecución es menor cuanto menor es el tamaño de la ventana, y menor el desplazamiento permitido. Pero las diferencias entre los gráficos apenas son apreciables.

Figura 5.12: Gráficos de flujo óptico según resolución.

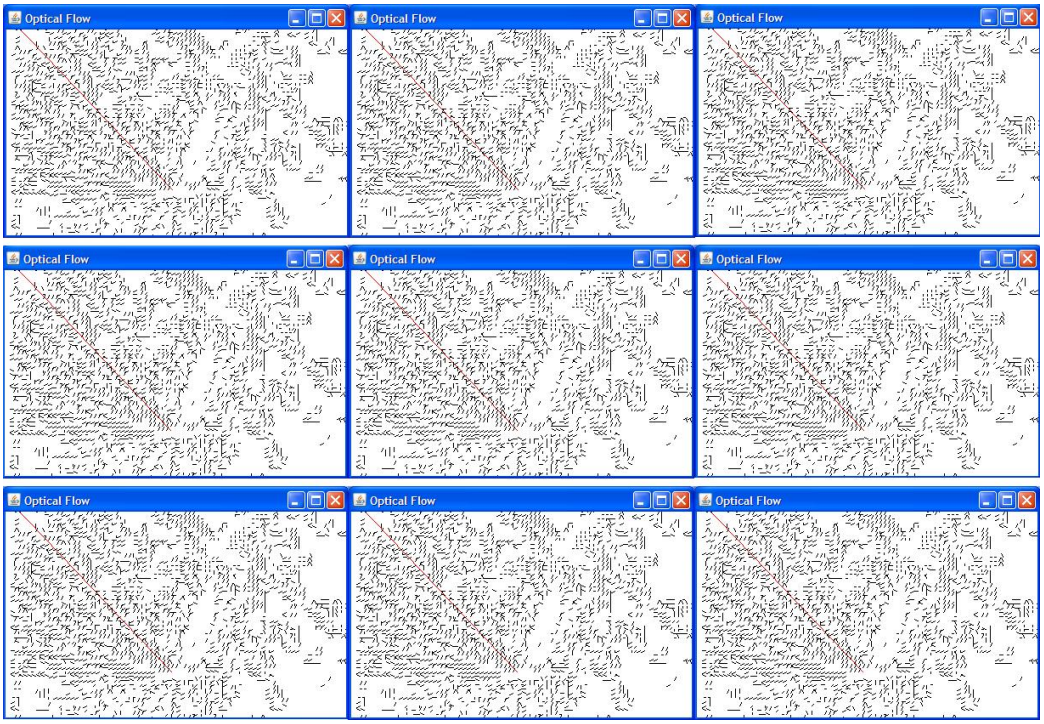

Figura 5.13: Variación de los parámetros del flujo óptico. De izquierda a derecha, tamaños de ventana de 2x2, 4x4 y 6x6 píxeles. De arriba a abajo, desplazamientos 2, 4 y 6 píxeles.

5.5.2. Consideraciones

La calidad de las imágenes repercute en el funcionamiento del algoritmo. En la figura 5.14 aparece un ejemplo. La imagen es de baja resolución, y aunque la sustracción del fondo es satisfactoria, existen diferencias apreciables entre la imagen de fondo y la que se está analizando, porque los píxeles son 'demasiado grandes'y se tiene ruido en la sustracción. Esto provoca que aparezca movimiento donde realmente no lo hay.

5.6. Integración del flujo óptico y la transformada de Hough

5.6.1. Funcionamiento

Los datos procedentes de los algoritmos del cálculo de la transformada de Hough y del flujo óptico se integran para determinar si la detección del foco de las chispas se ha llevado a cabo correctamente.
Para ello se calcula y se traza una línea entre el punto de mayor velocidad, procedente del flujo óptico, con el centro del foco detectado. Esa línea se representa gráficamente sobre la imagen con las trayectorias detectadas tras la transformada de Hough, para dar una idea visual de dónde se encuentra el punto de máxima velocidad.
En la figura 5.15 se muestran dos resultados obtenidos. En ambos la línea trazada está en un extremo del chorro de chispas.

5.6.2. Discriminación de focos erróneos

Para estimar si la identificación de la región del foco se ha llevado a cabo correctamente, se calculan las coordenadas polares de la línea que une el punto de máxima velocidad con el foco, y se comparan con las de las trayectorias obtenidas en el algoritmo de Hough. Si el valor de theta calculado no se encuentra en el rango delimitado por los valores extremos de theta de las líneas - con tolerancia dada por el parámetro de configuración 'distance'-, el programa imprime un aviso, ya que no se tiene la certeza de haber identificado correctamente el foco de las chispas. Esto no quiere decir que la identificación no sea correcta, sino que no se tiene la certeza de que lo sea.

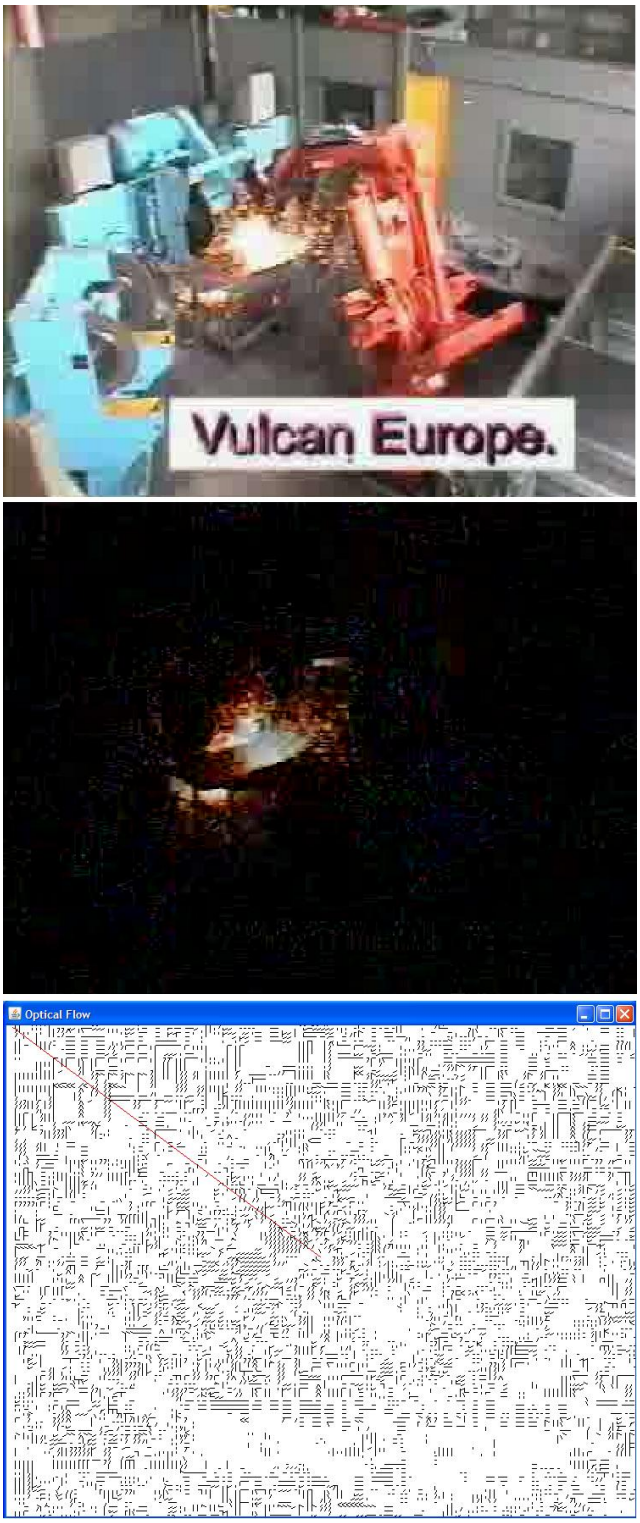

Figura 5.14: Ruido en el gráfico del flujo óptico de una imagen de baja resolución y mala calidad.

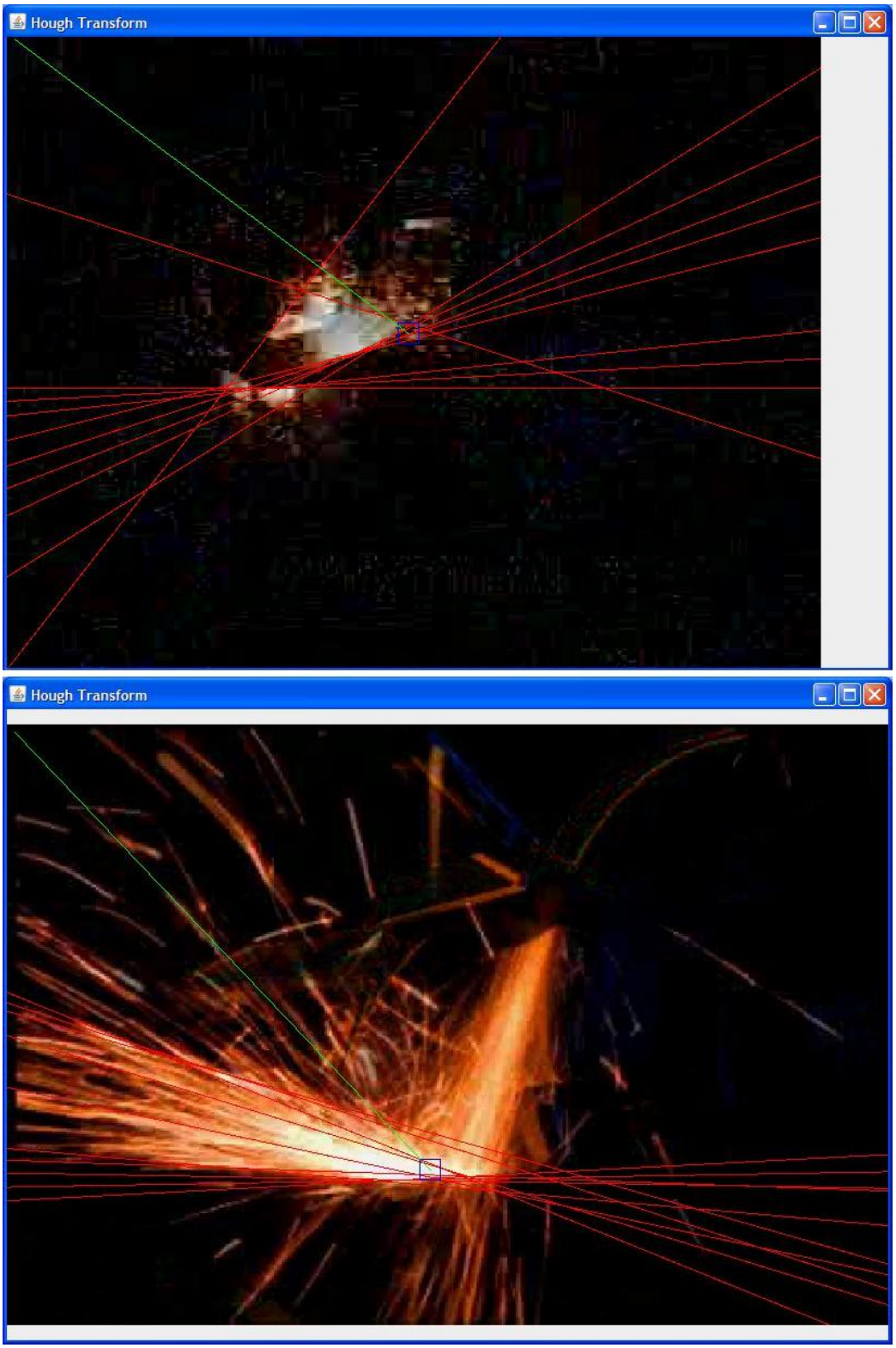

Figura 5.15: Resultado de la integración del flujo óptico y la transformada de Hough.

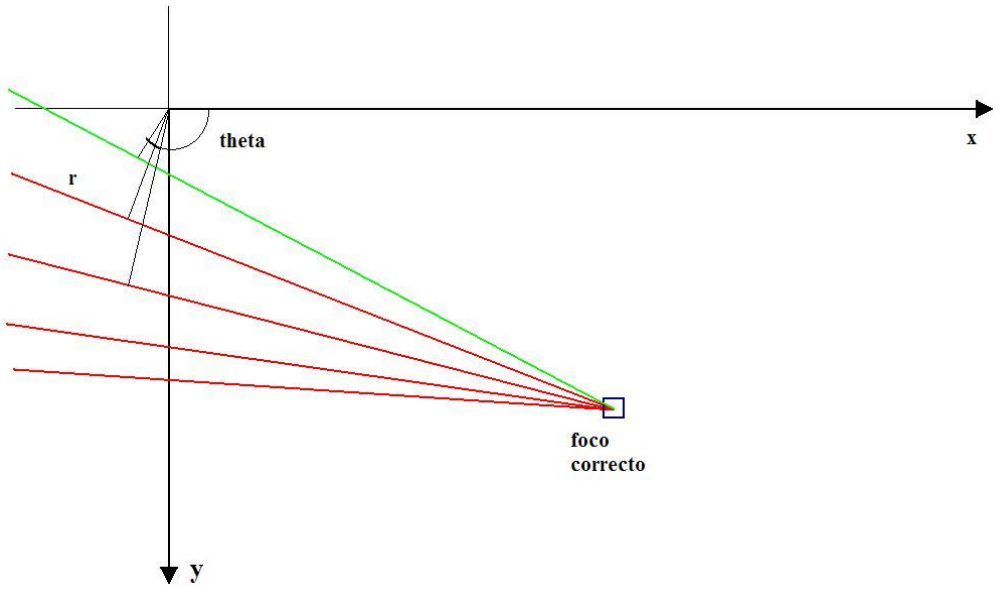

Figura 5.16: Variación de las coordenadas r y theta entre líneas procedentes del mismo punto.

Por ejemplo, en la figura 5.15, se detecta correctamente el foco en ambas ocasiones, pero en ambas se da el aviso debido a que el parámetro theta se aleja de las trayectorias detectadas. En esto influye también el número de líneas detectado, ya que si se tiene mayor número de líneas, es más probable que la línea verde esté entre ellas.

5.6.3. Consideraciones de diseño

Para diseñar esta comprobación, nos hemos basado en la observación de la coordenada theta. Esto se debe a que es la que varía menos entre las trayectorias que parten de un mismo foco. Esto se ilustra en la figura 5.16.

En la figura 5.17 se ilustra la variación de theta entre líneas procedentes de dos puntos diferentes.

Además se ha optado por unir el punto de máxima velocidad con el foco, y no por prolongar el vector de la velocidad de dicho punto hacia el foco. Esto se ha debido a que en las imágenes analizadas se obtenía que el punto de máxima velocidad se correspondía con una chispa en su caída hacia el

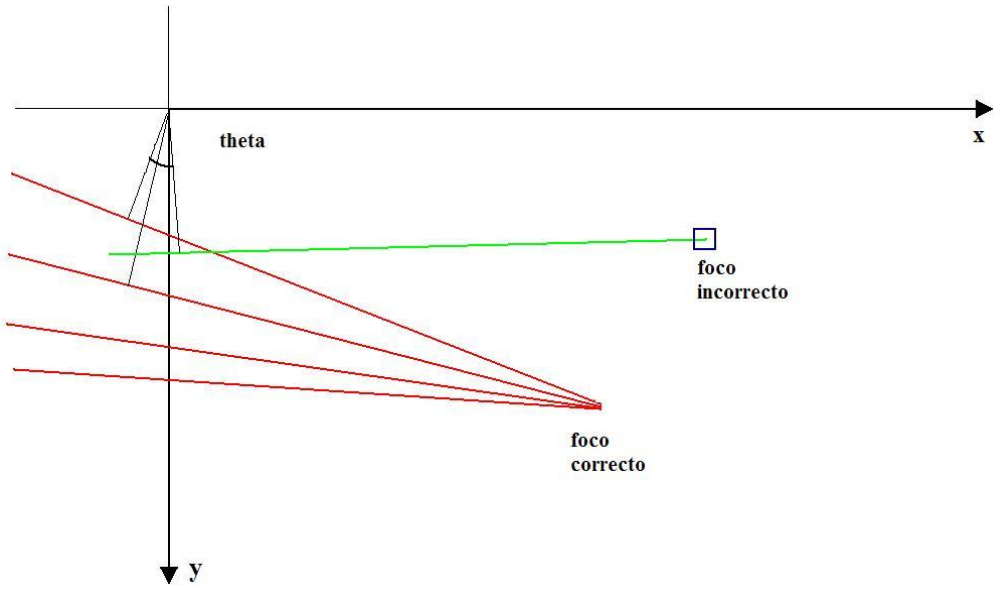

Figura 5.17: Variación de la coordenada theta entre líneas procedentes de puntos diferentes.

suelo. Por eso la prolongación de su velocidad no nos llevaba al foco, y se desechó esa opción.

5.7. Análisis de vídeo

SparksAnalyzer permite configurar las características del trabajo con vídeo, además de la precisión del algoritmo del flujo óptico. Con esto se consigue regular la velocidad del análisis y su precisión.

Como se ha señalado anteriormente, el algoritmo del flujo óptico tiene un gran coste computacional, lo que hace que el funcionamiento de SparksAnalyzer se ralentice. Por tanto si se desea conseguir mayor velocidad se puede reducir la resolución de las imágenes con las que trabaja el algoritmo.
También se puede configurar la tasa de procesamiento de las frames que llegan del flujo de vídeo. Por ejemplo si el vídeo tiene una tasa de 30 frames por segundo, se puede optar por procesar una de cada 10. Esto depende del objetivo del análisis. Para un análisis de entrenamiento en el que se busque obtener la mayor cantidad de datos posibles sin importar el tiempo, se

pueden procesar todas las frames al máximo detalle. En cambio si se busca ajustar los parámetros para lograr funcionar en el tiempo de duración del vídeo, habrá que sacrificar detalles y saltar algunas frames.

En cuanto al funcionamiento de los algoritmos, el primer paso que se realiza en el procesamiento de las frames es la sustracción del fondo. Como se mencionó en el apartado 5.1, conviene que el fondo sea estático. Por tanto el funcionamiento de la aplicación será mejor cuanto menos varíe el entorno con respecto al fondo. Por ejemplo, si la primera frame del vídeo, que es la que se toma como fondo, incluye al brazo robótico en una posición que posteriormente va a variar, se obtendrá un resultado peor que si no aparece el brazo. También el resultado será mejor cuanto menos se mueva el brazo.

Por todo lo mencionado anteriormente, la aplicación permite gran flexibilidad de funcionamiento gracias a los numerosos parámetros de configuración, pero la calidad de los resultados depende de la calidad de las imágenes obtenidas y de la localización de los elementos del entorno de trabajo.

Capítulo 6

Conclusiones y líneas de desarrollo futuras

6.1. Conclusiones

6.1.1. Objetivos que se han conseguido

Las dos aplicaciones que se han desarrollado en el proyecto fin de carrera permiten el análisis de imágenes y vídeos de procesos de rectificado de metales.

De los resultados obtenidos se ha concluido que es viable obtener información de imágenes de rectificado. Como se ha expuesto en el Capítulo 5, con los programas SparksSolver y SparksAnalyzer se pueden obtener los siguientes datos:

- Las trayectorias de las chispas.
- La velocidad a la que se desplazan las chispas.
- El punto del que proceden.
- Características de color e intensidad luminosa.
- Área de la zona de chispas.

Todos los parámetros que interviene en el funcionamiento de los algoritmos aplicados son configurables.

Además, el trabajo con vídeo permite obtener todos estos datos de grandes cantidades de imágenes.

La estructuras de datos y los algoritmos que se han diseñado en ambas aplicaciones permiten extender los análisis que realizan a otros tipos de imágenes para otros estudios. Sobre la misma base se pueden ajustar los parámetros de configuración y añadir las funcionalidades que se deseen. Se han diseñado aplicaciones de procesamiento de imágenes versátiles que tienen aplicación en otro tipo de procesos.

6.1.2. Coste computacional y tiempo de ejecución

Dada la complejidad de algunos de los algoritmos implementados, el tiempo de ejecución de las aplicaciones desarrolladas puede crecer demasiado con determinadas configuraciones de los parámetros de funcionamiento.

En el caso de análisis de estudio y entrenamiento, puede que no importe este hecho, y se prefiera realizar análisis más completos y detallados a costa de este aumento en el tiempo de ejecución.

Si se desea reducir el tiempo, habrá que sacrificar detalle en el análisis. El algoritmo crítico en cuanto a tiempo de ejecución en las dos aplicaciones desarrolladas es el del cálculo del flujo óptico. Como se ha mostrado en el Capítulo 5, cuando se reduce la precisión en el funcionamiento, se logra una reducción del tiempo de procesamiento muy superior cualitativamente a la resolución sacrificada, por tanto es una medida que merece la pena.
La otra medida que se puede tomar en SparksAnalyzer es dejar frames sin analizar. En este caso la precisión perdida y el tiempo ganado son similares, por lo que hay que llegar a una solución de compromiso según el aspecto que interés más.

Para optimizar el funcionamiento de SparksAnalyzer en el análisis de vídeo, es importante conocer de antemano los parámetros de configuración más adecuados. Esto se logra deduciendo los valores de estos parámetros a partir de un estudio previo con SparksSolver de las imágenes del proceso que se va a analizar.

6.1.3. Aplicación de la transformada de Hough a imágenes con chispas

Durante el desarrollo del proyecto, se ha llegado a la conclusión de que la transformada de Hough es una herramienta con la que se obtienen excelentes resultados en imágenes con chispas. En el Capítulo 5 se pueden ver algunos resultados experimentales.

Dadas las características de intensidad de estas imágenes, se pueden llevar a cabo los pasos del algoritmo de las transformada de Hough obteniendo buenos resultados intermedios. El operador de Sobel detecta bien los extremos de los chorros de chispas. A esto le sigue una binarización mediante umbral, que en nuestro caso se ha desarrollado mediante dos umbrales dinámicos con histéresis. Esto reduce la cantidad de ruido de la imagen, y por tanto los elementos ajenos al chorro de chispas. Por último se calcula la transformada de Hough y se obtiene información de la trayectoria de las chispas y del foco del que proceden.

El descubrimiento de que la banda azul de las imágenes en color RGB contiene prácticamente sólo chispas, ha servido para mejorar aún más el funcionamiento del algoritmo. Se ha variado la estructura del algoritmo clásico, que opera con la información de intensidad luminosa, para trabajar con la banda azul. Así se reduce la presencia de elementos que no son de interés, y las trayectorias que se obtienen pertenecen a las chispas en mayor cantidad.

El algoritmo desarrollado se puede aplicar con imágenes de características similares a las estudiadas. Esto unido a la flexibilidad en la configuración del programa permite realizar estudios de las imágenes de otros procesos.

6.1.4. Herramientas utilizadas

Los programas que se han realizado se han escrito en lenguaje Java. Esto ha supuesto para mí el aprendizaje y la profundización en la programación orientada a objetos, y en particular en una de las tecnologías más demandadas en la actualidad en numerosos sectores de la ingeniería como es Java.

El procesamiento de imágenes en Java está en desarrollo a la vez que en alza, lo cual ha supuesto un reto inicial. Las librerías que existen para estos fines no son tan completas como las de otros lenguajes de programación como C++. Sin embargo, las carencias de las librerías se han suplido con

funciones propias, que si bien no son tan potentes como las funciones de librería, cumplen su cometido satisfactoriamente.

El software del proyecto Eclipse para el desarrollo de aplicaciones en Java ha demostrado ser potente, flexible, sencillo de utilizar y de gran ayuda. Como se describe en el Apéndice A, la instalación tanto del entorno de desarrollo como de los componentes adicionales y las librerías es simple, extensible e intuitiva. Eclipse es un software que facilita la labor del programador, y permite sacar todo el partido posible a las aplicaciones que se desarrollan con él.

6.1.5. Aspectos enriquecedores

Además de lo mencionado en el apartado anterior acerca de lo aprendido del procesamiento de imágenes con Java, durante el desarrollo del proyecto ha habido otros aspectos enriquecedores.

Por un lado, se han aplicado conocimientos teóricos adquiridos durante el estudio de la carrera. Algunos, como las técnicas de procesamiento de imágenes, directamente y otros, como el tratamiento de señales, indirectamente.

Por otro lado, se ha realizado una investigación multidisciplinar. En el estudio del problema que ha motivado la realización de este proyecto, se han tocado áreas de conocimiento muy variadas.
Inicialmente, para la comprensión del problema inicial del control del proceso de rectificado de metales, se ha estudiado el proceso llevado a cabo por los robots, la forma de controlarlo y los resultados que se obtienen. Además se han adquirido conocimientos del trabajo con metales, impropios de la carrera. Posteriormente, se evaluaron los sensores que se podían aplicar al proceso; y finalmente se ha profundizado en el ámbito del procesamiento de imágenes.

Esta variedad en las materias tratadas ha sido de gran interés y utilidad, y ha servido de motivación para el desarrollo de este proyecto.

6.2. Líneas de desarrollo

6.2.1. Mejora de las prestaciones

Como se ha señalado en la sección 6.1.2, algunos de los algoritmos diseñados tienen gran coste computacional. Esto hace que el tiempo de ejecución de SparksSolver y SparksAnalyzer en determinadas condiciones sea elevado.
Además, las características de Java hacen que su ejecución no sea óptima, dado que es un lenguaje interpretado.

En el futuro este problema se minimizará gracias a la mejora de los procesadores y de la máquina virtual Java (VM). En concreto el aspecto del comportamiento en tiempo de ejecución es una de las principales líneas de desarrollo de Java.

Además algunas de las funciones que se han tenido que elaborar podrían ser sustituidas por funciones de librería, más potentes y óptimas, en futuras versiones de Java Advanced Imaging y Java Media Framework.

6.2.2. Aplicación comercial

Las aplicaciones que se han desarrollado tienen como objetivo el estudio de procesos de rectificado de metal en condiciones variadas. Una mejora que se propone es la evolución de SparksAnalyzer, que procesa vídeo, hacia una aplicación con propósito comercial para la extracción de patrones de comportamiento que se puedan aplicar en la supervisión visual de rectificado robotizado.

Para que esto sea posible, se debe lograr previamente una mejora en las prestaciones, como se ha descrito en el punto anterior.

6.2.3. Estudio con otros sensores

El objetivo general de la línea de investigación en la que se enmarca este proyecto ra lograr la fusión de varios sensores, para mejorar el control del rectificado robotizado de metales.

Una vez realizado el estudio de las imágenes del proceso, resta estudiar la información que se puede obtener de sensores de fuerza en el efector final del robot, y de micrófonos en el entorno de trabajo. La información obtenida deberá ser correlada con la que procede de las imágenes. Así se integrarán los datos de los diferentes sensores, y se podrá diseñar una estrategia de control óptima.

Apéndice A

Instalación del entorno de desarrollo

A.1. Instalación del software Java para desarrolladores

En el Capítulo 4 se explicó la forma de instalar el software Java para usuarios (Java Runtime Environment, JRE). Para desarrollar aplicaciones en Java, es necesario instalar un software adicional para desarrolladores: Java SE Development Kit (JDK)
Este software puede descargarse gratuitamente en la página
java.sun.com/javase/downloads/index.jsp, que se muestra en la figura A.1. En JDK 6 update 2, pulsar Download (figura A.2). Después aceptar el acuerdo de licencia y descargar la versión de nuestro sistema operativo. Una vez descargado, instalamos el software simplemente siguiendo los pasos que nos marca el asistente (figura A.3))

A.2. Instalación de Eclipse SDK

Para instalar el entorno de desarrollo Eclipse, hay que descargarlo gratuitamente de la página www.eclipse.org (figura A.4) Pulsando el botón **Download Eclipse** damos paso a la página de la figura A.5. En ella seleccionamos la versión **Eclipse classic**, y comienza la descarga del archivo

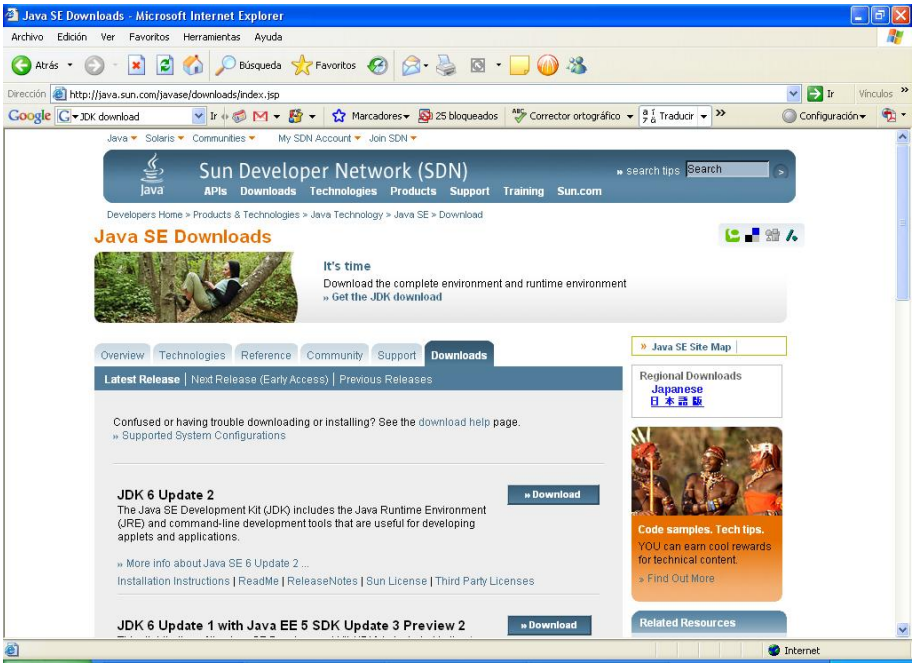

Figura A.1: Página de descarga del software Java para desarrolladores.

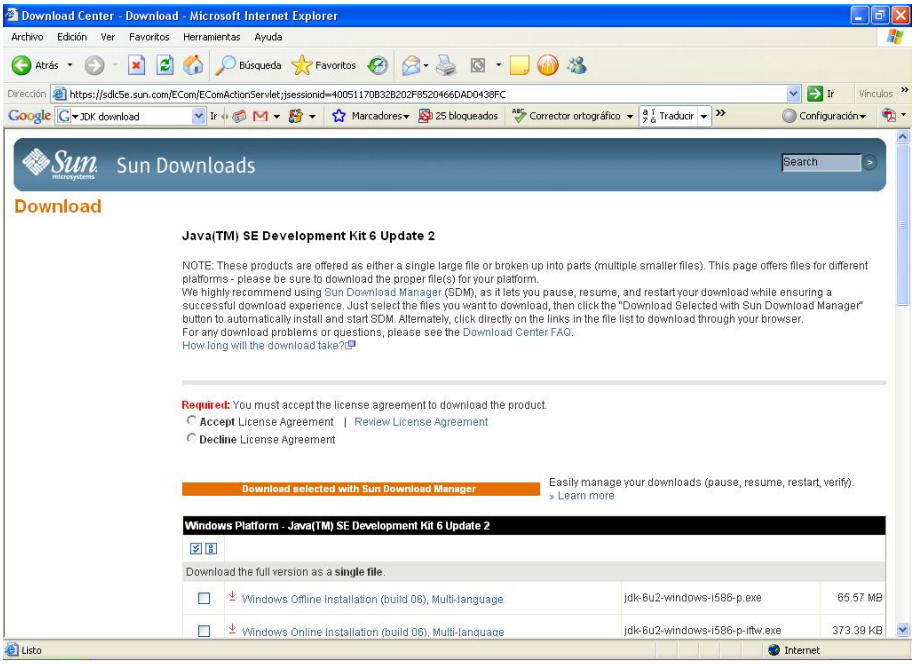

Figura A.2: Descarga de JDK.

Figura A.3: Asistente para la instalación de JDK.

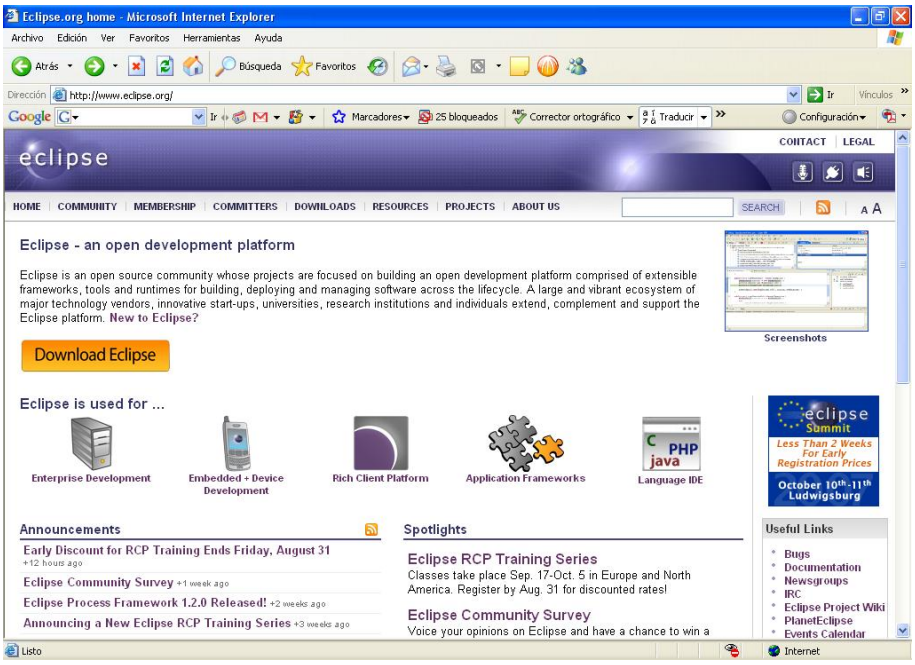

Figura A.4: Página de descarga de Eclipse.

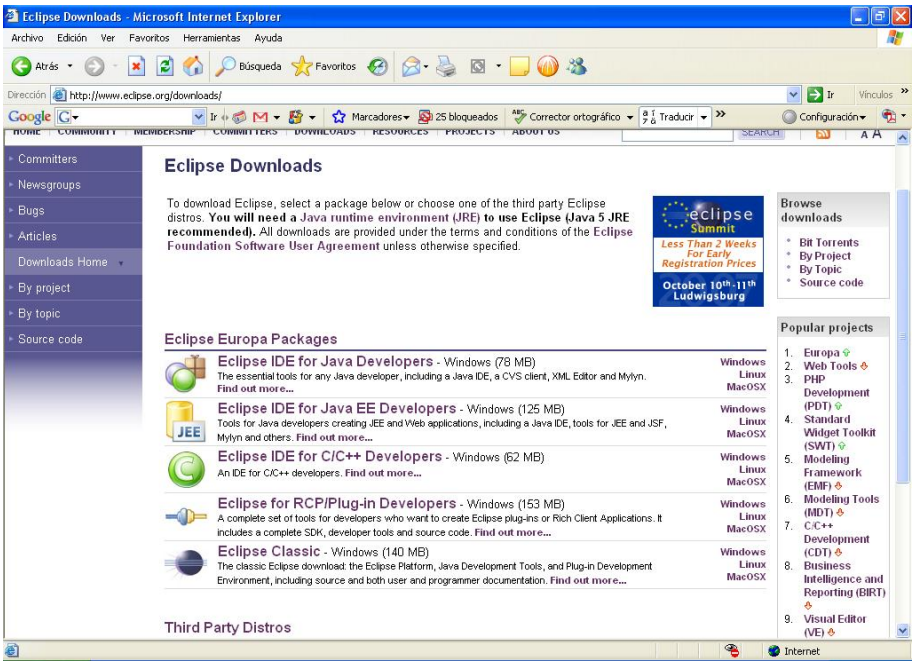

Figura A.5: Página de descarga de Eclipse.

eclipse-SDK-3.3-win32.zip.
Una vez descargado, se descomprime y se copia todo el contenido de la carpeta 'eclipse'(figura A.6) en la carpeta jdk1.6.0_02, en el directorio donde hayamos instalado JDK - normalmente Archivos de programa/Java -.

A.2.1. Plug-ins adicionales

Para el desarrollo de este proyecto se han instalaron adicionalmente los plug-ins:
- GEF runtime 3.1
- VE runtime 1.1.0
- EMF SDO runtime 2.1.0

Con la versión de Eclipse disponible en la actualidad (3.3), hay que instalar: - GEF 3.3 - EMF SDO

Desde la figura A.5, se hace click en **Find out more** y aparece la página de la figura A.7. De ahí se pulsa **GEF**, y en la página de la figura A.8 se

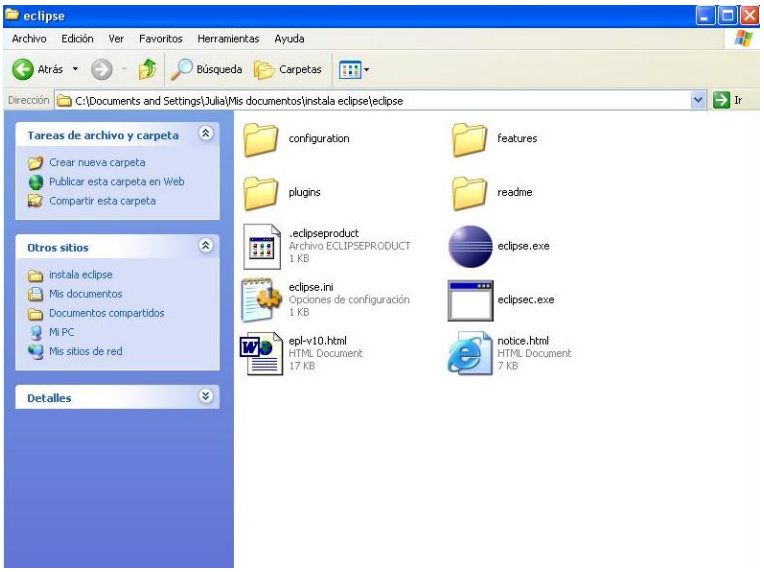

Figura A.6: Contenido de la carpeta 'eclipse'descargada.

selecciona **Downloads - GMF 3.3**, con lo que se descarga el archivo GEF-runtime-3.3.zip. Después se vuelve a la página de la figura A.7 y se pulsa **EMF**, dando paso a la página de la figura A.9, donde se selecciona **SDO - Downloads - All-In-One SDK**.

Cuando estén descargados los dos plug-ins, la instalación es muy simple: Abrir las carpetas, copiar el contenido de la carpeta 'plugins'y de la carpeta 'features', y pegarlo en las carpetas 'plugins'y 'features'del directorio de Eclipse.

Una vez hecho esto, se puede iniciar Eclipse haciendo doble click en el icono eclipse.exe.

A.2.2. Configuración

En primer lugar, al iniciar Eclipse se nos pedirá que seleccionemos un directorio donde guardar el workspace (figura A.10). Seleccionamos un directorio y pulsamos OK.

Aparece una pantalla de bienvenida que nos explica la estructura y el funcionamiento del programa. Para empezar a trabajar, pulsamos en la flecha de la derecha: **Workbench** (figura A.11).

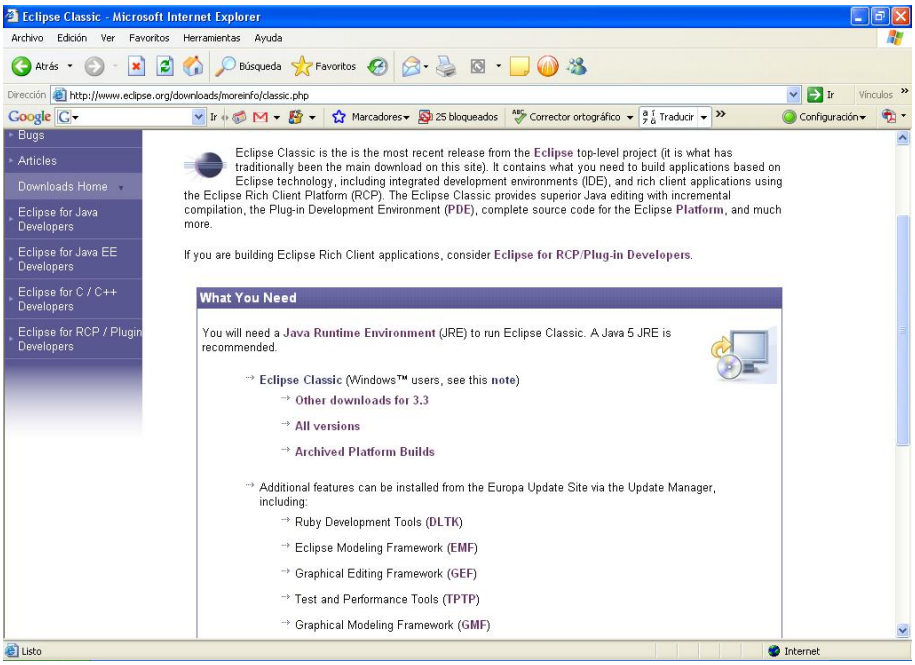

Figura A.7: Página de descarga de plug-ins.

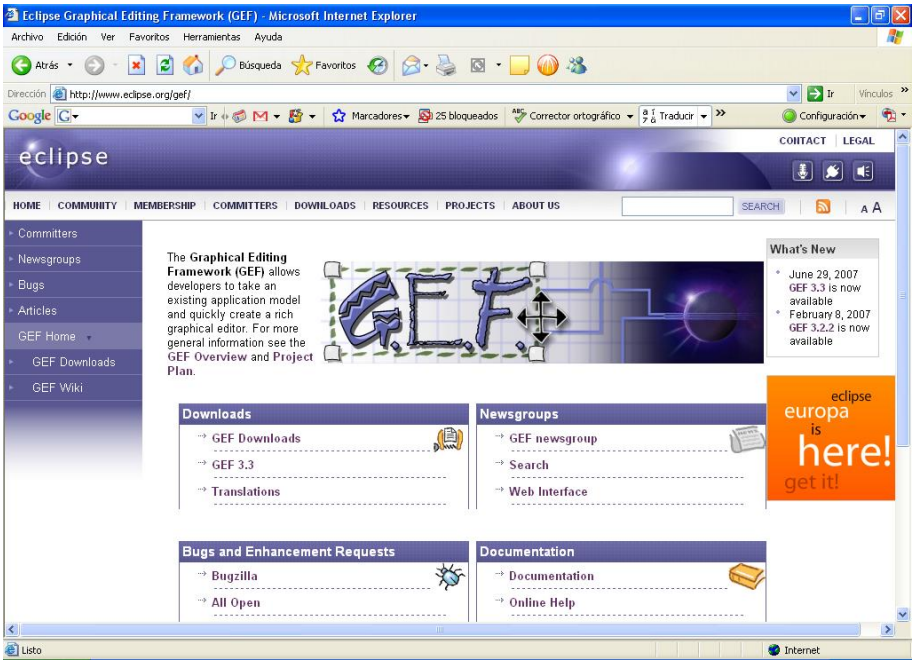

Figura A.8: Página de descarga de GEF.

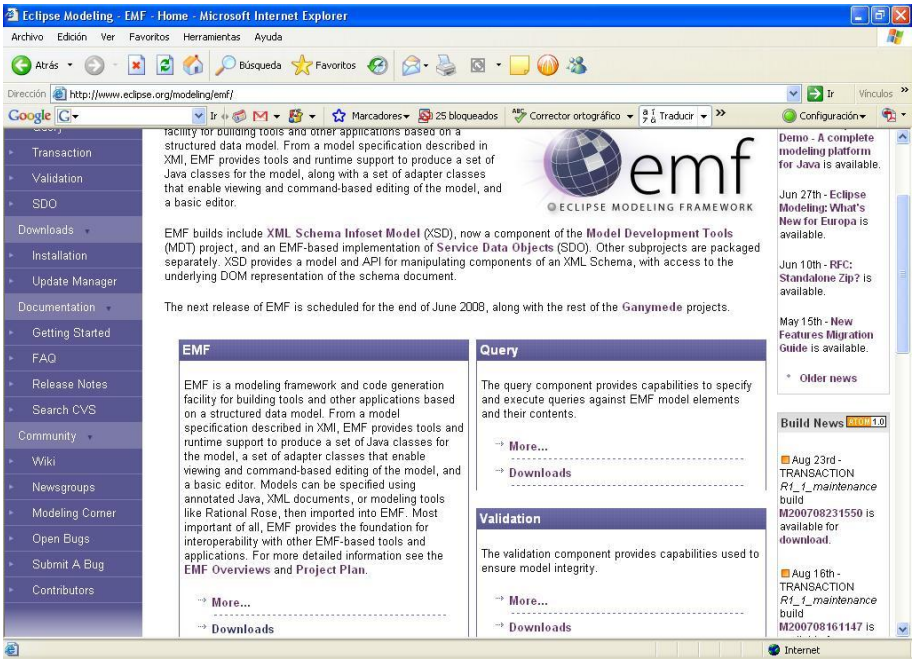

Figura A.9: Página de descarga de EMF.

Figura A.10: Inicio de Eclipse. Selección de destino para el workspace.

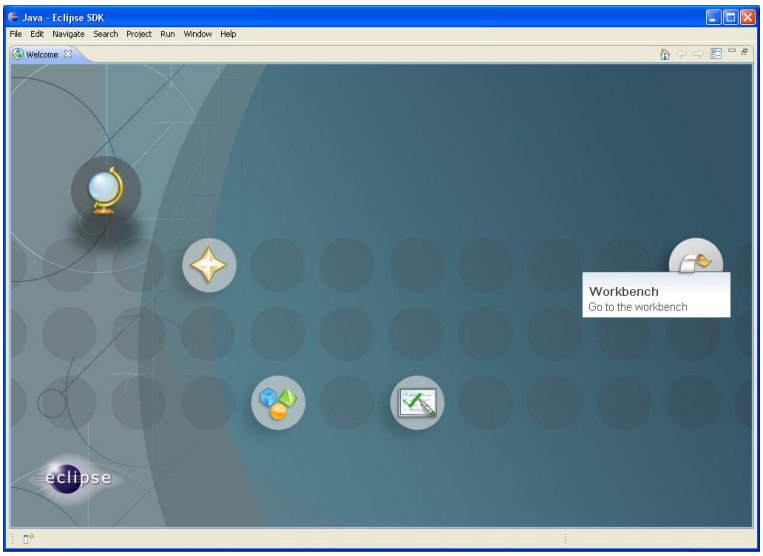

Figura A.11: Inicio de Eclipse. Pantalla de bienvenida

Para optimizar el funcionamiento del editor, es recomendable deshabilitar la opción 'Build automaticallyén el menú 'Project'.

A.2.3. Crear un proyecto nuevo o desde archivo

Desde el workbench (figura A.12), se selecciona File - New - Java Project - Next (figura A.13). Se escribe un nombre para el proyecto y se selecciona **Create new project in workspace** o **Create new project from existing source**. En este último caso, se busca la carpeta con el código fuente .java. Por último se pulsa **Finish** para cerrar el asistente.

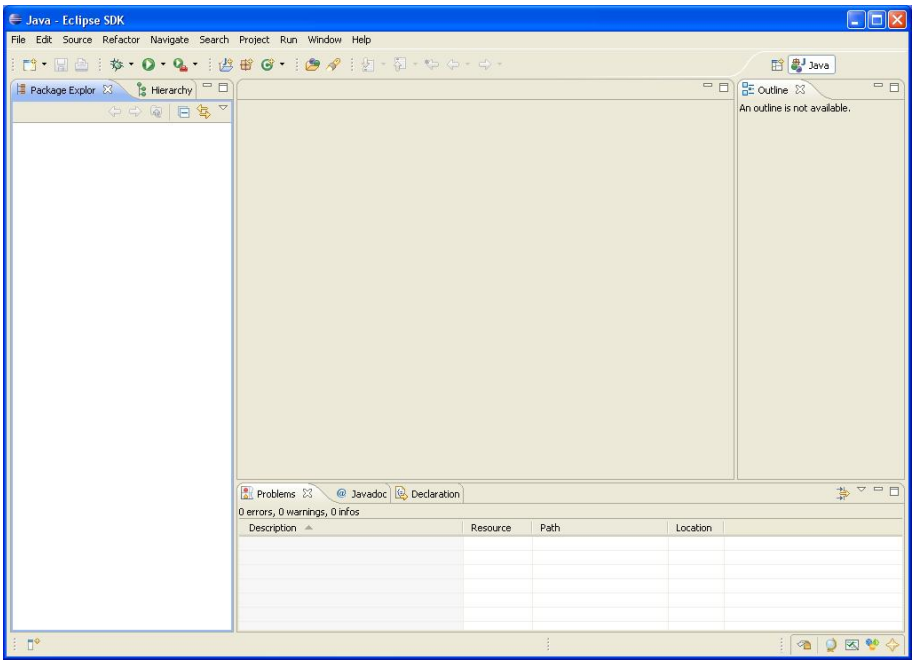

Figura A.12: Inicio de Eclipse. Aspecto del workbench.

A.3. Instalación de Java Advanced Imaging (JAI)

La versión actual de JAI se descarga de la página java.sun.com/products/java-media/jai/current.html (figura A.14). Se pulsa sobre la versión que se quiere descargar. En la página de descarga, se descargan los cuatro archivos correspondientes al sistema operativo (figura A.15). .

Una vez descargados, se ejecutan los archivos de instalación uno por uno y se siguen los pasos que dictan los asistentes de instalación.

A.4. Instalación de Java Media Framework (JMF)

JMF se descarga de la página java.sun.com/products/java-media/jmf/2.1.1/download.html (figura A.16).

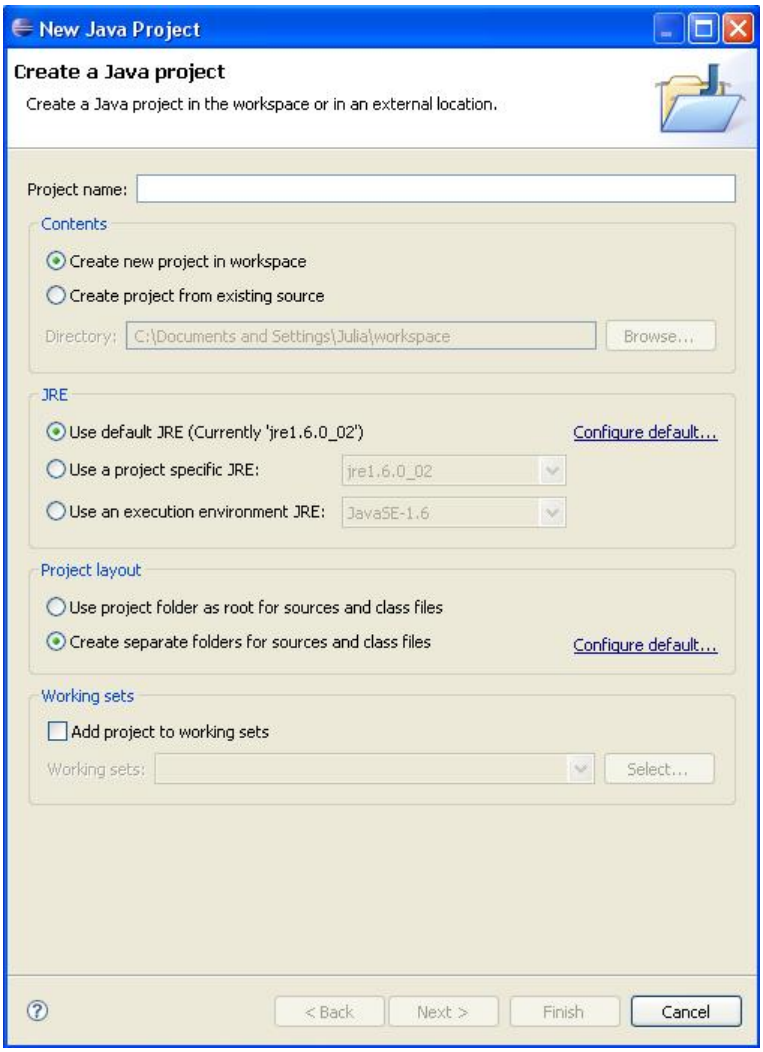

Figura A.13: Creación de un proyecto.

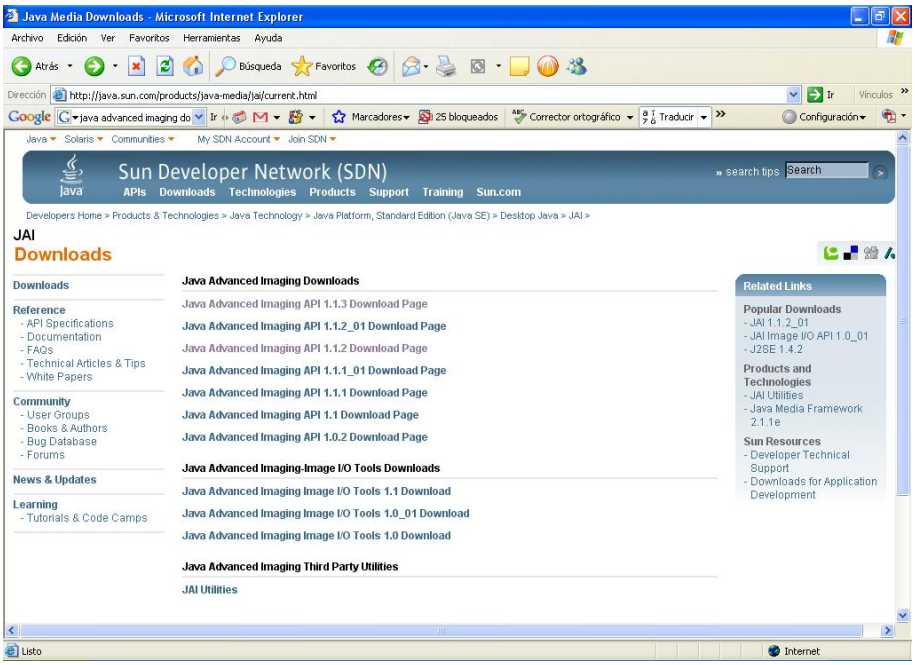

Figura A.14: Página de descarga de Java Advanced Imaging

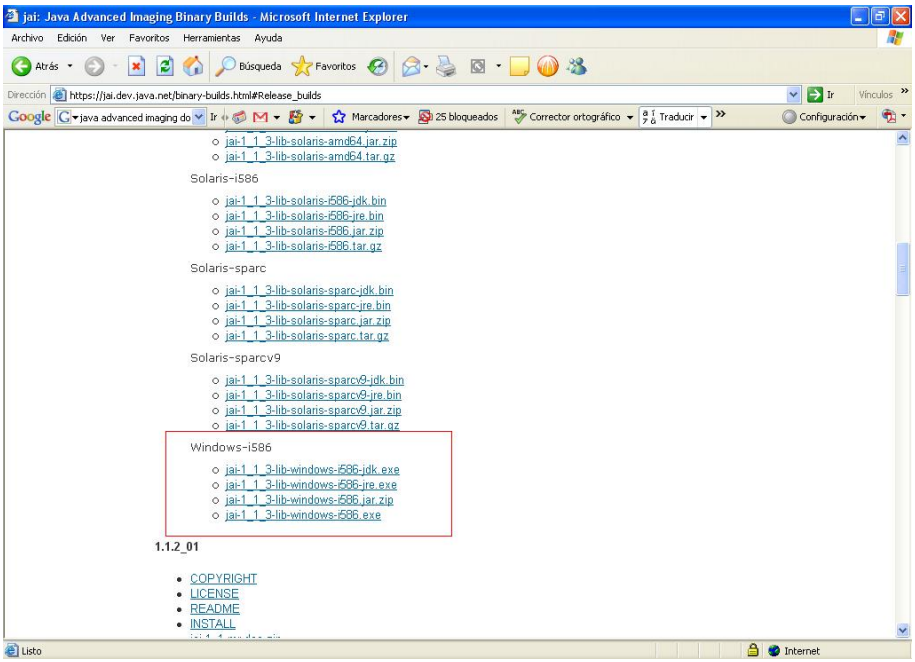

Figura A.15: Página de descarga de Java Advanced Imaging

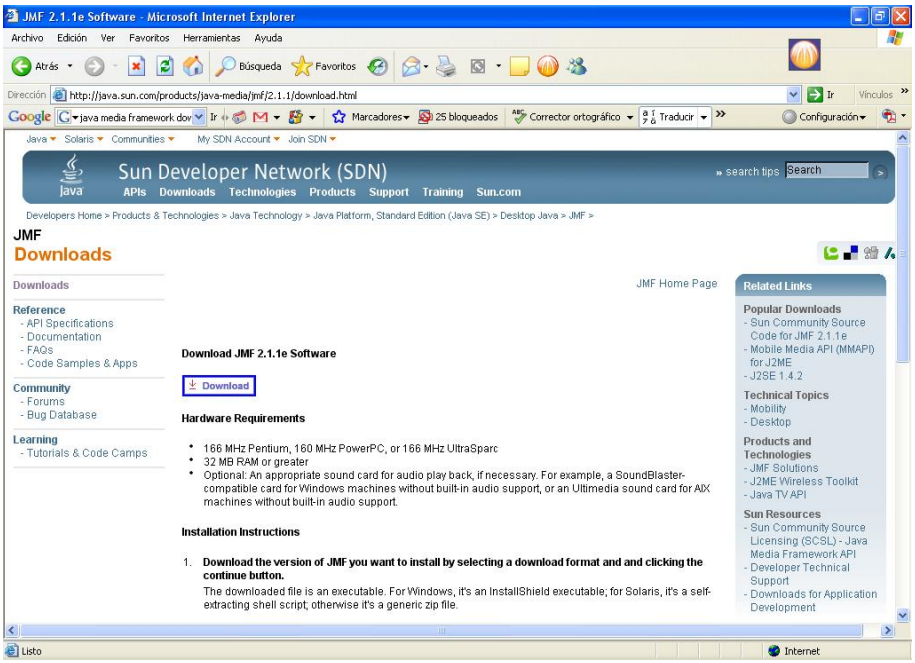

Figura A.16: Página de descarga de Java Media Framework

Se pulsa sobre la versión que se quiere descargar. En la página de descarga, se descarga el archivo correspondiente al sistema operativo (figura A.17). .

Después se ejecuta el archivo de instalación y se siguen los pasos que dicta el asistente de instalación.

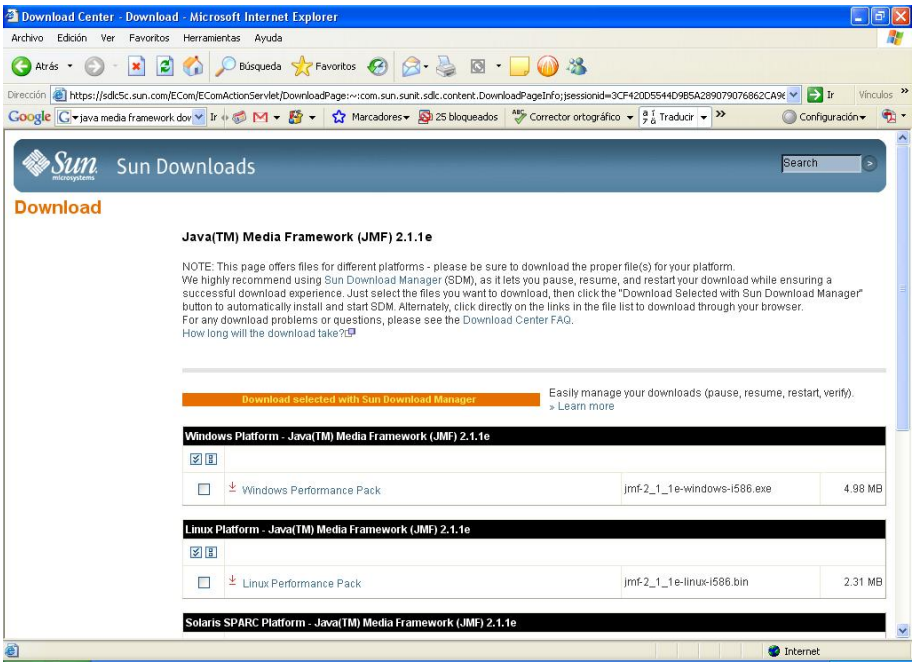

Figura A.17: Página de descarga de Java Media Framework

A.5. Instalación de JAI y JMF en Eclipse

Para instalar las librerías en un nuevo proyecto que no las traiga incorporadas, creamos una carpeta 'libén el directorio del proyecto. Examinamos el directorio donde hemos instalado JAI - por ejemplo Archivos de programa/Sun Microsystems -. Abrimos la carpeta lib y copiamos el contenido en nuestra carpeta. Repetimos el proceso con JMF. El resultado se muestra en la figura A.18. .

A continuación seleccionamos el menú Project - Properties - Java Build Path y la pestaña **Libraries** (figura A.19). Pulsamos el botón **Add JARs...** y buscamos la carpeta que hemos creado, 'lib'. Seleccionamos todos los archivos .jar (figura A.20). Pulsar OK y salir del asistente.

Ahora nuestro proyecto tiene instaladas todas las librerías que componen JAI y JMF, y mediante la sentencia import se pueden utilizar en las clases que se diseñen.

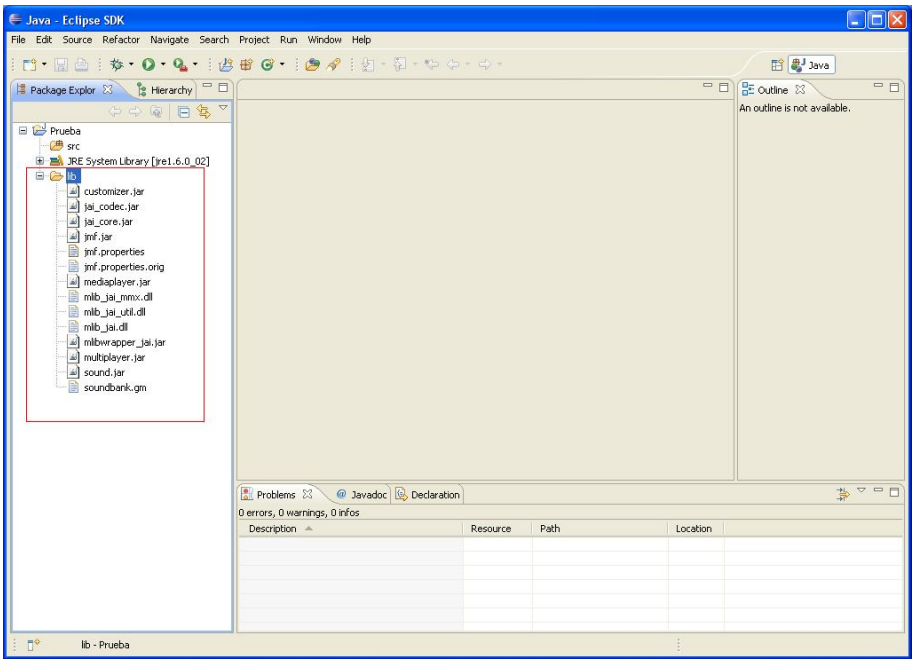

Figura A.18: Instalación de JAI y JMF en un proyecto de Eclipse.

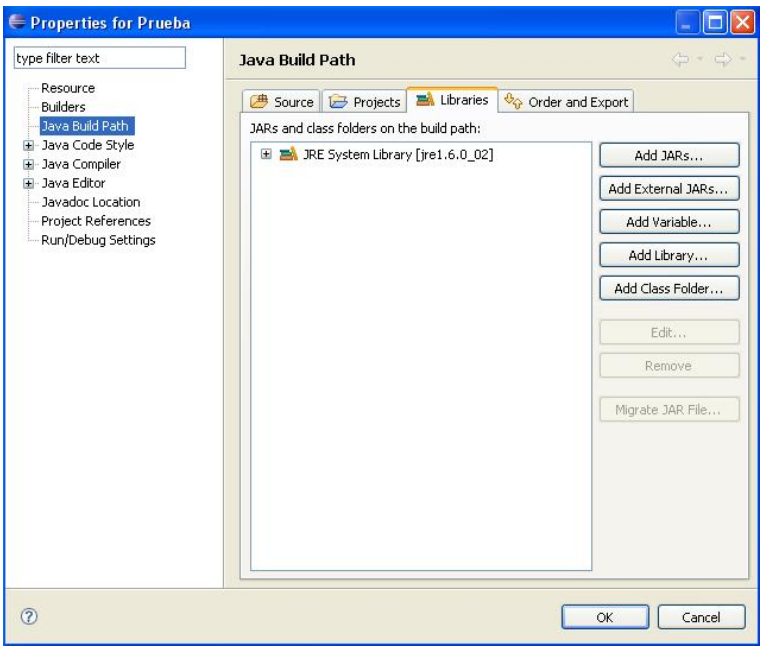

Figura A.19: Asistente para agregar librerías.

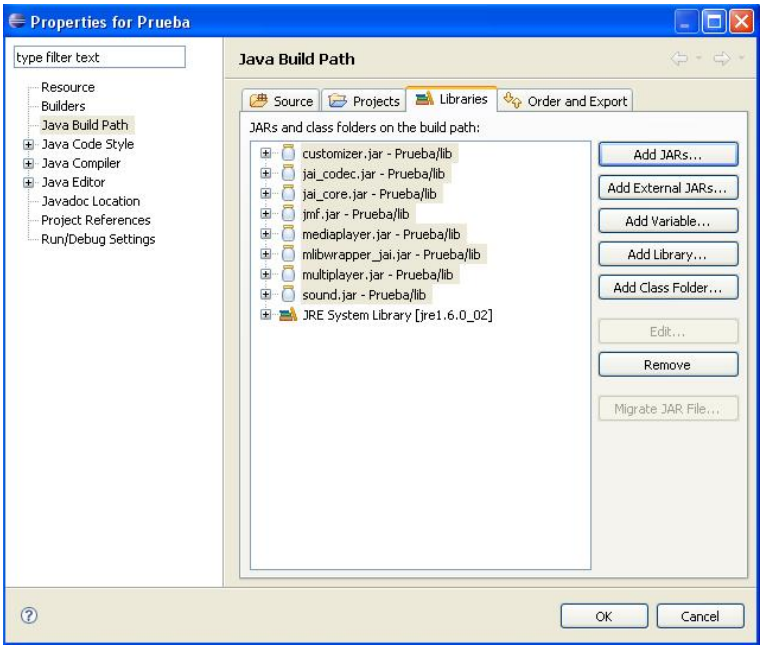

Figura A.20: Asistente para agregar librerías.

Apéndice B

Funciones de librería

En este apéndice se va a explicar con detalle las funciones de las librerías JAI y JMF que se han utilizado. También se incluyen funciones pertenecientes a AWT.

B.1. Java Advanced Imaging, JAI

- **JAI.create**
  - Declaración 1: RenderedOp javax.media.jai.JAI.create(String arg0, Object arg1)
  - Parámetros:
    - String arg0: nombre de la operación.
    - Object arg1: objeto parámetro.
  - Descripción: Esta función crea una imagen como objeto RenderedOp.
    Si arg0="stream", obtiene la imagen del archivo donde se encuentra.
    Si arg0="scale", escala la imagen según los valores introducidos en el objeto arg1.
    Si arg0="AWTImage", sirve para obtener una PlanarImage a partir de una Image.
  - Declaración 2: RenderedOp javax.media.jai.JAI.create(String arg0, ParameterBlock arg1)

- Parámetros:
  - String arg0: nombre de la operación.
  - ParameterBlock arg1: objeto que contiene las fuentes y/o parámetros de la operación.
- Descripción: Crea una imagen a partir de una operación cuyos parámetros están contenidos en un objeto ParameterBlock. En la forma JAI.create(.ˆextrema", pb), permite obtener los valores extremos de las bandas de una imagen.

- Declaración 3: RenderedOp javax.media.jai.JAI.create(String arg0, ParameterBlock arg1, RenderingHints arg2)
- Parámetros:
  - String arg0: nombre de la operación.
  - ParameterBlock arg1: objeto que contiene las fuentes y/o parámetros de la operación.
  - RenderingHints arg2: condiciones de presentación.
- Descripción: Crea una imagen a partir de una operación cuyos parámetros están contenidos en un objeto ParameterBlock, y bajo una condiciones de representación dadas por arg2.
  Con arg0=“subtract” resta una imagen a otra.
  Con arg0=“mean”, se obtienen los valores medios de las bandas de una imagen.
  Si arg0=”BandCombine”, se combinan las bandas de una imagen multibanda según lo especificado en arg1. Con arg0=”histogram”, se calculan los histogramas de las bandas de una imagen.

- **PlanarImage.getAsBufferedImage**
  - Declaración: BufferedImage javax.media.jai.PlanarImage.getAsBufferedImage()
  - Descripción: Devuelve una copia de la imagen como BufferedImage.

- **RenderedOp.getProperty**
  - Declaración: Object javax.media.jai.RenderedOp.getProperty(String arg0)
  - Parámetros:
    - String arg0: propiedad que se pretende obtener.

- Descripción: Devuelve la propiedad arg0 de un objeto RenderedOp. Si arg0="extrema", se obtienen los valores máximos y mínimos de las bandas de la imagen representada por el objeto RenderedOp. Si arg0="mean", se obtienen los valores medios de las bandas.

- **PlanarImage.getProperty**
  - Declaración: Object javax.media.jai.PlanarImage.getProperty(String arg0)
  - Parámetros:
    - String arg0: propiedad que se pretende obtener.
  - Descripción: Devuelve la propiedad arg0 de un objeto PlanarImage. Si arg0="histogram", se obtienen los histogramas de las bandas de la imagen.

- **ROI.getAsImage**
  - Declaración: PlanarImage javax.media.jai.ROI.getAsImage()
  - Descripción: Devuelve una imagen binaria de tipo PlanarImage a partir de un objeto ROI.

- **DisplayJAI.set**
  - Declaración: void com.sun.media.jai.widget.DisplayJAI.set(RenderedImage arg0, int arg1, int arg2)
  - Parámetros:
    - RenderedImage arg0: Imagen que se va a presentar en la clase DisplayJAI.
    - int arg1: coordenada x de la imagen en el panel DisplayJAI
    - int arg2: coordenada y.
  - Descripción: Para colocar una imagen arg0 en un panel DisplayJAI, con coordenadas arg1 y arg2.

B.2. Java Media Framework, JMF

- **MediaLocator**
  - Declaración: javax.media.MediaLocator.MediaLocator(String arg0)
  - Parámetros:
    - String arg0: URL del archivo multimedia.
  - Descripción: Crea un MediaLocator a partir de una representación en String de una URL.
- **createProcessor**
  - Declaración: Processor javax.media.Manager.createProcessor(MediaLocator arg0) throws java.io.IOException, NoProcessorException
  - Parámetros:
    - MediaLocator arg0: objeto origen, describe el contenido multimedia.
  - Descripción: crea un objeto Processor a partir de un MediaLocator.
- **getData**
  - Declaración: Object javax.media.Buffer.getData()
  - Descripción: Obtiene el objeto interno con los datos de la porción del medio contenida en un Buffer.

B.3. Abstract Windowing Toolkit, AWT

- **PixelGrabber**
  - Declaración: java.awt.image.PixelGrabber.PixelGrabber(Image img, int x, int y, int w, int h, int[] pix, int off, int scansize)
  - Parámetros:
    - Image img: imagen origen, de la que se van a extraer píxeles.
    - int x, int y: coordenadas de los píxeles iniciales de una sección rectangular de img.
    - int w, int h: dimensiones de la sección rectangular.

- int[] pix: vector donde se almacenan los píxeles de la sección.
- int off: offset en el vector donde almacenar el primer píxel.
- int scansize:distancia de una fila de píxeles a la siguiente en el vector.

- Descripción: 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.

▪ textbfMemoryImageSource

- Declaración: java.awt.image.MemoryImageSource.MemoryImageSource(int w, int h, ColorModel cm, int[] pix, int off, int scan)
- Parámetros:
  - int w, int h: dimensiones de la imagen que se va a crear.
  - ColorModel cm: modelo de color especificado para la imagen.
  - int[] pix: vector donde se almacenan los píxeles de la sección.
  - int off: offset en el vector donde almacenar el primer píxel.
  - int scansize:distancia de una fila de píxeles a la siguiente en el vector.
- Descripción: Construye un objeto Image a partir de un vector de enteros. Realiza la operación inversa a PixelGrabber.

Apéndice C

Ficheros de salida de texto

En este apéndice se va a explicar la estructura de los ficheros de texto que generan las aplicaciones desarrolladas en el proyecto fin de carrera. Estos ficheros recogen los resultados de los análisis llevados a cabo.

C.1. SparksSolver

Como ejemplo se muestra el fichero resultante del análisis que se llevó a cabo en el Capítulo 4 para explicar el funcionamiento del programa.

En primer lugar aparecen las rutas y los nombres de la imagen analizada y de la imagen de fondo, así como sus tamaños originales y tras el escalado para su presentación.

```
Image analyzed: C:\Documents and Settings\Julia\Mis documentos
\Ejecutable\SparksSolver\images\image_2316876890.jpg
Image name: image_2316876890.jpg
Background image: C:\Documents and Settings\Julia\Mis documentos
\Ejecutable\SparksSolver\images\background2.jpg
Background name: background2.jpg

Original image: Height = 324 ; Width = 480
Scaled image: Height = 553 ; Width = 819
```

A continuación tenemos la información extraída del algoritmo de la transformada de Hough. Se muestran los umbrales de binarización con histéresis de la imagen de bordes resultante de aplicar el operador de Sobel. Después aparecen los parámetros que definen las líneas obtenidas, en coordenadas polares. Por último, las coordenadas de la región del foco identificada, y su tamaño en píxeles.

```
Hough Transform:
Hysteresis threshold values: Lower: 50, upper: 100
Line: r = 437, theta = 87
Line: r = 425, theta = 89
Line: r = 412, theta = 91
Line: r = 243, theta = 108
Line: r = 326, theta = 102
Line: r = 253, theta = 106
Line: r = 228, theta = 112
Line: r = 273, theta = 107
Line: r = 400, theta = 92
Line: r = 388, theta = 95
Focus area begins in: x=380, y=400and has size:20x20
```

Después aparecen los datos obtenidos del cálculo del flujo óptico entre la imagen y el fondo:

```
Optical Flow:
Mean horizontal displacement: -0.95165014 px.
Mean vertical displacement: -0.9838289 px.
Minimum horizontal displacement: 0 px/sec at pixel (0,0).
Minimum vertical displacement: 0 px/sec at pixel (0,0).
Maximum horizontal displacement: -2 px at pixel (7,7).
Maximum vertical displacement: -2 px at pixel (7,7).
```

Y tras haber obtenido datos de los dos algoritmos anteriores, se unen los resultados para sacar conclusiones comunes. Se une mediante una línea el punto de máxima velocidad, obtenido del flujo óptico, con el foco de las chispas, identificado mediante la transformada de Hough. Se calculan las coordenadas polares de dicha línea, y se comparan con las de las líneas obtenidas en el algoritmo de Hough. Si el valor de theta calculado no se encuentra en el rango delimitado por los valores extremos de theta de las líneas - con tolerancia dada por el parámetro de configuración 'distance'-, el programa imprime

un aviso, ya que no se tiene la certeza de haber identificado correctamente el foco de las chispas. Esto ocurre en este ejemplo:

```
Data from Optical Flow and Hough Transform fusion:
Polar coordinates of the line between focus and maximum
 speed point:
r = 2, theta = 136.
Result: There is uncertainty in focus identification
```

El aviso no implica necesariamente que el foco se haya identificado erróneamente, es un mensaje conservador.

Lo siguiente son las estadísticas de las tres bandas (RGB) de la imagen sin el fondo: los valores máximos, mínimos y medios y los histogramas de cada banda (en ternas R - G - B).

```
RGB Image data:

Mean values:
Red mean = 38.436438385805474
Green mean = 18.57843884064499
Blue mean= 11.204949360464731

Extrema values:
Red max = 255.0
Red min = 0.0
Green max = 250.0
Green min = 0.0
Blue max = 255.0
Blue min = 0.0

RGB image Histogram:
211967 267818 271831
21654 16596 23425
11133 16759 10930
7883 8430 12278
7620 6367 8570
4866 5708 8515
4911 4696 5915
4890 4039 6704
```

...

```
930 379 66
932 369 48
1064 256 69
985 153 44
913 157 23
903 146 14
867 114 31
828 66 36
763 8 24
786 48 18
669 0 14
648 0 12
1005 0 20
1228 0 15
2611 0 11
```

A continuación aparecen las mismas estadísticas para la luminancia:

```
Grayscale Image data:

Luminance mean
= 23.703022916404468

Extrema values:
Luminance max = 251.0
Luminance min = 0.0

Grayscale image Histogram:
190258
46542
24147
13148
10093
8406
5997
```

...

```
289
137
80
27
23
10
16
0
2
0
0
0
0
```

Por último aparecen los datos calculados a la imagen umbralizada en intensidad: las estadísticas y los histogramas RGB y de luminancia, y el área de la zona de chispas, en porcentaje de la imagen y en número total de píxeles.

```
Threshold-established image
data:

Threshold value = 150

RGB Image data:

Mean values:
Red mean = 242.70025974025975
Green mean = 173.35229437229438
Blue mean= 118.66419913419914

Extrema values:
Red max = 255.0
Red min = 177.0
Green max = 250.0
Green min = 111.0
Blue max = 255.0
Blue min = 19.0

RGB image Histogram:
0 0 0
0 0 0
0 0 0
0 0 0
0 0 0

...

818 66 36
746 8 24
```

```
771 48 18
654 0 14
642 0 12
997 0 20
1220 0 15
2581 0 11

Grayscale Image data:

Luminance mean
 = 188.0912987012987

Extrema values:
Luminance max = 251.0
Luminance min = 150.0

Grayscale image Histogram:
0
0
0
0
0
```

```
0

     ...

240
246
289
137
80
27
23
10
16
0
2
0
0
0
0

ROI size: 23100 px
5.1003847% total.
```

C.2. SparksAnalyzer

En el Capítulo 4, cuando se explicaba el uso de esta aplicación, se analizó el vídeo **sparks.mpeg**, que se proporciona con SparksAnalyzer. En este apartado se va a explicar el contenido del fichero de salida con los datos del análisis llevado a cabo en el Capítulo 4.

En primer lugar se muestra la información del vídeo analizado: ruta de acceso, nombre y formato.

```
Analyzed Video: C:\Documents and Settings\Julia\Mis documentos
\Ejecutable\SparksAnalyzer\videos\sparks.mpeg
Video name: sparks.mpeg

Push Format RGB, 480x324, FrameRate=23.9, Length=466561, 24-bit,
 Masks=3:2:1, PixelStride=3, LineStride=1440
Frame size: 480x324
```

A partir de ahí empieza la información relativa a las frames que se van analizando. La primera frame se toma como fondo para el análisis, y se muestran sus características:

```
Background Image:
 Time stamp: 481708375
 Time: 0.48secs
 Sequence #: 1
 Data length: 466560
 Key Frame: false 32
 *** Created file image_481708375.jpg
```

El resto del fichero recoge la información de las frames que se analizan restándoles el fondo anterior. Para cada frame se imprime información similar a la que se proporciona en SparksSolver, descrita en la sección anterior. En concreto:

- Características.
- Estadísticas de las bandas RGB: valores extremos y medios.
- Histogramas de las bandas RGB.

- Estadísticas de luminancia.
- Histograma de luminancia.
- Estadísticas e histogramas RGB y de luminancia para la imagen umbralizada en intensidad. Tamaño de área de chispas.
- Existencia o no de chispas según el umbral seleccionado.
- En caso de que existan chispas:
  - Información obtenida de la transformada de Hough: umbrales de binarización con histéresis, parámetros de las trayectorias obtenidas y región del foco identificada.
  - Información de velocidad obtenida del flujo óptico entre la frame actual y la anterior.
  - Integración de los datos obtenidos en los dos algoritmos anteriores.
- Nombre del archivo en el que se salva la imagen, si se habilitó esa opción.

Cabe destacar que hasta que no se tienen dos frames útiles, esto es, sin contar la que se toma como fondo, no se calcula el flujo óptico.
En el ejemplo que nos ocupa:

```
Analyzed Image:
 Time stamp: 3151044396
 Time: 3.15secs
 Sequence #: 65
 Data length: 466560
 Key Frame: false 32
Original image:
Height = 324 ; Width = 480
Scaled image:
Height = 553 ; Width = 819

RGB Image data:

Mean values:
Red mean = 35.437264162399785
Green mean = 17.805787943220132
Blue mean= 10.40468352222421

Extrema values:
Red max = 255.0
Red min = 0.0
Green max = 254.0
Green min = 0.0
Blue max = 255.0
Blue min = 0.0

RGB image Histogram:
269807 272265 325398
9173 27368 7797
7271 18355 7872
5714 12925 6833
5412 7802 6851
```

```
4845 5651 5999
     ...
897 48 14
758 34 6
773 22 12
696 8 2
631 4 2
4479 0 88

Grayscale Image data:

Luminance mean
 = 13.869370533023336

Extrema values:
Luminance max = 251.0
Luminance min = 0.0

Grayscale image Histogram:
202927
61704
30201
11491
8877
7552
5086
     ...
12
10
0
0
0
0

Threshold-established
 image data:
Threshold value = 150

RGB Image data:

Mean values:
Red mean = 243.0228688557534
Green mean = 179.90964993368434
Blue mean= 128.271050198947

Extrema values:
Red max = 255.0
Red min = 144.0
Green max = 254.0
Green min = 112.0
Blue max = 255.0
Blue min = 25.0

RGB image Histogram:
0 0 0
0 0 0
0 0 0
0 0 0
0 0 0
0 0 0
0 0 0
     ...
814 48 14
715 34 6
719 22 12
655 8 2
605 4 2
4323 0 88

Grayscale Image data:

Luminance mean
 = 193.1657087737631

Extrema values:
Luminance max = 251.0
```

```
Luminance min = 150.0                    ...

Grayscale image Histogram:       12
0                                10
0                                0
0                                0
0                                0
0                                0
0
0                                ROI size: 24881 px
                                 5.4936223% total.
```

Si en la imagen hay chispas tenemos:

```
Hough Transform:
Hysteresis threshold values: Lower: 50, upper: 100
Line: r = 325, theta = 94
Line: r = 237, theta = 114
Line: r = 58, theta = 130
Line: r = 211, theta = 115
Line: r = 42, theta = 132
Line: r = 69, theta = 126
Line: r = 227, theta = 116
Line: r = 412, theta = 91
Line: r = 187, theta = 121
Line: r = 201, theta = 117
Focus area begins in: x=400, y=420and has size:20x20

Optical Flow:
Mean horizontal speed: -2.1259468 px/sec.
Mean vertical speed: -2.0515125 px/sec.
Minimum horizontal speed: 0.0 px/sec at pixel (0,0).
Minimum vertical speed: 0.0 px/sec at pixel (0,0).
Maximum horizontal speed: -4.878049 px/sec at pixel (14,14).
Maximum vertical speed: -4.878049 px/sec at pixel (14,14).

Data from Optical Flow and Hough Transform fusion:
Polar coordinates of the line between focus and maximum speed point:
r = 1, theta = 136.
 *** Created file image_3151044396.jpg
```

Si no hay certeza de que se haya realizado correctamente la identificación del foco de las chispas:

```
Data from Optical Flow and Hough Transform fusion:
Polar coordinates of the line between focus and maximum speed point:
r = 1, theta = 134.
Result: There is uncertainty in focus identification
 *** Created file image_4610837533.jpg
```

Y si en la imagen no hay chispas:

```
No sparks in this frame.
 *** Created file image_857083753.jpg
```

Por último, al final del fichero se resumen las frames recibidas y las procesadas:

```
End of analysis
Sources closed
Received frames: 110
Processed frames: 11
```

Apéndice D

Contenido del CD adjunto

En el CD que se proporciona junto a esta memoria se encuentran los siguientes elementos:

- Memoria del proyecto.
- Aplicaciones desarrolladas:
  - SparksSolver
  - SparksAnalyzer
- Código fuente de las aplicaciones desarrolladas

En las aplicaciones desarrolladas se incluyen imágenes y vídeos como ejemplo para su análisis. Las instrucciones para su uso se explican con detalle en el Capítulo 4 de esta memoria.

Bibliografía

[1] T. Thomessen, T. K. Lien, P. K. Sannæs. *Robot control system for grinding of large hydro power turbines.* Industrial Robot: An International Journal. Volume 28. Number 4. 2001. pp. 328-334.

[2] Fundación de investigación de la máquina-herramienta. `http://www.invema.es`.

[3] Wikipedia. `http://www.wikipedia.org`.

[4] J.L.Barron and N.A.Thacker. *Tutorial: Computing 2D and 3D Optical Flow.* Imaging Science and Biomedical Engineering Division, Medical School, University of Manchester, 2005.

[5] S. Narasimhan. Computer vision, lecture 16, Spring 2006.

[6] José Ramón Cerquides Bueno. Televisión, Tema 4: Televisión Digital, Apuntes de Cátedra. Universidad de Sevilla, 2007.

[7] E. Ospina Piedrahíta, J. P. Urrea Duque. *Implementación de la transformada de Hough para la detección de líneas para un sistema de visión de bajo nivel.* Universidad Nacional de Colombia. Facultad de Ingeniería y Arquitectura. Departamento de Ing. Eléctrica, Electrónica y Computación, 2002.

[8] Rafael C. González, Richard E. Woods. *Digital Image Processing.* Addison-Wesley, 1992.

[9] Berthold Klaus Paul Horn. *Robot Vision.* Cambridge, Mass. MIT Press, 1998 New York [etc.] McGraw-Hill, 1998

[10] *Programming in Java Advanced Imaging.* Sun Microsystems, Release 1.0.1. November 1999.

[11] *Java Media Framework API Guide.* Sun Microsystems, November 19,1999.

[12] Eclipse Project.
`http://www.ecilpse.org`.

[13] *Eclipse Platform Technical Overview.* Object Technology International, Inc. February 2003 (updated for 2.1; originally published July 2001).
`http://www.eclipse.org/whitepapers/eclipse-overview.pdf`.

[14] Iván García Puebla. *Eclipse: una herramienta profesional al alcance de todos.* Grupo Universitario de Informática de la UVA.
`http://www.gui.uva.es/~laertes/nuke/index.php?option=com_content&task=view&id=56&Itemid=41`.

[15] Computer Vision Demonstration Website. Electronics and Computer Science, University of Southampton
`http://users.ecs.soton.ac.uk/msn/book/new_demo/`.

[16] Java Technology
`http://java.sun.com/`.

[17] Java Advanced Imaging API Documents
`http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/`

[18] Java Media Framework API Documents
`http://java.sun.com/products/java-media/jmf/2.1.1/apidocs/`

[19] Java 2 Platform, Standard Edition, v 1.4.2 API Specification
`http://java.sun.com/j2se/1.4.2/docs/api/`

[20] T. Oetiker, H. Partl, I. Hyna and E. Schlegl. *The Not So Short Introduction to LaTeX 2ε.*, Version 4.22, June 30, 2007.

[21] Mini manual de instalación de LaTeX.
`http://copa.uniandes.edu.co/software/latex/manual.html`