Universitat Autònoma de Barcelona

Escola Tècnica Superior d'Enginyeria

AYUDA PARA LA CONFIGURACIÓN DE CÁMARAS

Memoria del proyecto de Ingeniería Informática realizado por Daniel Haro Ruiz

Y dirigido por Xavier Sánchez Pujadas

Bellaterra, 2 de Febrero de 2010

Universitat Autònoma de Barcelona

Escola Tècnica Superior d'Enginyeria

El abajo firmante Xavier Sánchez Pujadas

Professor/a de l'Escola Tècnica Superior d'Enginyeria de la UAB,

CERTIFICA:

Que el trabajo al que corresponde esta memoria ha sido realizado bajo su dirección por Daniel Haro Ruiz

Y por tal que conste firma la presente.

Firmado: Xavier Sánchez Pujadas

Bellaterra, 2 de Febrero de 2009

A mis padres y a mi hermano por su apoyo y comprensión.

A Laia por estar siempre a mi lado.

TABLA DE CONTENIDO

1 Introducción 1

1.1 Contenido de la memoria 1

1.2 Motivaciones 2

1.3 Objetivos 3

1.4 Estado del arte 4

2 Planificación 7

2.1 Planificación inicial 7

2.2 Planificación final 9

2.3 Estudio de viabilidad 13

3 Conceptos previos 15

3.1 MFC (Microsoft Foundation Classes) 15

3.1.1 Arquitectura Documento/Vista 16

3.1.2 Clase CFormView y CDialog 17

3.1.3 Otros conceptos de implementación 19

3.2 IIDC 1394-based Digital Camera Specification 20

3.3 VisionOkII 21

3.3.1 C.M.U. Drivers 21

3.3.2 Implementación de VisionOkII 23

3.3.3 Clase CVOkImageDoc 24

3.3.4 Clase CVOkImageView 25

3.3.5 Creación y configuración de cámaras 27

3.4 Conceptos teóricos a desarrollar 29

3.4.1 Histograma 29

3.4.2 Exposición 30

3.4.3 Balance de blancos 31

3.4.4 Enfoque 33

4 Desarrollo 39

4.1 Área de ajuste 39

4.2 Exposición 40

4.3 Balance de blancos 42

4.4 Histograma 45

4.5 Enfoque 49

4.6 Configuración de parámetros de la cámara 51

5 Pruebas 57

5.1 Test funcional 57

5.2 Test de rendimiento 59

6 Conclusiones 61

6.1 Conclusiones 61

6.2 Líneas futuras 62

7 Anexo 1: Manual de usuario 63

8 Bibliografía 79

9 Índice 81

TABLA DE ILUSTRACIONES

Figura 2. 1 Planificación Inicial y Final 12

Figura 3. 1 Arquitectura documento Vista 16
Figura 3. 2 Estructura *WINDOWPLACEMENT* 19
Figura 3. 3 Ejemplo dibujar píxel 20
Figura 3. 4 Instancia *CMultiDocTemplate* 24
Figura 3. 5 *ON_COMMAND* y *ON_UPDATE_COMMAND_UI* 26
Figura 3. 6 Obtención del valor de un parámetro 28
Figura 3. 7 Histograma 29
Figura 3. 8 Exposición 30
Figura 3. 9 Balance de blancos 31
Figura 3. 10 Temperatura de color en la escala Kelvin 32
Figura 3. 11 Enfoque 33
Figura 3. 12 Operador *Sobel* y *Prewitt* 35
Figura 3. 13 Aplicación del operador *Sobel* 35

Figura 4. 1 Subexposición y Sobreexposición 41
Figura 4. 2 Formulario Balance de blancos 43
Figura 4. 3 Formulario Histograma 45
Figura 4. 4 Estructura *color* y *nColor* 46
Figura 4. 5 Enfoque 49
Figura 4. 6 Mapa de eventos 52
Figura 4. 7 Diálogo nueva cámara configurable 53
Figura 4. 8 Diálogo de la configuración de cámaras 55
Figura 4. 9 Opción de Menu 55

Figura 7. 1 Cargar Proyecto 63
Figura 7. 2 Elección de la cámara 64
Figura 7. 3 Proyecto en línea 64
Figura 7. 4 Selección Nueva cámara auto configurable 65
Figura 7. 5 Menú nueva cámara auto configurable 66
Figura 7. 6 Configuración automática 66
Figura 7. 7 Diálogo de configuración de cámara 67

Figura 7. 8 Nueva vista 68
Figura 7. 9 Elección de la vista 68
Figura 7. 10 Ajustes de la imagen 69
Figura 7. 11 Imagen sobreexpuesta 70
Figura 7. 12 Histograma de la imagen 71
Figura 7. 13 Balance de blancos 71
Figura 7. 14 Exposición 72
Figura 7. 15 Movimiento rectángulo de ajuste 72
Figura 7. 16 Imagen normal 73
Figura 7. 17 Inquiry Registrer for video format 73
Figura 7. 18 Inquiry Registrer for video mode 74
Figura 7. 19 Inquiry Registrer for video frame rate 78

Capítulo 1

INTRODUCCIÓN

El mundo empresarial es un mundo cada vez más competitivo, donde las empresas hacen de la tecnología una herramienta para prosperar en el mercado. Por esta razón, las empresas cada vez necesitan tener más procesos industriales automatizados que proporcionen aumento de la productividad y reducción de costes.

El campo de la visión por computador y el procesamiento de imágenes no han quedado al margen de esta evolución. Actualmente, estos campos de la inteligencia artificial participan en proyectos relacionados con la automatización de procesos que, hasta hace poco, tenía que hacer el ser humano. Ejemplos podrían ser la fabricación de sistemas de análisis de vídeo, de fotografía, etc.

Este proyecto está destinado a proporcionar herramientas que ayude en la configuración de cámaras para uso industrial pero también se pueden exportar a otros campos relacionados con el vídeo y la fotografía digital.

1.1 CONTENIDO DE LA MEMORIA

Esta memoria está dividida en diferentes capítulos. A continuación se detalla en qué consiste cada uno de ellos:

- **Capítulo 2: Planificación**
  En este apartado se detalla tanto la planificación inicial del proyecto como la final. También se comenta el estudio de viabilidad realizado.

- **Capítulo 3: Conocimientos previos**
  Este punto contiene el estudio realizado para poder abarcar los objetivos propuestos en este proyecto. Se realiza un proceso de aprendizaje del software proporcionado, la teoría relacionada con los módulos a desarrollar y la librería que se va a utilizar.

- **Capítulo 4: Desarrollo**

En esta sección se comenta detalladamente el diseño y la elaboración del software implementado.

- **Capítulo 5: Pruebas**
  Este punto hace referencia a todo el juego de pruebas que se ha realizado durante todo el proyecto, tanto para verificar el correcto funcionamiento como el rendimiento de los módulos desarrollados.

- **Capítulo 6: Conclusiones**
  En este punto se explica cuales han sido las conclusiones obtenidas durante la realización del proyecto. También se hace referencia a posibles futuras ampliaciones de este proyecto.

- **Capítulo 7: Anexo 1**
  Este capítulo consiste en un manual de usuario para facilitar la familiarización con el software desarrollado. También se encuentran las tablas de configuración para cámaras *firewire*.

- **Bibliografía**
  Por último, esta sección hace referencia a toda la bibliografía recogida durante la elaboración del proyecto.

- **Índice**

1.2 MOTIVACIONES

Este proyecto forma parte de un software que se está diseñando para un cliente del *Centre de Visió per Computador* de la *Universitat Autònoma de Barcelona*. Esto quiere decir, que por un lado es interesante realizarlo ya que permite conocer, aunque de manera indirecta, como funciona este tipo de proyectos. La otra cara de la moneda, es que no consiente libertad absoluta para desarrollar e implementar el proyecto ya que existen una serie de requerimientos y restricciones que se deben cumplir en la medida de lo posible.

También influye el hecho de trabajar sobre un software en edad avanzada de desarrollo, lo que permite adquirir ciertas habilidades a la hora de interpretar ciertos conceptos o métodos de implementación usados con anterioridad.

Para finalizar este punto, se debe tener muy en cuenta que las herramientas a desarrollar deben ser de fácil comprensión y usabilidad por parte del técnico encargado de manejarlo, ya que este individuo no tiene porque poseer grandes conocimientos en procesamiento de imágenes y en fotografía digital.

1.3 OBJETIVOS

El objetivo que presenta la realización de este proyecto es añadir al software *VisionOKII* herramientas para la configuración de una cámara. Estas herramientas permiten calcular el mejor enfoque de una imagen, visualizar el histograma de dicha imagen y también el grado de exposición de esta. La puesta a punto de la cámara debe realizarse manualmente por el usuario. De esta manera, se asegura que la configuración perdurará en el tiempo. Hay que pensar que la cámara debe captar imágenes lo más parecidas posibles para una buena detección de patrones.

Estas funcionalidades añadidas deben facilitar la puesta a punto de una cámara ya que estas herramientas proporcionan mucha información sobre la calidad del elemento a visualizar. Para cumplir este objetivo, también es necesario presentar la información de una manera clara y visualmente correcta. Es por esto, que se ha presentado la información mediante gráficos y formularios.

Desde el punto de vista funcional, no sólo interesa configurar la cámara sino que lo realmente interesante es poder configurarla para obtener aquella parte de la imagen que es útil. Es por esto, que se desarrolla una clase específica para permitir al usuario dicha funcionalidad.

Además se desarrolla una herramienta que permite introducir los parámetros de configuración de la cámara, como por ejemplo la resolución, el formato de los píxeles o la velocidad de captura de imágenes. Esto es útil porque facilita dicha configuración y no es necesario tener un conocimiento específico sobre los parámetros que se deben introducir para poder hacer uso de la cámara.

Tal y como se ha comentado en el punto anterior, se debe ofrecer una interface amigable y de rápida comprensión y de esta forma facilitar si uso.

A continuación se muestran los pasos que debe seguir un usuario para configurar correctamente una cámara:

i. Una vez abierto *VisionOkII* crear un nuevo proyecto.
ii. Crear un nuevo dispositivo *Firewire*.
iii. Activar Proyecto en línea.
iv. Añadir una *Nueva Cámara Autoconfig* desde el menú desplegable.
v. Seleccionar la cámara a configurar o, si se desea crear una nueva, elegir un nombre.
vi. Seleccionar los parámetros de dicha cámara:
    a. Tipo: Blanco/negro o Color
    b. Resolución
    c. Velocidad
vii. Una vez configurada la cámara crear una nueva vista para dicha cámara.
viii. Abrir la vista y seleccionar el modo *Captura continua*.
ix. Ajustar exposición en el área deseada mediante el módulo de exposición a desarrollar.
x. Ajustar el balance de blancos utilizando las herramientas del histograma y balance de blancos.
xi. Ajustar el enfoque mediante el módulo de autoenfoque.

Al finalizar este proyecto, un usuario debe ser capaz de ajustar los parámetros de una cámara con el fin de obtener una imagen con la mayor calidad posible.

1.4 ESTADO DEL ARTE

Hoy en día se puede encontrar diferentes softwares que proporcionan herramientas parecidas a las implementadas en este proyecto.

Actualmente, la inmensa mayoría de cámaras fotográficas y de vídeo llevan incorporadas software que ajusta automáticamente el enfoque de la imagen que se está visualizando en el momento. También pueden llevar incorporado software que

visualiza el histograma de la imagen capturada, etc. Por otro lado, existe software destinado a la edición de imágenes digitales como puede ser *Adobe Photoshop* que incorpora herramientas que desvelan si la imagen tiene un correcto balance de blancos. No solo se puede ver mediante el histograma representativo de la imagen sino que se logra hacer visualmente en la propia imagen visualizando la posible sobreexposición o subexposición a la que ha sido sometida dicha captura.

Capítulo 2

PLANIFICACIÓN

En el capítulo 2, se describe la planificación inicial y final del proyecto. Se han representado mediante unos diagramas, como se puede observar en la **Figura 2.1**. Se observa cómo es alterada respecto la propuesta inicialmente, debido a la falta de experiencia en la planificación de proyectos y, sobretodo, a la aparición de imprevistos durante el desarrollo del mismo. En este capítulo también se encuentra el estudio de viabilidad correspondiente al proyecto realizado.

2.1 PLANIFICACIÓN INICIAL

Para expresar el tiempo utilizado en el desarrollo del proyecto, se usa la semana como forma representativa de las horas trabajadas. Se tiene en cuenta que cada día se van a trabajar entre 2:30 – 3 horas. De esta manera se trabajan alrededor de 400 horas durante el proyecto.

A continuación se exponen los puntos que se tuvieron en cuenta en un primer momento y cuyo resultado define la planificación inicial:

- **Estudio teórico**
  Inicialmente, se ha de realizar un estudio teórico acerca de los conceptos relacionados con la fotografía: exposición, enfoque, balance de blancos y otros conceptos que no se detallan en esta memoria, pero que son imprescindibles en el mundo de la fotografía. También se efectúa el estudio de diferentes técnicas de adquisición de datos para determinar los algoritmos que mejor se adecuan a este software. Se ha estimado una duración de diez días para esta etapa.

- **Estudio VisionOkII**
  Este apartado corresponde a la tarea aprendizaje y familiarización con el software *VisionOkII*, realización de pequeños experimentos con diferentes imágenes y comprensión del código de las partes más importantes del software. La duración de esta sección corresponde a quince días.

- **Estudio CMU Drivers**
  Este punto requiere de una tarjeta *PCI* con puertos *firewire*, una cámara en blanco y negro, una lente y el cable de conexión. Los últimos tres elementos han sido proporcionados por el director de este proyecto. La tarea consiste en la instalación y configuración de los drivers mencionados y la realización de pruebas con la demo que proporciona el proveedor. De nuevo, el tiempo estimado para realizar este apartado es de diez días.

- **Desarrollo Exposición**
  Este apartado requiere que los dos primeros hayan sido finalizados, en la medida de lo posible, para un correcto seguimiento de la planificación. Como bien describe el nombre de la tarea, esta se basa en el desarrollo del módulo de exposición. El periodo de tiempo destinado a este punto es de quince días.

- **Desarrollo del histograma**
  Una vez finalizado el módulo de exposición, se prosigue con el desarrollo del módulo del histograma. Para la realización con éxito de este apartado se hace necesario el estudio de conceptos sobre programación referentes al trato de formularios. Este apartado debe cumplimentarse en diez días.

- **Desarrollo del balance**
  El desarrollo del balance de blancos debe realizarse en quince días. Para ello es necesario, adquirir conocimientos acerca del balance blancos.

- **Desarrollo del enfoque**
  El tiempo necesitado para la elaboración del módulo de enfoque es el punto más sensible en cuanto al estudio teórico a realizar ya que existen diferentes tipos de algoritmos a usar para obtener el mejor enfoque de una escena. Se establecen veinticinco días para la finalización del módulo.

- **Periodo de exámenes**
  Este tiempo está destinado al estudio intensivo previo a los exámenes y realización de estos. En total se necesita de treinta días de pausa en la elaboración del proyecto.
- **Realización de pruebas**

La realización de pruebas va acorde con el tiempo destinado al desarrollo de los diferentes módulos que contempla este proyecto. Además, se añade quince días adicionales para el testeo de todos los módulos ensamblados a modo de pruebas finales.

- **Vacaciones**
  El espacio destinado a las vacaciones estivales es de quince días.

- **Elaboración de la memoria**
  Este punto se realiza en paralelo con los anteriores. De esta manera se lleva la memoria al día y no se pierden datos sobre la realización del proyecto que pueden ser relevantes.

- **Presentación**
  Este apartado hace referencia al tiempo necesario para la elaboración de la presentación. Se necesitan alrededor de diez días para realizar esta tarea.

2.2 PLANIFICACIÓN FINAL

Al finalizar el proyecto, se ha visto modificada la planificación inicial, a causa de diferentes problemas surgidos durante su elaboración. También hay que tener en cuenta que no siempre se ha podido trabajar la cantidad de horas diarias establecidas inicialmente.

En este apartado, se explican, únicamente, los apartados que han hecho variar la planificación pensada en un primer momento y los que han surgido durante el desarrollo del proyecto.

- **Desarrollo Rectángulo de Ajustes**
  En un primer momento no se pensó en la necesidad de incluir una herramienta que permitiese al usuario seleccionar aquella parte de la imagen sobre la cual se aplican los módulos de ajuste. Como se puede apreciar en el diagrama sobre la planificación final, estos módulos se han visto arrastrados a finalizar más tarde, ya que se han tenido que adaptar a esta nueva implementación. Se han dedicado quince días a la realización de esta tarea.

- **Desarrollo Exposición**
  Este módulo ha visto variada su planificación al incluir la nueva herramienta de selección de imagen. Esto ha provocado que se tenga que adaptar parte de su implementación y no se haya podido finalizar la tarea hasta una semana después de haber finalizado la anterior.

- **Desarrollo del histograma**
  Esta tarea se ha visto afectada del mismo modo que la anterior, se ha debido modificar la implementación sobre la obtención de los datos de la imagen. Es por esto, que se ha demorado su finalización hasta dos semanas después de haber finalizado el rectángulo de ajustes.

- **Desarrollo del balance**
  Este módulo se ha desarrollado en el tiempo previsto, pero se tuvo que posponer su inicio dadas sus características de funcionamiento ya que en ese momento no se disponía de una cámara a color.

- **Desarrollo del enfoque**
  Esta tasca ha visto recortada su duración por dos motivos: finalmente no se ha visto afectada por el período de vacaciones y, al disponer de más tiempo, se ha podido acortar su desarrollo dedicándole más horas al día.

- **Desarrollo configuración automática**
  Este módulo se ha propuesto una vez avanzado el desarrollo del proyecto como una herramienta adicional al software desarrollado.

- **Realización de pruebas**
  Evidentemente, esta tarea ha debido adaptarse a los cambios efectuados por las anteriores ya que se han realizado constantemente pruebas mientras se iban desarrollando módulos.

- **Documentación**
  Para no demorar el desarrollo del proyecto, se ha decidido ir documentando los pasos realizados en cada etapa del proyecto y dejar la elaboración de la memoria para la parte final de este. También se toma constancia de bibliografía usada en cada momento para una posterior recopilación al realizar la memoria.

Como consecuencia, este apartado se realiza paralelamente a los puntos referentes al desarrollo y estudios teóricos.

- **Elaboración de la memoria**
  La realización de la memoria se ha hecho en quince semanas ya que ha coincidido en que un servidor ha empezado su carrera profesional al llevar dos semanas con la elaboración de la misma.

- **Presentación**
  Este apartado no ha presentado diferencias respecto a la planificación inicial. El tiempo destinado a esta actividad continúa siendo de dos semanas.

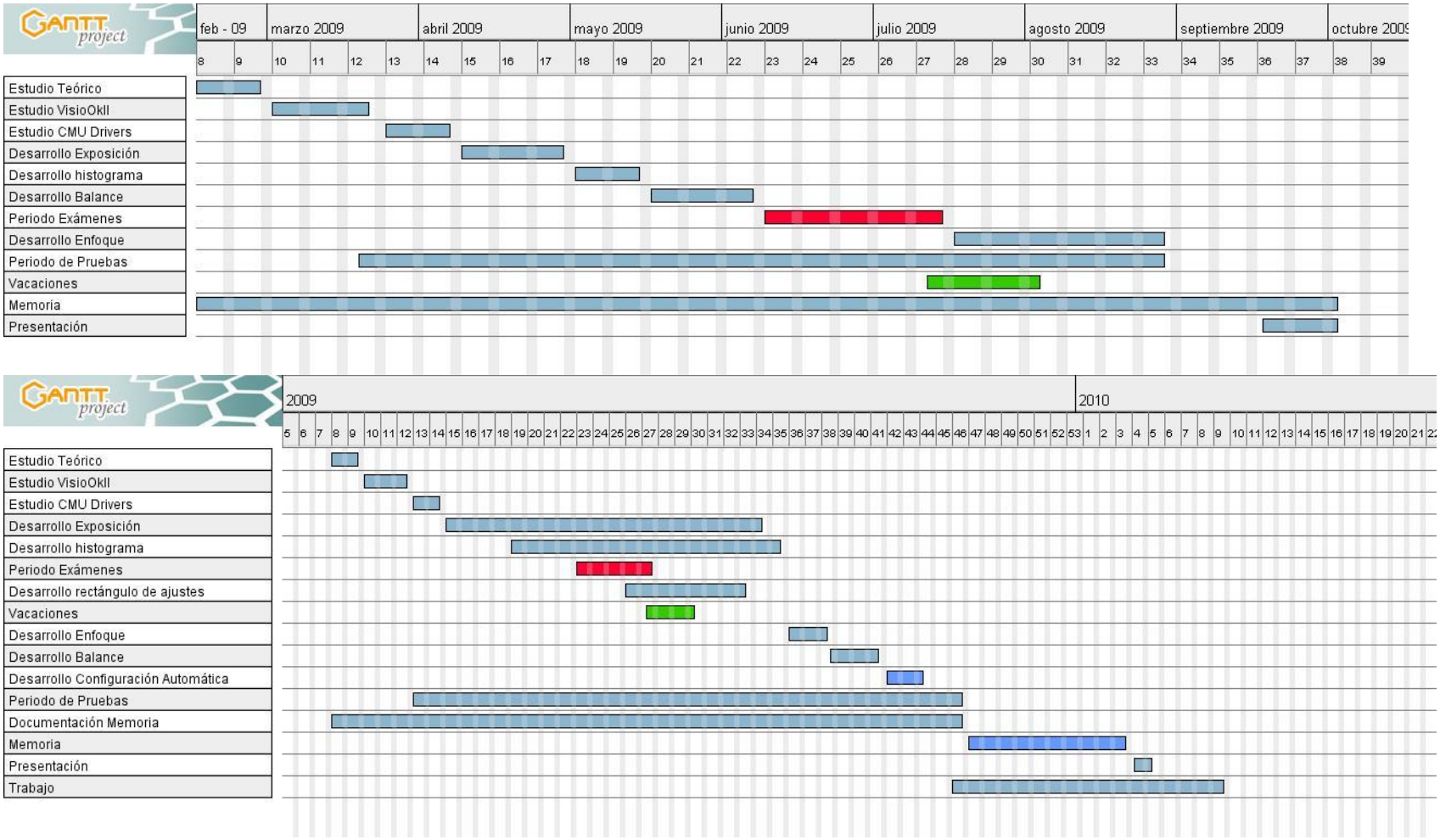

Figura 2. 1 Planificación Inicial y Final

2.3 ESTUDIO DE VIABILIDAD

Para la realización de este proyecto se requieren conocimientos de procesamiento de imágenes i programación orientada a objetos, en este caso *C++*. Se ha de tener en cuenta que el proyecto se desarrolla sobre un software en proceso de creación en el *Centre de Visió per Computador* de la *Universitat Autònoma de Barcelona* y, por esto, es necesario un estudio sobre el funcionamiento de este software.

Para poder hacer el proyecto se ha proveído, de manera temporal, una cámara *CMOS/CCD*, una lente y un cable *Firewire* con el que conectar la cámara a un ordenador. Para poder lograr esta conexión, es imprescindible tener un puerto *Firewire* en dicho computador.

VisionOkII se implementa sobre *Microsoft Visual Studio 2005*. Esta herramienta ha sido adquirida al *Servei d'Informàtica* de la *Escola Tècnica Superior d'Enginyeria* y, obviamente, es la herramienta sobre la que se ha desarrollado la parte de implementación del proyecto.

En un caso real, es decir, un cliente decide contratar el desarrollo de un software de este tipo hay que tener en cuenta varios puntos para establecer su coste. La suma de las horas trabajadas por el ingeniero.

- Según la planificación inicial, se trabajan 20 horas cada una de las veinticinco semanas en las que se ha llevado a cabo la realización del proyecto. Creyendo que un ingeniero cobra 30 €/hora, el coste total del proyecto asciende a 15.000€. Teniendo en cuenta la planificación final, se produce una desviación sobre el coste inicial del proyecto de 4.200 €, provocando que el presupuesto final sea de 19.200 €.

- Un PC de sobremesa Intel Core i7-860 con 4 GB de memoria RAM DDR3 1333 MHz, un disco duro con capacidad de 1 TB, monitor TFT de 19'' panorámico y sistema operativo Microsoft Windows 7 Professional asciende a un total aproximado de 1100 €.

- El importe de la cámara *CMOS/CCD*, una lente y una tarjeta con puerto *Firewire*, además de un cable para establecer la conexión, puede oscilar entre los 1000 € y 6000 €.

- Por último, el software utilizado es *Microsoft Visual Studio 2005*. Este software es propietario y, por tanto, se debe comprar una licencia cuyo valor es de aproximadamente 870 €.

Por tanto, el coste final del desarrollo de este proyecto oscila entre los 22000 € y los 26000 €, dependiendo de las calidades tanto del ordenador y de los componentes de la cámara.

Capítulo 3

CONCEPTOS PREVIOS

En este apartado se detallan los conceptos necesarios para poder desarrollar este proyecto. Primero, se va exponer un estudio teórico de la librería *MFC* (*Microsoft Foundation Classes*) necesaria para la implementación del software a desarrollar, ya que *VisionOkII* es una aplicación que usa este *framework* y es importante comprender su funcionamiento. Seguidamente, se muestra, de manera esquemática, la parte del software *VisionOkII* que está involucrada directamente con la realización de este proyecto. Finalmente, se hace un estudio de los conceptos relacionados con el procesamiento de imágenes que posteriormente se implementarán.

3.1 MFC (MICROSOFT FOUNDATION CLASSES)

La biblioteca *MFC* es un marco de trabajo de aplicaciones para la programación en *Microsoft Windows*. Desarrollada en *C++*, *MFC* proporciona gran parte del código necesario para administrar ventanas, menús y cuadros de diálogo; realizar operaciones básicas de entrada y salida de archivos; almacenar colecciones de objetos de datos, etc. Dada la naturaleza de la programación con clases de *C++*, es fácil ampliar o reemplazar la funcionalidad básica que proporciona este marco de trabajo. [1]

MFC otorga un fácil acceso a elementos de la interfaz de usuario. Además, simplifica la programación con bases de datos (en este proyecto no se trata).

VisionOkII aprovecha muchas de estas ventajas, además de la posibilidad de usar una interfaz de múltiples documentos (*MDI*). Las aplicaciones *MDI* permiten múltiples ventanas de marco que están formadas por un marco y su contenido. Este contenido mostrado se define como una vista del documento sobre el que se trabaja. Hay que indicar que estas ventanas están abiertas en la misma instancia de una aplicación. Es decir, *MDI* proporciona una ventana en la cual pueden abrirse múltiples ventanas secundarias y contienen cada una un documento independiente. Es posible que en algunas aplicaciones, las ventanas secundarias puedan ser de diferentes tipos, por ejemplo, en *VisionOkII* tiene documentos proyecto y documentos imágenes, entre

otros. En estos casos, es posible que la barra de menús pueda variar a medida que se van activando las diferentes ventanas secundarias.

El marco de trabajo *MFC* está basado en los conceptos de documento y vista. Un documento es un objeto de datos con el que interactúa el usuario mediante una vista del documento, por tanto, una vista siempre estará asociada a un documento. El documento se crea con los comando *Nuevo* o *Abrir* del menú *Archivo* y suele guardarse en un archivo. Los documentos estándar de *MFC* derivan de la clase *CDocument* y las vistas derivan de la clase *CView*. No obstante, hay que remarcar que existen otro tipo de clases, que añaden diferentes funcionalidades y que derivan, o bien directamente de las clases base o bien mediante una clase que hereda de éstas. [1]

3.1.1 ARQUITECTURA DOCUMENTO/VISTA

VisionOkII aparte de trabajar en un entorno *MDI* también aprovecha la arquitectura Documento/ Vista. La relación existente entre los dos elementos es de *1* a *N*, es decir, un documento puede estar asociado a múltiples vistas pero una vista sólo puede estar asociada a un documento. Este es el motivo por el que principalmente se usa esta arquitectura ya que nos permite mostrar datos diferentes de un mismo documento de manera simultánea, como por ejemplo, mostrar en una vista una hoja de cálculo y en otra mostrar una gráfica.

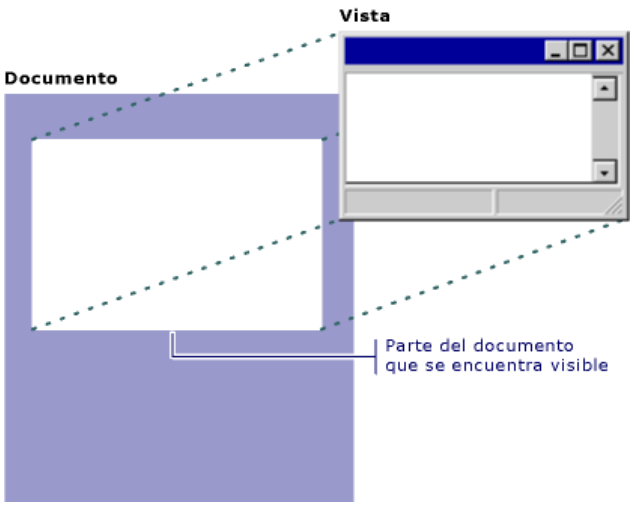

Figura 3. 1 Arquitectura documento Vista

El documento almacena los datos y coordina la actualización de todas las vistas del este. Las vistas se encargan de la presentación de los datos y de la interacción con el usuario. En el caso de que se produzca la modificación de alguno de los datos mostrados por la vista, ésta se encargará de devolver los cambios realizados al documento y llamará a *CDocument::UpdateAllViews* que se encarga de hacer una petición de actualización a todas las vistas del documento. Esta es una de las ventajas que proporciona esta relación. De esta manera, no se producen incongruencias entre las vistas del documento. [2]

Dejando de lado las *CDocument* y *CView*, existen dos elementos de notoria importancia que sin ellos no se podría implementar esta arquitectura, las clases *CFrameWnd* (o derivadas) y *CDocTemplate* (para la realización del proyecto se usa la clase derivada *CMultiDocTemplate* ya que trabajamos en una interfaz *MDI*). La primera, nos proporciona el marco que engloba las vistas de un documento. La segunda, nos permite relacionar un documento con sus vistas y una ventana de marco. [2]

En la mayoría de aplicaciones de usuario, esta arquitectura es ideal ya que simplifica y reduce el código redundante que se pueda generar. Sin embargo, existen situaciones o casos en que no es adecuada, por ejemplo, en aplicaciones sencillas o en caso de tener una aplicación o gran parte de ella con la vista y los datos mezclados, es muy probable que no nos convenga renovar el código ya que puede resultar muy costoso.

3.1.2 CLASE CFORMVIEW Y CDIALOG

En este proyecto se utilizan dos tipos de clase para representar datos en un formulario. Por un lado, la clase *CFormView* hereda de la clase vista *CScrollView* y, por el otro, está *CDialog*. Ambas clases muestran un formulario que el usuario es capaz de manipular con la diferencia que *CFormView* es una vista de una clase documento. [1]

3.1.2.1 CFORMVIEW

Una vista de formulario no es otra cosa que una vista que contiene controles de manera que el usuario puede interactuar con ellos. Estos controles se establecen en una plantilla de recursos de formulario y facilitan la manipulación de datos.

Esta clase proporciona métodos que son usados durante este proyecto heredados de la clase *CView*. A continuación se definen los métodos de esta clase utilizados a lo largo del desarrollo de este proyecto:

- ***GetDocument*:**
  Retorna el documento asociado a la vista.

- ***OnDraw***
  Es llamado para mostrar una imagen del documento para su visualización por pantalla. Es necesaria su implementación.

- *OnUpdate*: Es llamado para notificar a la vista de que el documento ha sido modificado.

- OnInitialUpdate: Es llamado al iniciarse una vista por primera vez.

3.1.2.2 CDIALOG

Esta clase presenta una caja de diálogo en la cual el usuario puede visualizar y manipular datos. Estos objetos pueden usarse de diferentes modos. *MFC* proporciona los llamados *AfxMessageBox* que están predefinidas y sirven para mostrar mensajes al usuario. También hay otro tipo de ventanas prediseñadas como son las destinadas a abrir documentos, guardarlos, imprimirlos, etc. Pero las realmente importantes en el desarrollo de este proyecto son las que personalizables ya que sirven para hacer prácticamente cualquier tipo de formulario. Estos diálogos se diseñan con facilidad gracias a las posibilidades que ofrece *Microsoft Visual Studio*.

Además, existen dos tipos de diálogos:

- **Modales:**
  Este tipo de diálogos obligan al usuario a interactuar con él antes de proseguir con la ejecución del resto de la aplicación. Para visualizarla hay que llamar a su método *CDialog::DoModal*. Inicialmente, estos diálogos incorporan los botones *aceptar* y *cancelar*. Si bien el segundo cierra la ventana del diálogo, el primero ejecuta el método *OnOK*, que permite actualizar y validar los datos que contiene el formulario.

- **No modales:**
  Estos diálogos permiten al usuario continuar con la ejecución del programa aún estando abiertos. Para visualizarlos hay que llamar al método CDialog::Create.

Para hacer uso de estas cajas, primero hay que definir un objeto de una clase que derive de *CDialog*. Entonces se dispara el constructor por defecto que da paso al diálogo. Esto se puede realizar mediante un puntero.

3.1.3 OTROS CONCEPTOS DE IMPLEMENTACIÓN

A parte de los conceptos introducidos en los anteriormente, también es necesario hacer hincapié en el dibujo de formas y/o líneas sobre un formulario, además de ver propiedades de los formularios como son el tamaño y la localización de este.

Para controlar el tamaño y localización de un formulario en pantalla se usa la estructura *WINDOWPLACEMENT*. Como se puede observar en la **Figura 3.2**, esta estructura contiene una estructura *RECT* que permite establecer el tamaño y posición cambiando los campos de dicha estructura (top, bottom, left, right). [4]

```
typedef struct _WINDOWPLACEMENT {
    UINT length;
    UINT flags;
    UINT showCmd;
    POINT ptMinPosition;
    POINT ptMaxPosition;
    RECT rcNormalPosition;
} WINDOWPLACEMENT;
```

Figura 3. 2 Estructura *WINDOWPLACEMENT*

Estas modificaciones se deben realizar usando las funciones *GetWindowPlacement*, para obtener el *WINDOWPLACEMENT* actual, y *SetWindowPlacement*, para establecer el nuevo *WINDOWPLACEMENT*.

Por último, se utilizan la clase *CClientDC* para establecer el contexto del área de cliente que tiene una ventana. En este proyecto se usan los métodos de *CClientDC* llamados *CClientDC::FillRect* y *CClientDC::SetPixel*, entre otros, para dibujar un rectángulo y un píxel respectivamente. En la **Figura 3.3** se puede observar cómo se dibuja un píxel de color rojo en la coordenada *(1,1)* de un *CFormView*:

```
Void CFormView::PaintPixel() {
    CClientDC pDC(this);
    OnPrepareDC(&pDC);
    pDC.SetPixel(1,1,RGB(255,0,0));
}
```

Figura 3. 3 Ejemplo dibujar píxel

A parte de la clase mencionada anteriormente, también se han usado objetos gráficos proporcionados por la librería *MFC*. Estos objetos proporcionan herramientas de dibujo que se pueden utilizar en diferentes contextos, por ejemplo, para el dibujo del área de ajuste desarrollado en el proyecto. Los objetos más usados pertenecen a las clases CBrush y CPen y son usados en el módulo del histograma y en el de enfoque. [3]

3.2 IIDC 1394-BASED DIGITAL CAMERA SPECIFICATION

IIDC 1394 es la especificación de cámaras digitales *firewire* existente actualmente. Por tanto, todos los fabricantes mundiales deben ceñirse a la norma dictada por este estándar. Consecuentemente, cualquier usuario que quiera establecer una comunicación con la cámara para enviar o recoger datos debe atenerse a la normativa igualmente.

La información necesaria para el desarrollo del módulo de configuración de la cámara consiste, básicamente, en el estudio de las diferentes configuraciones que puede soportar una cámara.

La resolución y la información que puede contener un píxel vienen definidas por dos parámetros de la cámara. Estos son el formato y el modo de funcionamiento. En el anexo adjunto a la memoria se pueden apreciar los diferentes tipos de configuración definidos en el *standard*. En la **Figura 7.19** se puede apreciar lo comentado anteriormente. También hay una tabla que establece la velocidad de captura en *frames* por segundo mínima que debe soportar como mínimo cada una de las configuraciones posibles.

Obviamente, el *standard* ofrece mucha más información adicional pero en este caso no es necesario comentarlo ya que no se hace referencia en el resto de la memoria. [5]

3.3 VISIONOKII

En este apartado, se va a proceder a desgranar únicamente la parte de *VisionOkII* que se verá afectada directamente por la realización de este proyecto. Esto es debido a que dicho software es muy extenso y la explicación del resto de componentes es irrelevante para la evaluación del proyecto.

Antes de profundizar en las diferentes clases y métodos de *VisionOkII* se va a explicar, de manera general, como se obtienen los datos procedentes de la cámara.

3.3.1 C.M.U. DRIVERS

La captura de imágenes procedentes de cámaras digitales, es uno de los pasos más complejos en el desarrollo de *VisionOkII*. Para cumplir con este objetivo, *VisionOkII* ha sido implementado usando una librería de C++ para cámaras que cumplen con el *1934 Digital Camera Specification* publicada por *1934 Trade Association*, una asociación sin ánimo de lucro se encarga de promocionar activamente el *IEEE 1934 Serial Bus Standard*. También están involucrados en temas relacionados con este *standard* ya sea con empresas de desarrollo o con el mercado de componentes electrónicos. [6]

Esta librería ha sido desarrollada por el *The Robotics Institute* de la *Carnegie Mellon Univesity*. La ventaja que proporciona esta librería es que es software gratuito y está a disposición de toda la comunidad de desarrolladores. Otro punto fuerte de este software es su facilidad de uso y configuración.

A continuación, se exponen los diferentes métodos usados para interactuar con la cámara *firewire*.

La conexión entre la cámara *firewire* usada y el software de configuración se realiza mediante los métodos expuestos a continuación:

- ***Int CheckLink()***
  Esta llamada a la función se realiza para comprobar si existe alguna cámara firewire conectada.

- ***Int SelectCamera(int node)***

Tal y como indica su nombre, se utiliza esta función para seleccionar una cámara de todas las que puedan estar conectadas. Dicha selección se materializa gracias al parámetro que recibe.

- **Int InitCamera()**
  Inicializa la clase C1934Camera para la cámara seleccionada anteriormente.

Estas tres funciones retornan un valor entero para indicar al usuario si se han ejecutado con éxito o no.

Una vez realizados estos pasos, hay que añadir los parámetros de configuración de la cámara. Antes de añadir estos parámetros hay que verificar si son compatibles con la cámara que se está usando. En este proyecto se realizan las siguientes comprobaciones:

- ***Bool HasVideoFormat(int format)***
  Dado un formato representado por un entero, retorna si la cámara es compatible con este tipo de formato (*true*) o si no lo es (*false*).

- ***Bool HasVideoMode(int format, int mode)***
  Esta función establece si los parámetros *format* y *mode* son igualmente compatibles por la cámara.

- ***Bool HasVideoFrameRate(int format, int mode, int rate)***
  Como las anteriores indica si los parámetros de configuración son compatibles o no con la cámara.

Una vez obtenidos unos parámetros de configuración correctos, se deben enviar a la cámara. Este paso se realiza mediante las siguientes funciones:

- ***Int SetVideoFormat(int format)***
  Establece el formato de video.

- ***Int SetVideoMode(int mode)***
  Se envía el parámetro *mode* a la cámara.

- ***Int SetVideoFrameRate(int rate)***
  Indica a la cámara los *frames* por segundo.

Tal y como sucede con las funciones vistas anteriormente, estas también devuelven un entero para indicar si han podido realizar el ajuste de la cámara o no.

Además de estas funciones, también se usan otras encargadas de devolver los valores actuales que contiene la cámara. Estos valores están representados como un entero.

- ***Int GetVideoFormat()***
- ***Int GetVideoMode()***
- ***Int GetVideoFrameRate()***

Una vez configurada la cámara con los valores especificados, se empieza con la captura de imágenes. Las funciones usadas en este proyecto para tal fin son las siguientes:

- **Int StartImageCapture()**
  Inicializa el proceso de adquisición de imágenes.
- **Int CaptureImage()**
  Graba un *frame* de la cola. Ambas funciones retornan un valor que indica si se ha podido proceder correctamente o no.

Por último, se hace hincapié en que existen más funciones de configuración pero en este proyecto sólo se han utilizado estas.

Una vez visto el mecanismo de interactuación con la cámara, se muestra el estudio realizado sobre las clases y métodos de *VisionOkII* necesarios para la realización de este proyecto.

3.3.2 IMPLEMENTACIÓN DE VISIONOKII

Como se ha comentado anteriormente, *VisionOkII* usa la arquitectura Documento/Vista. Las clases que componen este concepto son *CVOkImageDoc* (derivada de *CDocument*) y *CVOkImageView* (derivada de *CScrollView*). Estas son las clases sobre las cuales se trabaja en este proyecto y se relacionan mediante la clase *CMultiDocTemplate*, explicada en el apartado anterior. La relación se crea al inicializar la instancia de la aplicación *VisionOkII* (*CVisionOkIIApp::InitInstance*), concretamente

en la clase *CVisionOkIIApp* que hereda de la clase *CWinApp*. Esta última es la clase principal de las aplicaciones en MFC. Encapsula la inicialización, ejecución y cierre de una aplicación para Windows. Estas aplicaciones deben tener un único objeto de una clase derivada de *CWinApp*. Esta clase deriva de *CWinThread*, que representa el subproceso principal de ejecución de la aplicación y puede tener uno o varios subprocesos.

Por último, se añade el *Document Template*, a la lista de disponibles de la aplicación. A continuación, en la **Figura 3.4** se muestra el código de dicha operación:

```
m_pDTImage = new CMultiDocTemplate(
        IDR_IMAGETYPE,
        RUNTIME_CLASS(CVOkImageDoc),
        RUNTIME_CLASS(CChildFrame),
        RUNTIME_CLASS(CVOkImageView));
AddDocTemplate(m_pDTImage);
```

Figura 3. 4 Instancia *CMultiDocTemplate*

En el ejemplo se puede observar que en el paso de parámetros incluimos la clase documento, la clase vista y la clase de ventana de marco. En el proyecto se han añadido nuevos objetos *CMultiDocTemplate* para los diferentes módulos desarrollados. Esto se puede comprobar en el siguiente capítulo.

A continuación, se detalla el funcionamiento de las clases *CVOkImageDoc* y *CVOkImageView*, especialmente esta última ya que es la clase desde donde parte el desarrollo del proyecto.

3.3.3 CLASE CVOKIMAGEDOC

Esta clase deriva directamente de la clase *CDocument* y contiene todos los datos del documento. Básicamente, esta clase usada para trabajar con la clase *CVImage* que tiene asociada, ya sea como un elemento de la propia clase o a través de un elemento *VOkView* y del que se puede obtener dicha clase. *CVImage* contiene toda la información de la imagen o *frame* captado por la cámara. Los métodos usados en

CVOkImageDoc son *CVOkImageDoc::GetImage()* y *CVOkImageDoc::SetImage*. Como indican sus nombres, el primero se usa para obtener el *CVImage* asociado a la clase documento y, el segundo para asociar un *CVImage* a la clase documento des de donde se esté invocando.

Para poder trabajar con los datos propios de la imagen se deben conocer algunos métodos de la clase *CVImage* que proporcionan los recursos suficientes para poder desarrollar el proyecto. Uno de ellos es *CVImage::Cut* que, pasados como parámetros las coordenadas (x,y) del punto origen y punto final, nos devuelve otro objeto *CVImage* con el área de la imagen cortada. El método *CVImage::GetRaster* devuelve un puntero a la posición de memoria donde está almacenada la imagen a tratar. *CVImage::GetSizeX* y *CVImage::GetSizeY* proporcionan el tamaño de la imagen para las coordenadas x e y. Estas junto con *CVImage::GetCha*, que retorna el canal de la imagen, permite calcular el tamaño real de la imagen guardada en memoria. Por ejemplo, para una imagen en color el canal es de tres, por tanto, para calcular el tamaño real de la imagen se debe multiplicar el canal por el número de píxeles para la coordenada x y también por el número total para la coordenada y.

Por último, destacar que se han usado dos métodos de la clase *CVOkImageDoc* para obtener la clase *CVOkImageView* asociada a la clase documento. Esto es necesario ya que la principal fuente de trabajo se realiza sobre elementos de esta clase. El método *GetFirstViewPosition()* retorna un elemento de tipo *POSITION* que representa la posición de la primera vista de la lista de vistas del documento. El segundo método a comentar es *GetNextView(POSITION)* que obtiene la vista asociada al elemento *POSITION*. Por defecto, dicha vista será siempre *CVOkImageView*, por tanto, es una buena manera de obtener la vista que interesa.

3.3.4 CLASE CVOKIMAGEVIEW

CVOkImageView hereda directamente de la clase *CScrollView* y a continuación se detalla alguno de sus métodos que se ven involucrados en este proyecto.

CVOkImageView::OnDraw es el método encargado de imprimir por pantalla el contenido del documento, en este caso una imagen, en el marco desde donde se llama. Pero *CVOkImageView ::OnDraw* no sólo dibuja la imagen que contiene *CVOkImageView* sino que también dibuja elementos gráficos definidos por el usuario.

Estos elementos son objetos que definen áreas de la imagen y que se usan para diferentes cálculos que realiza el software.

Esta última aclaración es importante porque en este proyecto se ha creado un elemento gráfico, parecido a los anteriormente comentados, del cual los módulos desarrollados extraen la parte de información de la imagen que contiene dicho elemento. Este objeto pertenece a la clase *VOkRectangleAjustes* y no se hace referencia únicamente en el método *CVOkImageView::OnDraw* sino que también se hace en el resto de métodos que se van exponer inmediatamente.

El método *CVOkImageView::OnKeyDown* se ejecuta cuando el usuario presiona una tecla. Gracias a este método es posible asociar teclas a funcionalidades. Por ejemplo, se puede asociar la tecla "*+*" con una función que aumente el zoom de una imagen.

Otro método a tener en cuenta es *CVOkImageView::OnLButtonDown*. Este es el encargado de verificar si se produce el evento sobre algún elemento gráfico. En caso de producirse sobre uno de estos elementos, incluido el elemento *VOkRectangleAjustes*, lo resalta como elemento seleccionado y comprueba si ha sido trasladado de una posición a otra.

CVOkImageView::OnUpdate es un método que proporciona la clase *CView* y es heredado por parte de *CVOkImageView*. Este es llamado después de que se produzca un cambio o modificación en la vista de documento actual. *CVOkImageView::OnUpdate* es llamado por *CDocument::UpdateAllViews* y permite actualizar la pantalla para poder visualizar los cambios.

Para finalizar este apartado, se hace referencia a *mapa de eventos*. En este proyecto, básicamente, se usan *ON_COMMAND* y *ON_UPDATE_COMMAND_UI* en las cuales se pasan como parámetros un mensaje de comando y una función miembro. En la figura X se muestran dos ejemplos del uso de estas macros:

```
ON_COMMAND(ID_VER_ZOOMX1, OnVerZoomx1)
ON_UPDATE_COMMAND_UI(ID_VER_ZOOMX1,
OnUpdateVerZoomx1)
```

Figura 3. 5 *ON_COMMAND* y *ON_UPDATE_COMMAND_UI*

En el caso de *ON_COMMAND*, la función miembro se ejecutará cuando se produzca un evento en el mensaje de comando. Por ejemplo, activar una opción del menú *ampliar zoom*. En este caso se hará una llamada a la función asociada que ejecutará la acción de ampliar el zoom de una cierta imagen. En cambio, *ON_UPDATE_COMMAND_UI* no está asociado simplemente a que se produzca un evento sino que más bien al producirse una actualización del ítem. Por ejemplo, se puede tener una opción de ver la barra de estado y esta puede permanecer *checked* o no en función de si la barra de estado está visible al usuario o no. Esta acción se realizaría mediante la función miembro que tiene asociado el ítem.

Estos ejemplos presentados son semejantes a los utilizados en la elaboración del proyecto, ya que los eventos están enmarcados en el menú que aparece al trabajar con la vista de documento *CVOkImageView*.

3.3.5 CREACIÓN Y CONFIGURACIÓN DE CÁMARAS

VisionOkII tiene un elaborado sistema de creación y configuración de cámaras. Para poder desarrollar la cámara autoajustable es necesario tener unos conocimientos previos sobre cómo se realiza manualmente.

3.3.5.1 CREACIÓN DE UNA NUEVA CÁMARA

El proceso de creación de la cámara se realiza a partir de activar el diálogo *CNewCameraDlg*. Este diálogo permite introducir un nombre si se desea crear una nueva cámara o seleccionar una cámara ya existente a configurar. Una vez seleccionada la cámara, se establece su configuración. Una vez presionado el botón Aceptar se hace una llamada a *VOkCamera::NewCamera*, cuyo primer parámetro es el nombre de la nueva cámara y, el segundo, corresponde al dispositivo relacionado con el proyecto. Este método se asegura de que no haya otra cámara con el mismo nombre y que el proyecto no esté en línea. Una vez hechas estas comprobaciones se hace un *VOkDevice::InsertCamera*, del dispositivo recibido como parámetro. *VOkDevice::InsertCamera* crea un objeto *FirewireCamera* y lo añade a la lista de cámaras del dispositivo actual.

La realización de estos pasos asegura la creación de la nueva cámara que aparece en la vista árbol del proyecto.

Los parámetros de esta nueva cámara se recogen en un objeto de la clase *VOkParametersList*, cuyos campos corresponden a cada uno de los parámetros de la cámara.

El acceso necesario para manipular los valores de estos parámetros se realiza mediante el método *VOkParametersList::GetValueByIdent* que, gracias a la *id* que recibe como primer argumento, permite acceder al objeto *VOkValue* relacionado con dicha *id*. *VOkValue* es la clase base de todo un seguido de clases que derivan de esta, como por ejemplo, *VOkValueInt*. Según el parámetro accederemos a objetos de unas clases u otras. A continuación, se muestra un ejemplo sobre cómo acceder al valor de un parámetro:

```
dynamic_cast<VOkValueInt*>(
     m_pCam->GetValueByIdent("FrameRate",NULL))
     ->Set(2);
```

Figura 3. 6 Obtención del valor de un parámetro

En este caso, se accede al parámetro *FrameRate* para establecer su valor en 2.

3.3.5.2 ESTABLECER PROYECTO EN LINEA

Una vez creada y configurada la cámara se debe realizar el proceso de proyecto en línea para poder establecer la comunicación entre la cámara y el software creado.

El proceso comienza con el cierre de todos los dispositivos abiertos hasta el momento si los hay. Una vez, realizada esta acción se procede a reabrirlos uno por uno mediante el método *VOkDevice::Open()*. Este método se encarga de coger todas las cámaras que tiene asociadas el objeto dispositivo y realiza un *FirewireCamera::Open()* de cada una de ellas. Este método crea un objeto *FirewireCameraImp* que es el encargado de gestionar todo lo referente a la configuración y comunicación con la cámara *firewire*.

El constructor de la clase *FirewireCameraImp* se encarga de leer todos los parámetros de configuración introducidos por el usuario, los introduce en la cámara y, finalmente, la deja preparada para capturar *frames*. Estas acciones se materializan mediante los métodos comentados anteriormente.

FirewireCameraImp::GrabBMP se encarga de comprobar si se ha realizado alguna modificación de los parámetros introducidos en la cámara. Si hay algún parámetro modificado, se envía a la cámara. Una vez hechos todos los cambios, se restablece la captura de imágenes.

3.4 CONCEPTOS TEÓRICOS A DESARROLLAR

Una vez vistos los conceptos necesarios para desarrollar sobre *VisionOkII* los módulos de configuración, se va a mostrar, teóricamente, en qué consisten dichos módulos.

3.4.1 HISTOGRAMA

El histograma no es más que un gráfico de columnas. Sobre el eje x tenemos las 256 posibles tonalidades que van desde 0 (correspondiente al negro) hasta el 255 (correspondiente al blanco). El eje Y se suele comprender desde 0 hasta el número máximo de píxeles de una cierta tonalidad. El histograma suele implementarse o bien en blanco y negro o bien en formato *RGB*(del inglés Red, Green, Blue), que hace referencia a la composición del color en términos de la intensidad de los colores primarios de los que está formado (rojo, verde y azul). Este modelo está basado en síntesis aditiva, por tanto, es posible representar un color mediante la mezcla de estos tres colores.

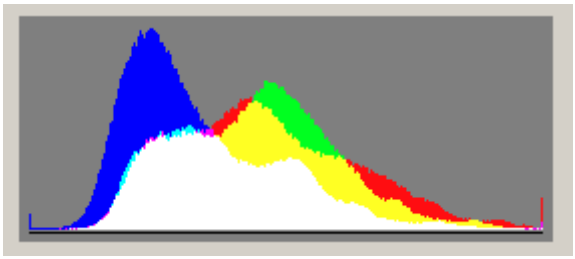

Figura 3. 7 Histograma

Como se puede apreciar en el ejemplo de la **Figura 3.7**, podemos observar que el histograma suele ir acompañado de información numérica que ayuda a extraer

conclusiones sobre el histograma de una imagen. Podemos encontrar información sobre la cantidad de píxeles que contiene la imagen, un promedio de todas las tonalidades, la cantidad de píxeles con una cierta tonalidad, etc.

3.4.2 EXPOSICIÓN

La exposición es la acción de someter un elemento fotosensible a la acción de la luz, es decir, es la cantidad de luz que irradia un cierto elemento. Una correcta exposición determina la calidad final de una fotografía, por tanto, debe ser el primer elemento a calibrar a la hora de obtener una buena imagen.

Según el grado de exposición una fotografía puede estar subexpuesta, expuesta o sobreexpuesta.

- **Subexposición**
  La imagen obtenida carece de suficiente luz respecto a la escena original, es decir, los píxeles de la imagen tienen un valor más cercano al 0 (negro) que al 255 (blanco).

- **Sobreexposición**
  La imagen contiene un exceso de luz respecto a la escena original, esto tiene como consecuencia que los valores de los píxeles se sitúen en un punto más cercano a 255 que a 0.
- **Exposición**
  La fotografía recoge la cantidad de luz fiel a la presentada por la escena original.

Figura 3. 8 Exposición

Fundamentalmente hay tres elementos que influyen en la exposición de una fotografía. Estos elementos son el obturador que determina el tiempo en que la luz incide sobre el sensor; la apertura del diafragma que, como su nombre indica, permite entrar más o menos luz en función de su estado; y la sensibilidad del sensor a la luz, cuanto más sensible sea este más se verá afectado por la exposición.

En la **Figura 3.8** se puede apreciar los diferentes tipos de exposición que puede sufrir una imagen según los parámetros que se introduzcan en los elementos anteriormente mencionados. La primera imagen corresponde a una correcta exposición, la segunda está sobreexpuesta y la tercera subexpuesta.

Una consecuencia directa de una mala exposición es la pérdida de información que se produce en la escena retratada. Esto se puede observar fácilmente en la **Figura 3.8** ya que podemos ver que en las dos últimas imágenes hay partes que no se pueden visualizar correctamente.

3.4.3 BALANCE DE BLANCOS

En fotografía y en procesamiento de imágenes, se utiliza una técnica llamada balance de blancos que pretende corregir el color de esta con el objetivo de eliminar colores dominantes, es decir, existe un color que predomina sobre el resto. Este concepto es fácilmente reconocible en las tonalidades neutras de una fotografía, como por ejemplo, una superficie blanca o una tonalidad grisácea. En la **Figura 3.9**, se pueden apreciar diferentes ajustes del balance de blanco, siendo las imágenes de la izquierda dos casos de un balance de blancos incorrecto. [9]

Figura 3. 9 Balance de blancos

También se observa el grado de temperatura de color de cada una de las imágenes. Este último concepto expresa la dominante de color de una fuente de luz determinada que emitiría un cuerpo negro calentado a una cierta temperatura, que varía según la distribución espectral de la energía y se expresa en *Kelvins*. Cuanto más cálida sea la luz (rojo-amarillo), más baja será la temperatura de color. En cambio, cuanto más fría sea la luz (azul), más alta será la temperatura de color. En la **Figura 3.10** se muestra la escala de temperatura de color que suelen usar las cámaras digitales. [13]

Figura 3. 10 Temperatura de color en la escala Kelvin

Las escenas dominadas por un único color pueden requerir un ajuste del equilibrio de blancos para ayudar a la cámara a reproducir los colores con más exactitud para garantizar que los blancos se van a mostrar blancos en la imagen final.

Actualmente, las cámaras digitales traen incorporado el balance de blancos automático y está basado en que la parte más brillante de la escena se ajusta a color blanco y la menos brillante a negro. Otras traen un software más avanzado y permiten varias opciones según donde esté situada la escena, como por ejemplo, en interiores, al sol, en sombra, etc.

En este proyecto, se apuesta por una configuración manual del balance de blancos. Este ajuste manual se basa en configurar el balance de blancos bajo una superficie blanca o gris neutro. De esta manera, se puede lograr un ajuste mucho más preciso y, por tanto, unos colores que se ajustan a la realidad de la escena.

3.4.4 ENFOQUE

El enfoque es el ajuste del objetivo a fin de obtener una imagen clara y definida del sujeto, lo que se hace desplazando hacia delante o hacia atrás el objetivo (lente), mediante un anillo de enfoque u otros sistemas similares. El resto de sujetos que aparecen a una distancia mayor o menor respecto al punto donde se enfoca, presentan borrosidad y poco detalle.

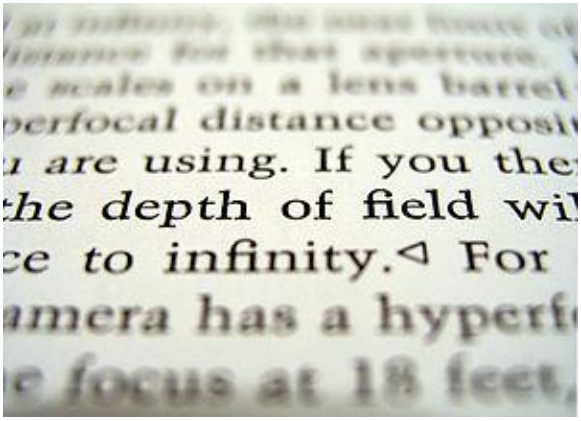

Figura 3. 11 Enfoque

3.4.4.1 INTRODUCCIÓN AL AUTOENFOQUE

El autoenfoque o autofoco es aquel mecanismo automático que es capaz de calcular el enfoque óptimo del motivo de una escena. Principalmente, existen dos tipos de autofoco, el activo y el pasivo. El primero, necesita conocer la distancia exacta entre la cámara y el motivo a enfocar, después sólo se trata de calibrar el enfoque según la distancia obtenida. El segundo, se basa en dos técnicas: la comparación de fases y la medida del contraste.

La comparación de fases trata sobre la triangulación de distancias mediante un par de sensores que hay tras la lente. La distancia final del motivo a enfocar se obtiene gracias al cálculo de la diferencia que hay entre las dos imágenes obtenidas. La medida del contraste, como bien hace suponer su nombre, obtiene el enfoque óptimo con el mejor contraste obtenido. [10]

Este módulo del proyecto calcula el mejor enfoque en función de la medida del contraste ya que, desde el punto de vista de desarrollo, no es posible conocer ni distancia real entre la cámara y el motivo de la escena ni se dispone de una cámara con sensores capaces de calcular distancia mediante la comparación de fases.

A simple vista, calcular el enfoque mediante el contraste puede resultar una idea un poco estúpida. Nada más lejos de la realidad. Cierto es que el contraste varía según las condiciones de iluminación y balance de blancos, pero también es cierto que las condiciones sobre las que se va a efectuar este automatismo se encargan de minimizar estos efectos. Hay dos motivos fundamentales: las condiciones de iluminación deben ser óptimas para la buena detección de objetos (principal objetivo de *VisionOkII*) y, el segundo motivo, se han creado las herramientas, posteriormente descritas, necesarias para obtener una mejor configuración de la cámara *Firewire*. Obviamente, el autofoco se debe ejecutar una vez la cámara esté correctamente configurada.

3.4.4.2 DETECCIÓN DE BORDES

El cálculo del mejor contraste se realiza mediante un algoritmo de detección de bordes. En imagen digital, se conoce como borde un cambio significativo en el nivel de gris entre dos o más píxeles adyacentes. Este tipo algoritmos funcionan mejor cuanto mayor es el contraste de una imagen. El único hándicap que tiene este tipo de algoritmos es cuando se intenta resolver el cálculo en una superficie plana, es decir, no se aprecia el suficiente contraste aunque las condiciones sean óptimas. Este problema no debe ser ningún inconveniente en este proyecto puesto que el software sobre el que se trabaja está destinado a la detección de piezas u otros objetos y, en estas ocasiones, no suelen aparecer casos de este tipo. [10]

En la detección de bordes se utilizan filtros para calcular la dirección y magnitud del gradiente para cada píxel. La magnitud del gradiente muestra la intensidad del cambio de un píxel a otro adyacente.

Existen diferentes tipos de filtros para realizar estos cálculos. En este proyecto, se han tenido en consideración el *Operador de Sobel* y el *Operador de Prewitt* para las orientaciones vertical (y) y horizontal (x) ya que son los filtros más comúnmente utilizados. En la siguiente **Figura 3.12** se puede apreciar que el *Operador de Sobel* da un mayor énfasis al valor del píxel central. Con esto se logra que el ruido no tenga tanta influencia en la detección de bordes. Este es, principalmente, el motivo por el cual se ha decidido usar el *Operador de Sobel* para este proyecto. [11]

Operador de Sobel: $M_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$ $M_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$

Operador de Prewitt: $M_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$ $M_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$

Figura 3. 12 Operador *Sobel* y *Prewitt*

A continuación se muestra un ejemplo sobre el resultado obtenido de aplicar el *Operador de Sobel* sobre una imagen cualquiera.

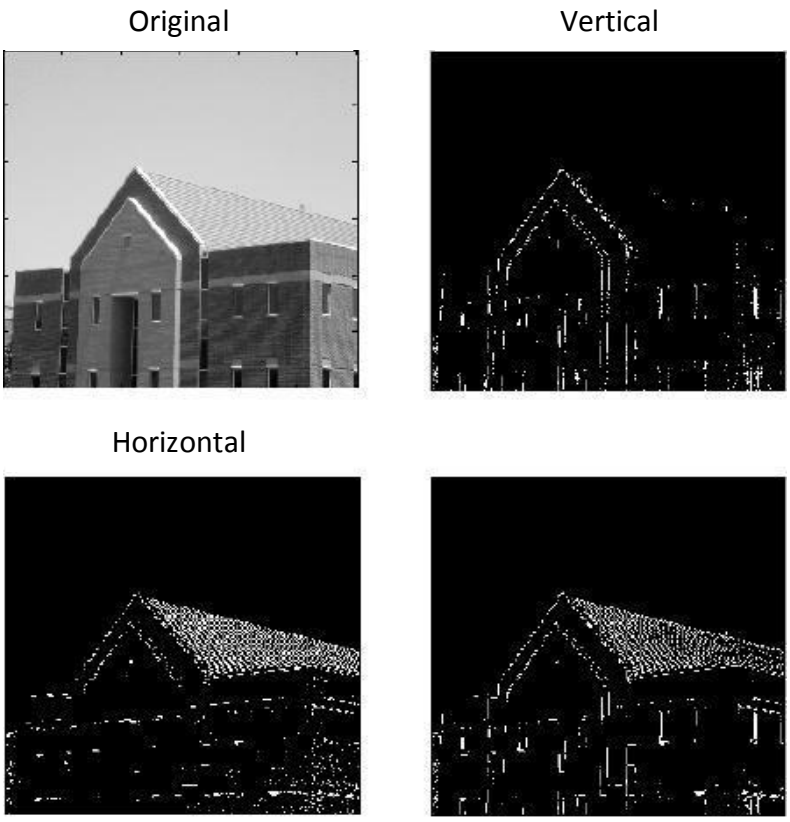

Figura 3. 13 Aplicación del operador *Sobel*

Finalmente, el algoritmo empleado para resolver el mejor enfoque para una imagen es *Tenengrad* ya que es uno de los algoritmos que presenta un mayor índice de acierto y su implementación es bastante sencilla.

3.4.4.3 ALGORITMO DE TENENGRAD

Este algoritmo se basa en obtener la máxima magnitud del gradiente. Está considerado uno de los algoritmos más robustos y que mejor funciona a la hora de calcular la calidad de una imagen.

El algoritmo funciona de la siguiente manera:

El valor *Tenengrad* de una imagen es calculado a partir del gradiente de cada píxel, obtenido gracias al uso de un filtro, en nuestro caso el operador de *Sobel* explicado anteriormente, y aplicando convolución sobre los ejes x e y de cada uno de los píxeles. [7]

Por tanto, la magnitud del gradiente se expresa como:

$$\mathrm{S(x,y)} = |\mathrm{G(x,y)}| = \sqrt{\left(\mathrm{I_x} * \mathrm{I(x,y)}\right)^2 + \left(\mathrm{I_y} * \mathrm{I(x,y)}\right)^2}$$

Donde I_x es el kernel que representa la aproximación del gradiente horizontal; I_y es el kernel que representa la aproximación del gradiente vertical; e *I(x,y)* representa el valor del píxel en la posición *(x,y)*.

Dado el alto grado de complejidad del cálculo a realizar para cada píxel, se presenta una modificación sobre la fórmula original que provoca una gran optimización de la velocidad de cómputo al no tener que realizar tantas operaciones.

$$\mathrm{S(x,y)} = |\mathrm{G_x(x,y)}| + |\mathrm{G_y(x,y)}|$$

Finalmente, el valor *Tenengrad* de la imagen se expresa:

$$\mathrm{TEN} = \sum_x \sum_y S(x,y)^2$$

Normalmente se suele utilizar un valor de umbral ($S(x,y)$ >*T*) con la intención de descartar aquellos valores procedentes del ruido de la imagen o de bordes insignificantes para la detección de bordes. En este proyecto se ha tomado la decisión de no incluir un umbral que descarte estos resultados intermedios ya que no se apreciaban cambios significativos en el caso de usarlos o no. [8]

3.4.4.4 OTROS ALGORITMOS USADOS

Existe otro algoritmo basado en la detección de bordes llamado *CGA* (Conteo de Gradiente Absoluto) que consiste en contabilizar los valores del módulo del gradiente que superan un cierto umbral. Básicamente, realiza los mismos cálculos que el algoritmo de *Tenengrad* pero en vez de ir sumando los valores retornados, se van contabilizando. [14]

Decir que este método ha sido probado en este proyecto pero se ha desestimado su uso ya que en la fase de test se comprobó que tenía una tasa alta de fallos comparado con el algoritmo anterior.

Capítulo 4

DESARROLLO

En este capítulo se muestra cómo se desarrollan los diferentes módulos. Según los requerimientos de este proyecto, se hace necesaria la elaboración de tres formularios, uno para el balance de blancos, otro para el histograma y, por último, uno para el módulo de enfoque. La herramienta de cálculo de la exposición de la imagen se realiza directamente sobre la clase de vista *CVOkImageView*, vista que muestra la imagen contenida por el documento. También se debe crear un objeto que consista en un área o porción de la imagen, manipulable por el usuario, cuya información sea usada por los diferentes módulos de configuración.

Por último, hay como restricción que los módulos sólo sean ejecutables cuando se esté trabajando con una vista de cámara de un proyecto, es decir, no se permite la ejecución de los módulos cuando se abra una imagen aleatoria desde el menú de archivo, por ejemplo, cargar una imagen aleatoria.

4.1 ÁREA DE AJUSTE

El área de ajuste de la imagen no es más que un rectángulo visto desde una perspectiva visual que selecciona una parte de la imagen para su uso durante el ajuste de la cámara. Realmente, es algo más que un simple rectángulo. Es una clase con la que el usuario puede interactuar, ya sea mediante el teclado o con el puntero. Esta clase se llama *VOkRectangleAjustes* y es semejante a las clases que heredan de *VOkValueGraph* aunque más sencilla.

El constructor de *VOkRectangleAjustes*, simplemente, contiene un par de variables booleanas que nos indican si dicho objeto ha sido seleccionado por el usuario o ha sido cambiado de posición. A parte de dichos objetos esta clase contiene cuatro variables enteras que nos permiten determinar las coordenadas donde se sitúa nuestro objeto *VOkRectangleAjustes*. Estas variables son *protected* y, por tanto, es necesario crear los métodos *get* y *set* para obtener y almacenar valores de ellas. También contiene otros métodos que son explicados a continuación:

- **BoundingBox(CDC &dc, int zoom)**
  Este sencillo método se encarga de devolver un objeto *CRect* con las coordenadas del elemento *VOkRectangleAjustes*. Este método es llamado cada vez que se ejecuta el método *CVOkImageView::OnUpdate*, de esta manera se actualiza el objeto y, como consecuencia, los datos que aporta a los módulos de configuración. También es llamado en el método *CVOkImageView::OnLButtonDown* para ser actualizado en el momento de ser movido o si debe estar en modo seleccionado o no.

- **Draw(CDC &dc, CVOkProjectDoc *pProject, int zoom)**
  Como indica su propio nombre, este método es usado para dibujar en el marco de la vista el propio rectángulo. En caso de estar seleccionado, imprime un rectángulo resaltado. Este método es ejecutado cada vez que se llama al método *CVOkImageView ::OnDraw*.

- **HitP(CDC &dc, int zoom, CPoint point)**
  Este elemento retorna el propio objeto si se ha hecho *click* con el cursor. Retorna *null* en caso contrario.

- **OnKeyDown(VOkView *pView, UINT nChar, UINT nRepCnt, UINT nFlags)**
  Este método se ejecuta cuando se pulsa alguna tecla en la vista *CVOkImageView*. Si es necesario cambia el tamaño o mueve el rectángulo por la vista actualizando los valores de las coordenadas que lo definen. Retorna true si se ha producido algún cambio. De esta manera, es fácil saber si el objeto ha cambiado su posición o no.

- **Track(CDC &dc,int zoom,VOkView *pView,CPoint point)**
  Este método se ejecuta en el momento de hacer *click* en el botón izquierdo del ratón. Este método traslada el rectángulo tanta distancia como se haya movido el *mouse*, permitiéndonos ver en todo caso donde se va a situar este en el momento de soltar dicho botón.

4.2 EXPOSICIÓN

Este módulo se integra directamente sobre la vista *CVOkImageView*, concretamente en el rectángulo creado mediante la clase *VOkRectangleAjustes*.

El usuario puede activar esta aplicación mediante el menú *Ver* disponible al visualizar una vista de la cámara.

En caso de visualizar píxeles subexpuestos, es decir, con tendencia a ser oscuros, la función no actúa para todos aquellos píxeles que no contengan ningún color negro puro, es decir, con un valor de 0. Aquellos que contengan uno o dos colores a 0, estos se pondrán a 255. De esta manera, el usuario puede visualizar qué píxeles están subexpuestos y más concretamente que color o colores. En caso de tener los tres colores con un valor de 0, el píxel se muestra tal cual, en negro.

En la **Figura 4.1** se puede visualizar los píxeles sobreexpuestos (derecha) y subexpuestos (izquierda). Para una correcta visualización de la figura se ha optado por mostrar únicamente los píxeles afectados ya que si no es complicado diferenciarlos. El área que rodea ambas imágenes corresponde al rectángulo de ajustes desarrollado.

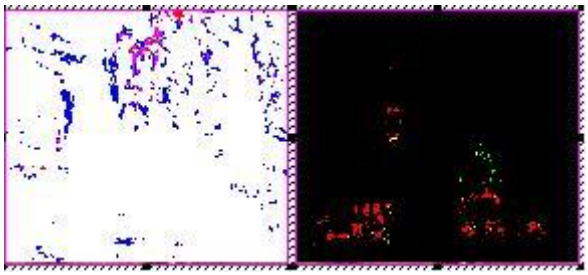

Figura 4. 1 Subexposición y Sobreexposición

En una visualización de píxeles sobreexpuestos, la función solamente actúa sobre aquellos píxeles que contienen algún color sobreexpuesto, en cuyo caso se muestra el color afectado y, en caso de estar los tres colores afectados, el píxel se muestra tal y como es, blanco.

Una vez escogido la opción de exposición del menú, la función ligada al evento del menú establece si se debe crear un objeto *VOkRectangleAjustes* o no, dependiendo de su existencia. En este momento, se ejecuta la función *CVOkImageView::SetExposicion(int s)*, cuyo parámetro *s* establece si se desea visualizar el grado de exposición del *frame* o el área sin modificación alguna.

En caso de tener un valor de s igual a 1, la parte de la imagen donde se encuentra el rectángulo de ajustes se muestra sin ninguna modificación. En cambio, si el valor es de 2 reemplaza la información seleccionada de la imagen original sobreexpuesta o subexpuesta según el caso.

Para realizar el cálculo de estas exposiciones primero, se debe calcular el tamaño en memoria de la información seleccionada ya que si la imagen es en blanco y negro tiene un tamaño menor que si la imagen es en color. Esto se deduce mediante el canal de la imagen (*cha = 1* para blanco y negro; *cha = 3* para imágenes en *RGB*). Una vez calculados estos datos, se debe obtener la información captada por el rectángulo *VOkRectangleAjustes* ya que es esta la parte donde debe realizarse las acciones a tratar. Por tanto, existe la obligación de establecer coordenadas auxiliares para saber en qué momento se deben realizar las modificaciones.

Antes de empezar con la lectura de la imagen, hay que establecer el puntero al inicio de la imagen almacenada en memoria. Esto se realiza, mediante el método *CVImage::GetRaster*.

A partir de este momento, se recorre la imagen y se aplican los cambios anteriormente explicados dependiendo el modo de exposición escogido. Si la imagen es en blanco y negro, el píxel ocupa una posición de memoria. En cambio, si es en color ocupa tres posiciones, una para cada color *RGB*.

Una vez finalizada toda la operación, se refresca la vista mediante el método *Invalidate* para obtener la nueva visualización.

4.3 BALANCE DE BLANCOS

La creación de esta herramienta se basa en la clase *CVOkImBalView* que hereda de la clase de vista formulario *CFormView*.

Esta herramienta, como el resto, es activable a partir del menú *Ver* de la vista sobre la cual se trabaja. El evento del menú tiene asociada una función miembro que permite la ejecución de la clase que se está tratando en este punto. Dicha función es *CVOkImageView::OnVerBalance()* y, básicamente, se encarga de configurar parámetros de visualización y muestra el formulario a través de *CFrameWnd::InitialUpdateFrame*.

El formulario proporcionado por la clase *CVOkImBalView* cuenta con tres campos que representan el balance de blancos para los colores rojo, verde y azul. Como se ha comentado en el capítulo destinado al estudio teórico, se debe seleccionar un área de color blanco o gris neutro de la imagen para poder realizar un cálculo correcto del balance. Para lograrlo, se debe computar la media de la tonalidad para cada uno de los colores *RGB* del rectángulo seleccionado y establecer la similitud de los valores obtenidos. Partiendo de que el color verde tiene el valor 1, el rojo y el azul se obtienen de dividirlos por el valor del color verde obtenido anteriormente. Si uno de los colores predomina sobre el verde, aparecerá con un valor superior a 1. En caso de predominar el verde, aparecerá con un valor inferior a 1.

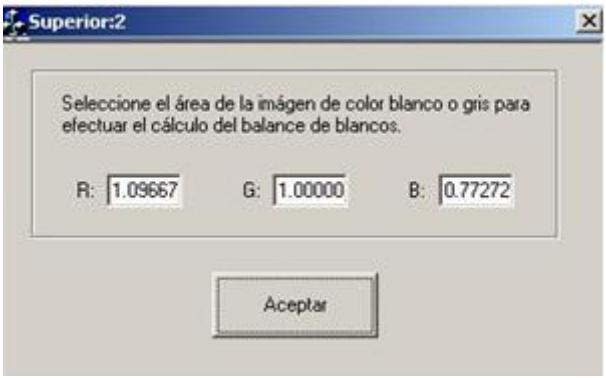

Figura 4. 2 Formulario Balance de blancos

En la **Figura 4.2** se observa un ejemplo exagerado de este concepto. En este ejemplo el color azul, correspondiente al campo *B*, está siendo dominado por los colores rojo y verde, *R* y *G* respectivamente. Por tanto, el usuario debería modificar el balance, otorgándole un valor más alto al color azul y, de este modo, equiparar los valores. Aunque el predominio entre el color rojo y el verde es mínimo, también sería aconsejable ajustar un poco el color rojo.

Antes de empezar a desgranar los métodos de la clase, *CVOkImBalView* necesita objetos de la clase *CVOkImageView*, para obtener el objeto *VOkRectangleView* que permite obtener las coordenadas de la porción de imagen a tratar, y otro de la clase *CVImage* para contener dicha porción. Realmente, estos objetos son prescindibles pero si se definen no se pierde tiempo en accesos a datos de otras clases y, por tanto,

se obtiene un rendimiento más óptimo. Como se verá más adelante el resto de módulos también incorpora estos elementos.

Seguidamente, se procede a explicar los métodos más relevantes que contiene la clase:

- **OnBnClickedButton1**
  Este método se ejecuta al producirse el evento de clickado sobre el elemento *Button1* y se utiliza para cerrar el formulario. Esto se logra, mediante el método *GetParentFrame()->DestroyWindow()*.

- **OnInitialUpdate**
  Este método es llamado al inicio de la ejecución de la clase. Simplemente, se encarga de obtener la clase vista desde donde se muestra la imagen al usuario y, posteriormente, se encarga de hacer una llamada a *SetBalance*, función destinada a realizar el cómputo del módulo.

- **OnUpdate**
  Es ejecutado al producirse cambios en los datos del documento, concretamente al hacer una llamada a *CDocument::UpdateAllViews*, y realiza una llamada a la función *SetBalance*.

- **SetBalance**
  Como se ha comentado en las líneas anteriores, este procedimiento se encarga de calcular el balance de blancos.
  Primero, se debe obtener la imagen con la que trabajar, concretamente usando el método *CVImage::Cut(int,int,int,int)* al cual le pasamos las coordenadas del rectángulo de ajustes para realizar el corte que interesa.
  Una vez obtenida la imagen, se debe calcular el nuevo tamaño de esta y verificar que se trata de una imagen a color, ya que en blanco y negro no tiene sentido el uso de esta herramienta. A partir de este momento, se van acumulando los valores correspondientes a la tonalidad de cada color para todos los píxeles y, posteriormente, se realiza la media de estos.
  Para finalizar el proceso, se dividen los valores de color rojo y azul entre el valor del color verde. De esta manera obtenemos la proporción de colores respecto al verde. Por último se muestran los valores resultantes en los *CEdit* que contiene el formulario mediante el método *CWnd::SetWindowTextA(LPCTSTR)*.

4.4 HISTOGRAMA

Esta funcionalidad está creada mediante una nueva clase llamada *CVOkImHistoView* que hereda de la clase *CFormView*. Esta clase permite crear un formulario como clase vista de un documento.

El módulo es activable a través del menú *Ver* de la vista donde se está trabajando. Este evento tiene una función asociada llamada *CVOkImageView::OnVerHistograma*, la cual establece el tamaño y algunas características del formulario a mostrar, como por ejemplo, la eliminación de los botones minimizar y maximizar. Mediante el método *CFrameWnd::InitialUpdateFrame* se abre el formulario destinado a contener el histograma.

El formulario cuenta con la posibilidad de escoger, mediante un *ComboBox*, la información en blanco/negro, rojo, verde, azul o en RGB. También se muestra información numérica como puede ser el valor total de píxeles de la imagen o, si el cursor se encuentra encima de una columna, se indica la tonalidad de esta y el número de píxeles que tienen esa tonalidad. A continuación, en la **Figura 4.3** se muestra el histograma desarrollado:

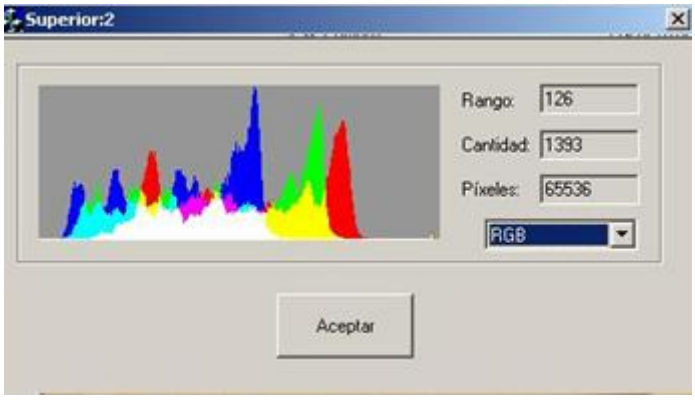

Figura 4. 3 Formulario Histograma

A parte de los métodos explicados posteriormente, la clase *CVOkImHistoView* contiene los siguientes elementos necesarios para su funcionamiento: La estructura color contiene toda la información acerca de los píxeles de la imagen. *nColor* es la estructura

que almacena los valores aportados por color escalados para poder ser imprimidos en el formulario. En la siguiente figura se puede observar dichas estructuras:

```
struct{                          struct{
       int m_pRed[256];                 long nRed[256];
       int m_pGreen[256];               long nGreen[256];
       int m_pBlue[256];                long nBlue[256];
       int m_pRGB[256];          } nColor;
 } color;
```

Figura 4. 4 Estructura *color* y *nColor*

Long nPaint[256] es el *array* que contiene la información que se imprimirá en el formulario, obtenida a través de la estructura *nColor*. *CVImage* m_pCVImB* es la nueva imagen obtenida de recortar la imagen del documento a través de las coordenadas de *VOkRectangleView*. Estos han sido los elementos más importantes que contiene la clase, ahora veamos como se ha desarrollado.

El constructor de la clase contiene un objeto de tipo *CVImage* destinado a contener la parte de la imagen seleccionada por el rectángulo de ajustes. También incorpora un objeto *CVOkImageView* para relacionar la clase actual con el objeto *VOkRectangleAjustes* que la compone. Así se obtienen las coordenadas del rectángulo de ajustes. Por último, está formado por una estructura con tres arrays de 256 posiciones inicializadas a 0. Esta estructura contendrá la información recopilada sobre el histograma de la imagen.

A continuación se muestran los métodos más relevantes de esta clase:

- **OnBNClickedButton5()**
  Esta función está asociada al elemento de formulario *Button5* y se ejecuta cuando el usuario hace *click* sobre este. La función hace una llamada al método *GetParentFrame()->DestroyWindow()* para cerrar el formulario actual.

- **OnCBNSelchangeCombo2()**
  Este método está asociado al objeto *CComboBox* llamado *Combo2* que aparece en el formulario. Se ejecuta al cambiar el elemento seleccionado de la lista de opciones que contiene. Según el elemento seleccionado, se hace un volcado de

datos sobre el *array nPaint* procedente de uno de los *arrays* de la estructura *color* según el color seleccionado, exceptuando en la opción RGB que se establece de otro modo.
Posteriormente, se llama a la función miembro *CVOkImHistoView::PaintHistograma* para dibujar el histograma en el formulario.

- **OnDraw(CDC *pDC)**
  Este método únicamente llama a la función miembro *CVOkImHistoView::PaintHistograma* encargada de imprimir el histograma en el formulario.

- **OnInitialUpdate()**
  Esta función miembro se ejecuta al iniciarse la clase a la cual pertenece. En este proyecto, primero se hace un ajuste del tamaño de la ventana de formulario, se obtiene la primera vista asociada al documento, normalmente *CVOkImageView* y, seguidamente se procede a cargar el objeto *CComboBox* que hay en el formulario, estableciendo como elemento seleccionado el valor asignado al color rojo.
  También se llama al método *CVOkImHistoView::SetHistograma* encargado de recopilar toda la información sobre la parte de la imagen seleccionada por el recuadro de ajuste y se hace un volcado sobre *nPaint* para, posteriormente, hacer una llamada a la función *CVOkImHistoView::PaintHistograma* y dibuja en el formulario la información sobre los píxeles que contiene *nPaint*.

- **OnMouseMove(UINT nFlags, CPoint point)**
  Este método se encarga de hacer una llamada la función miembro *CVOkImHistoView::WritePointData* encargada de actuar si el puntero se encuentra posicionado encima del histograma.

- **OnUpdate(CView* pSender, LPARAM lHint, CObject* pHint)**
  Como se ha explicado en el apartado 3 de la presente memoria, este método se ejecuta cuando se produce un cambio en la vista de documento actual. Por tanto, si se produce un cambio en la información que se obtiene procedente del documento se deberá hacer un recálculo de la información a presentar en el histograma y volver a imprimir dicho contenido en el formulario. El procedimiento seguido por *CVOkImHistoView::OnUpdate* es el mismo que en *CVOkImHistoView::OnInitialUpdate* visto anteriormente.

- **SetHistograma()**
  Este procedimiento se encarga de leer la información procedente de la imagen que contiene *CVOkImageDoc*, seleccionando únicamente aquella información de la imagen que coincide con las coordenadas del objeto *VOkRectangeAjustes* que contiene la clase *CVOkImageView*.
  Una vez obtenida la nueva imagen, almacenada en *CVImage m_pCVImB*, se procede a calcular su tamaño y, en función de si es en blanco y negro o en color, se procede a leer la información de los píxeles almacenándolos en la estructura *color*. Una vez ordenada la información, se establece el valor máximo que puede alcanzar una tonalidad para cada uno de los colores y se hace un escalado para su posterior presentación en el histograma.

- **PaintHistograma()**
  Este procedimiento se encarga de imprimir el histograma en el espacio del formulario habilitado para tal uso. Primero, se dibuja el rectángulo sobre el cual se imprimirá la información a mostrar, se crea un objeto de tipo *CBrush* y otro *CRect* con las coordenadas del rectángulo a dibujar y, posteriormente, se llama al método *CClientDC.FillRect(CRect,CBrush)* para su impresión. Por último, se va printando toda la información según la opción seleccionada en el *CComboBox* mediante el método *CClientDC.SetPixel(int,int,COLORREF).*

- **WritePointData(CPoint point)**
  Este método se encarga de aportar información más detallada al usuario sobre el histograma. El usuario puede pasar el cursor por encima del histograma y este le retorna información acerca de la tonalidad en la que está situado y el número de píxeles con dicha tonalidad que contiene la imagen. Según el tipo de histograma mostrado este método accede a la estructura color y retorna la cantidad de píxeles que tiene la tonalidad seleccionada para dicho color. En caso de mostrarse el histograma RGB, la función retorna la suma de las tres componentes RGB. La información se imprime en los objetos *CEdit* del formulario mediante el método *CWnd::SetWindowTextA(LPCTSTR)*. Si el cursor no está situado en el área que comprende el histograma, no se muestra ningún tipo de información.

4.5 ENFOQUE

El módulo de enfoque está creado de nuevo sobre una clase (*CVOkImFocusView*) que hereda de *CFormView,* usada en los dos módulos anteriores.

La herramienta vuelve a ser activable a través del menú *Ver* de la vista documento *CVOkImageView* donde volvemos a tener un evento ligado al método encargado de llamar a esta clase *CVOkImageView::OnVerEnfoque*. Este método tiene la misma funcionalidad que los anteriores, establece una serie de parámetros para conseguir una interfaz más amigable.

El formulario creado es bastante sencillo y práctico ya que cuenta con una barra en la cual se puede observar el grado de enfoque de la imagen a procesar. El máximo enfoque logrado se representa mediante una fina línea vertical de color verde que se sitúa en cierto punto de la barra de lectura. El valor de enfoque actual se muestra mediante una barra azul horizontal que rellena la barra de lectura. Esta forma de representación es muy buena ya que es fácilmente interpretable por el usuario. También se incluye el valor real calculado por el algoritmo *Tenengrad* para obtener un grado mayor de precisión.

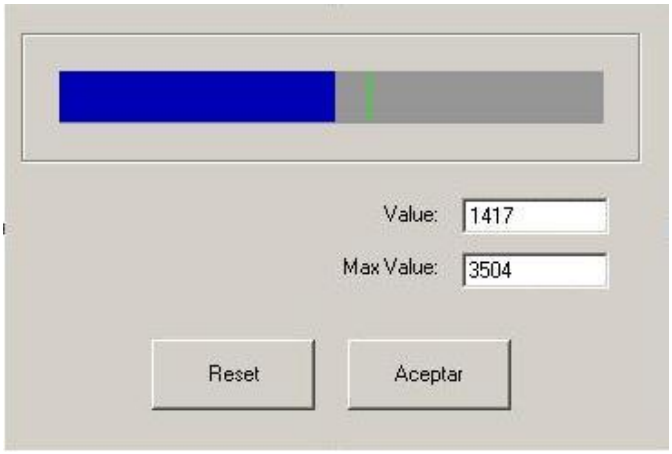

Figura 4. 5 Enfoque

También se ha añadido un botón que establece un *reset* del cálculo del mejor enfoque para que el usuario pueda volver a realizar el ajuste.

La clase contiene el entero *m_pMax* encargado de almacenar el valor máximo calculado por el algoritmo desarrollado y que, inicialmente, tiene un valor de 0. También se vuelve a incluir un objeto *CVImage* m_pCVImF* encargado de almacenar la imagen a procesar y otro *CVImageView* m_pView* para obtener la vista sobre la cual actúa el objeto *VOkRectangleView* el cual proporciona las coordenadas de la imagen que hay procesar.

Los métodos responsables del funcionamiento de este módulo son los siguientes:

- **OnDraw(CDC *pDC)**
  Este método únicamente llama a la función miembro *CVOkImFocusView::PaintBar* encargada de imprimir el resultado obtenido del cálculo del enfoque en el formulario.

- **OnInitialUpdate()**
  Esta función miembro funciona prácticamente igual que en los módulos anteriores. Se ejecuta al iniciarse la clase a la cual pertenece, se ajusta el tamaño de la ventana de formulario y se obtiene la primera vista asociada al documento, normalmente *CVOkImageView*. También se establece *m_pMax* a 0 y se procede a llamar al método *PaintBar*.

- **OnUpdate(CView* pSender, LPARAM lHint, CObject* pHint)**
  Al producirse un cambio en la información de la imagen que contiene la clase documento *CVOkImageDoc*, se procede a llamar al método *PaintBar*.

- **PaintBar()**
  Este procedimiento se encarga de calcular el valor de *Tenengrad* mediante una llamada a la función *GetCutImage* y, una vez obtenido dicho valor, se dibuja un rectángulo horizontal que contiene los resultados obtenidos, tanto el actual como el máximo. Este rectángulo se dibuja en color gris mediante las clases *CBrush* y *CRect*, y la llamada al procedimiento *CClientDC::FillRect(CRect,CBrush)*. El resultado obtenido en el último cálculo se realiza de la misma manera que en el caso anterior pero de un color azul marino para poderlo diferenciar. Por último, re realiza la impresión del valor máximo alcanzado por el algoritmo mediante una línea vertical de color verde. Esta se consigue, mediante la llamada al método *CClientDC::MoveTo(int, int)* que nos sitúa el cursor en las coordenadas especificadas como parámetros y el método *CClientDC::LineTo(int,int)* que nos dibuja una línea hasta las coordenadas destino pasadas por parámetro.

- **GetCutImage()**
  Este método se encarga de obtener la imagen a tratar, mediante la obtención del rectángulo de ajustes que contiene la clase *CVOkImageView* asociada al documento. Dicha obtención se realiza de la misma forma que en las anteriores herramientas de configuración. También se establece si el rectángulo de ajustes ha cambiado su posición ya que si ha sido así se debe proceder a restablecer la variable *m_pMax* y volver a realizar el calibrado del enfoque. Esto es debido a que se producen diferentes resultados al calcular el enfoque de una parte de la imagen o de otra. Posteriormente, en caso de trabajar con imágenes o *frames* en color, se debe realizar una conversión de la imagen a blanco y negro. Recordar que el tipo de algoritmo utilizado en este módulo trabaja con dicho formato.
  Para finalizar, se procede realizar el cálculo del mejor enfoque de la imagen obtenida. El procedimiento para tratar la imagen es el mismo que en los casos anteriores, mediante punteros. Para cada píxel se aplica *convolución* y se va almacenando el resultado en una variable acumulativa.
  Como la imagen puede tener una cantidad muy grande de píxeles es posible que el valor de *Tenengrad* final acabe saliéndose de rango y, por tanto, retorne error. Es por esto, que valor resultante de aplicar *convolución* a un píxel, se ajusta dividiéndolo por el número total de píxeles de la imagen. De esta manera obtenemos un valor menor y no se produce este error.
  Finalmente, el valor obtenido se escala para su mejor representación en el formulario.

- **OnBNClickedButton1()**
  Esta función está asociada al elemento de formulario *Button1* y se ejecuta cuando el usuario hace *click* sobre este. La función hace una llamada al método *GetParentFrame()->DestroyWindow()* para cerrar el formulario actual.

4.6 CONFIGURACIÓN DE PARÁMETROS DE LA CÁMARA

Esta herramienta permite al usuario crear una nueva cámara y configurarla o configurar una cámara existente previamente. Antes de explicar cómo se realiza dicha configuración se expone la implementación de una nueva cámara auto configurable ya que estas cámaras se crean de manera distinta a la ya existente previamente.

Para ello, es necesario crear un nuevo método de la clase VOkCamera que permite saltarse la comprobación referente a si el proyecto está en línea o no, ya que estas cámaras se crean en modo *Proyecto en línea*. El método llamado VOkCamera::NewCameraAuto cumple con esta condición y hace una llamada a VOkDevice::InserCamera destinado a crear la cámara definida por el usuario.

Una vez aclarado este punto, se hace necesaria la creación de una clase llamada *CNewCameraAuto:CDialog* destinada a permitir la creación de una nueva cámara para el dispositivo o la configuración de una existente. El diálogo que comprende a esta clase debe ser activable mediante el menú que incorpora el dispositivo. Esto se logra mediante el mapa de eventos siguiente:

```
ON_COMMAND(ID_DISPOSITIVO_NUEVACAMARAAUTO,
OnDispositivoNuevacamaraAutoConf)
```

Figura 4. 6 Mapa de eventos

El método asociado a este evento OnDispositivoNuevacamaraAutoConf contiene la creación de un objeto *CNewCameraAuto* y la activación de su método *DoModal*. También se pasa como parámetros el proyecto actual sobre el que se está trabajando y el correspondiente dispositivo.

La clase CNewCameraAuto es un diálogo que contiene, por un lado, la posibilidad de escoger una cámara existente mediante una lista desplegable o, por el otro, la creación de una nueva cámara a configurar introduciendo un nombre y pulsando el botón *Nuevo*. También aparecen tres *ComboBox* para efectuar la configuración de la cámara seleccionada anteriormente que corresponden a la resolución, color y *frame rate*. Una vez efectuada la configuración se debe pulsar el botón *Aceptar*.

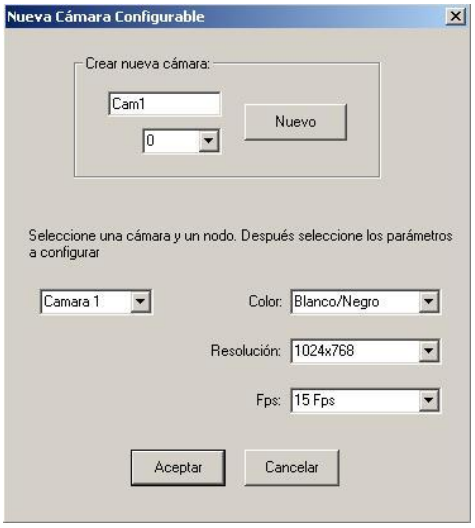

Figura 4. 7 Diálogo nueva cámara configurable

A continuación, se procede a desengranar los métodos utilizados en la creación de este diálogo:

- **OnInitDialog**
  Este método se encarga de leer del dispositivo todas las cámaras y los respectivos nodos a los que están conectadas y los almacena en un *ComboBox*. De esta manera el usuario puede seleccionar una de ellas en caso de quererlas configurar.
- **OnBnClickedButton1**
  Comprueba que el nombre introducido para la nueva cámara no es un campo vacío ni que está duplicado en la lista de cámaras del dispositivo. Una vez comprobado se procede a la creación de la nueva cámara en un nuevo libre. Se debe refrescar el *ComboBox* con la nueva cámara disponible.

- **OnCbnSelchangeCombo4**
  Al producirse un cambio en la selección de una cámara, se ejecuta este método el cual se encarga de establecer una configuración predeterminada a la cámara para, de esta manera, poder empezar el proceso de configuración por parte del usuario haciendo una llamada a *GetFormats*.

- **GetFormats**

En este procedimiento se establece los valores de *format* y *mode* que posteriormente recibirá la cámara. Primero, se deben establecer que combinaciones de *format* y *mode* acepta la cámara *firewire* usada. Para ello se hace uso de la función *FirewireCamera::GetVideoMode(format, mode)*. Si la configuración se acepta, se muestran en los *ComboBox* correspondientes la resolución y el tipo de color asignado a esta configuración. Además si se encuentra una configuración para una cámara a color se indica en el *ComboBox* destinado a almacenar si una cámara es en blanco y negro o no. Acto seguido se llama a *CNewCameraAuto::GetRates* para establecer el *frame rate* de la cámara.

- **GetRates(int format, int mode)**
  Este método se encarga de almacenar en un ComboBox los diferentes frame rates soportados por la cámara para una configuración determinada. Para ello se utiliza el método *FirewireCamera::GetVideoRate(format, mode, rate)*.

- **OnCbnSelchangeCombo1**
  Este método se activa cuando se produce un cambio en la selección de un elemento de la lista desplegable de resoluciones. En este caso, se comprueba esta escogida la opción de blanco y negro o color y, según sea, se establece una combinación de *format* y *mode* a la resolución seleccionada.

- **OnCbnSelchangeCombo3**
  En caso de seleccionarse un nuevo *frame rate* se guarda dicho valor seleccionado por el usuario.

- **OnBnClickedOk**
  Este método se ejecuta cuando el usuario pulsa el botón *Aceptar*. En este caso se procede a mandar los parámetros de configuración leídos de los ComboBox para la cámara seleccionada. Para ello se hace un *FirewireCamera::SetVideoValues(Camara, format, mode, rate)*.

Como se ha comentado anteriormente, una vez pulsado el botón Aceptar se hace una llamada a *FirewireCamera::SetVideoValues*. Este método hace una llamada a *FirewireCameraImp::SetVideoValues*, encargado de hacer una parada en la captura de imágenes y de establecer los nuevos parámetros para la cámara *firewire* con la que se está trabajando. Estos valores se envían a la cámara tal y como se ha explicado en el capítulo tres referente a los *CMU Drivers*. Una vez establecidos los valores, se procede a restablecer la captura de *frames* de la cámara mediante funciones proporcionadas por dicha librería.

Para finalizar este punto del desarrollo, se ha creado la clase *FirewireCameraAutoConfDlg:CDialog* similar a la ya mencionada *CNewCameraAuto:CDialog* que permite a un usuario modificar una cámara seleccionándola directamente a través de la vista árbol que proporciona el proyecto en el que se está trabajando. En la **Figura 4.8** se puede observar el diálogo creado, con los tres campos configurables.

Figura 4. 8 Diálogo de la configuración de cámaras

El funcionamiento de este diálogo es prácticamente idéntico al anterior si bien en este caso la llamada a dicho objeto se realiza mediante la creación de una opción de menú al hacer *click* con el botón derecho sobre la cámara seleccionada. En la **Figura 4.9** se muestra el código necesario para realizarlo:

```
CreateMenuOption("AutomaticConfiguration",
(MenuCommand)
&FirewireCamera::CommandOpenAutoConfDlg);
```

Figura 4. 9 Opción de Menu

El método asociado al menú crea el objeto de la clase *FirewireCameraAutoConfDlg:CDialog* y abre el diálogo el modo modal.

Capítulo 5

PRUEBAS

En este capítulo se detallan las pruebas realizadas sobre el software desarrollado. Se han tenido en cuenta pruebas que comprueban la robustez del software en todos los casos que se pueden dar cuando un usuario ejecute los módulos. También se ha tenido en cuenta el comportamiento que puede tener la ejecución de los módulos de ajuste sobre un tipo de hardware, sobretodo en el modo de captura de vídeo ya que es en este punto donde se requieren más recursos.

5.1 TEST FUNCIONAL

La comprobación del buen funcionamiento del software implementado se ha basado en pruebas de *caja blanca*. Este tipo de pruebas consisten en analizar el sistema mediante el recorrido de todos los caminos de ejecución posibles, haciéndolos pasar por todas las condiciones *if/else*, bucles con 0 o 1 iteraciones, etc. Este análisis se ha realizado tanto a nivel funcional como sobre el conjunto de varias de estas funcionalidades. Con ello, se ha podido eliminar algún que otro *bug* que aparecía en ciertas partes de la ejecución del programa, por ejemplo, intentar ejecutar los módulos de ajuste cuando no hay ningún proyecto sobre el que trabajar o hacer lo mismo cuando el documento no contiene ninguna imagen.

También se han realizado pruebas sobre la información obtenida al ejecutar alguno de los módulos.

En el caso de la exposición, se ha revisado que la detección de los puntos de sobreexposición y subexposición funciona correctamente. Esto se ha realizado mediante pequeñas muestras elegidas sobre las cuales se conoce de antemano que existen puntos con estos defectos. La comprobación se ha realizado mediante un aumento del zoom de la imagen y posicionando el cursor tanto en píxeles defectuosos como en píxeles correctos.

Las pruebas relacionadas con los datos mostrados en el histograma se han realizado mediante la creación de imágenes con escala de tonalidades. Primero, con las tres

componentes RGB y, posteriormente, con combinaciones de estas. Evidentemente, se han buscado obtener unos resultados los cuales sea fácilmente reconocible si son correctos o incorrectos.

El test realizado sobre el módulo del balance de blancos ha consistido en presentar imágenes de distintas variantes de dominantes. Por ejemplo, una imagen amarilla presenta unos dominantes rojo y verde en formato RGB. También se ha realizado captura de una escena blanca y, cambiando el balance de blancos que incorpora la cámara, se han obtenido unos resultados satisfactorios.

Las pruebas relacionadas con el enfoque pretenden demostrar el buen funcionamiento del algoritmo de *Tenengrad*.

Primero, se ha pretendido demostrar que este algoritmo, como el resto de algoritmos basados en la detección de bordes, no tiene un buen funcionamiento con superficies lisas o prácticamente lisas. Para este tipo de superficie ha presentado una gran tasa de fallos respecto a superficies en las cuales sus contornos están bien definidos.

También se ha utilizado el algoritmo con los operadores de *Prewitt* y *Sobel*. El segundo ha demostrado un mejor funcionamiento gracias al énfasis que se hace sobre las coordenadas vertical y horizontal otorgándoles el doble de peso, especialmente en imágenes que presentaban cierto ruido. Esto ha sido un buen indicador ya que las imágenes adquiridas por la cámara presentan ruido.

El ruido ha sido un punto a tener en cuenta, ya que en un primer momento se estableció un cierto umbral, cercano a cero, para poder aplacar resultados erróneos procedentes de esta variable. Los resultados obtenidos no han sido muy satisfactorios y, antes de suprimir el umbral, se tomó la decisión de variar el algoritmo y usar *CGA* (*Conteo de Gradiente Absoluto*). Este algoritmo presentó peores resultados que *Tenengrad* usando umbral. Por último se decidió suprimir el umbral y, junto con el operador de *Sobel*, se comprobó que *Tenengrad* funcionaba muy satisfactoriamente.

Hay que tener en cuenta que en los test realizados en este módulo requieren que factores como la iluminación o el balance de blancos se mantengan constantes ya que una variación en estos provoca que los resultados del algoritmo sean incorrectos.

Las pruebas realizadas sobre la herramienta de configuración de cámaras han sido satisfactorias ya que se puede cambiar dicha configuración y es posible corroborar los cambios al visualizar la captura de *frames* realizada por la cámara.

Es necesario comentar que estas pruebas se han ido realizando a lo largo del desarrollo del proyecto, especialmente al terminar alguno de los módulos de ajuste o configuración comentados anteriormente. De esta manera, se puede asegurar el buen funcionamiento de la herramienta a lo largo desarrollo producido posteriormente y que no influye directamente sobre esta. También se han realizado sobre todos los módulos en funcionamiento y se ha comprobado el rendimiento que ofrecen tal y como se comenta en el siguiente apartado.

5.2 TEST DE RENDIMIENTO

Este tipo de pruebas se han realizado para evaluar la calidad del software resultante de la realización del proyecto. Además, han servido para realizar ajustes o cambios en la implementación que permitan mejorar el rendimiento y se agilice la velocidad de procesado.

Esta última cuestión ha sido de vital importancia ya que en este proyecto se trabaja con cámaras *firewire* que pueden llegar a capturar *240 fps* (frames por segundo). Este requerimiento, ha provocado el uso de punteros. Se ha de entender que los módulos se aplican sobre imágenes que pueden tener millones de píxeles y, por tanto, es necesario el uso de punteros para recorrerlas.

También se ha intentado, en la medida de lo posible, usar el menor tipo de objetos especialmente robustos durante la ejecución de los módulos. Por ejemplo, se tuvo que minimizar el uso de la clase *CVImageView* porque provocaba un cuello de botella durante la ejecución del módulo de enfoque con imágenes a color, ya que este tipo de imágenes contiene mucha más información que imágenes en blanco y negro y, además se debe hacer un cambio de formato de *RGB* a *Mono* (Blanco y negro). Esto provoca que se consuman bastantes recursos para imágenes grandes.

Otro tipo de optimización usada en el módulo de enfoque ha sido la intención de evitar realizar excesivos cálculos mediante variables de tipo *float*. Por tanto, la convolución y cálculo del módulo del gradiente de la imagen a tratar se ha realizado mediante variables de tipo *integer*. De esta manera se optimiza la velocidad de cálculo aún utilizando imágenes de gran tamaño.

Tal y como se ha comentado en el apartado de desarrollo del módulo de enfoque, al realizar el testeo de este mediante imágenes de diferentes tamaños, se comprobó que

el valor de *Tenengrad* era demasiado grande y se salía de rango. Es por esto, que se realizó la división del valor calculado para un cierto número por la cantidad total de píxeles de la imagen. Así cuando mayor sea la imagen mayor será el denominador de la división, con lo que prácticamente se elimina esta posibilidad.

Por último, se han realizado pruebas sobre todos los módulos ejecutándose a la vez y el rendimiento no se ha visto afectado en ningún momento para selecciones de imagen que no sobrepasen 512 x 512 píxeles. Con imágenes de gran tamaño si se aprecian pequeñas ralentizaciones en el momento de mover el área de ajuste mediante las teclas asignadas para tal uso, causadas por la gran cantidad de cálculos a realizar.

Las optimizaciones anteriormente descritas han sido de gran ayuda en el uso de computadores de generación anticuada como son los *Pentium IV*. Para ordenadores actuales como pueden ser *Dual* o *Quad Core* no se presentan cambios muy evidentes, pero sí en el caso de seleccionar imágenes grandes ya que no se han producido problemas de rendimiento.

Como conclusiones se puede afirmar que el software desarrollado es soportable como mínimo a partir de ordenadores tipo *Pentium IV* o *AMD Athlon 2400+* con *1 GigaByte* de memoria *RAM*. Actualmente, ya no se fabrican computadores de este estilo y, por tanto, se puede dar como satisfactoria la prueba de rendimiento.

Capítulo 6

CONCLUSIONES

En este capítulo se muestran las conclusiones extraídas en la realización de este proyecto y también se hace un breve comentario a las líneas futuras que se pueden desarrollar a partir de este proyecto.

6.1 CONCLUSIONES

Las conclusiones extraídas tras la realización de este proyecto son satisfactorias. Aunque la planificación final del proyecto no ha sido la prevista, se han conseguido los objetivos marcados inicialmente y se han añadido otros como, por ejemplo, la realización de un área de ajuste con la que el usuario pueda interactuar y, también, una herramienta que facilita la configuración de parámetros de la cámara. Este punto es importante porque hasta ahora no era trivial dicha configuración. Ahora, sí se puede afirmar que un usuario será capaz de poner a punto una cámara *firewire*.

Gracias a la elaboración de este proyecto se ha aprendido a configurar manualmente una cámara de vídeo mediante herramientas destinadas a tal uso como es el histograma. También se ha profundizado en el campo de la fotografía digital asimilando conceptos como son el grado de *exposición* de una imagen, la *temperatura de calor* o la importancia que tiene una correcta configuración del *balance de blancos* para obtener una imagen que detalle con precisión la realidad de una escena.

También se han perfeccionado conceptos relacionados con el procesamiento digital de imágenes, concretamente con algoritmos de detección de bordes llegando a la conclusión de que tienen un funcionamiento muy correcto para la detección del grado de enfoque de una imagen.

En cuanto a la implementación ha sido de vital importancia realizar un análisis exhaustivo del software sobre el que se ha trabajado ya que presenta un nivel alto de complejidad para quien no lo ha desarrollado. También ha resultado productivo el aprendizaje realizado sobre el lenguaje *C++*, más concretamente sobre la programación orientada a objetos y el uso de *Microsoft Foundation Classes*.

Además de conceptos teóricos o prácticos, este proyecto ha ayudado a realizar un aprendizaje personal en cuanto a la habilidad en la búsqueda de información, resolución de problemas, etc.

6.2 LÍNEAS FUTURAS

La realización de este proyecto pretende facilitar el uso y configuración de cámaras *Firewire* a un técnico encargado de estas funciones. Dada la naturaleza de los módulos desarrollados en este proyecto se puede llevar a cabo un paso más en el afán de mejorar el sistema sobre el que actúan dichos módulos. Este paso no es otro que tratar la información obtenida por el software y enviar como respuesta, nuevos parámetros que nos permitan ajustar la cámara automáticamente sin la necesidad de ser necesaria la presencia de dicho técnico.

Entre las automatizaciones comentadas, con este software desarrollado se pueden implementar fácilmente la automatización de los módulos de exposición y de balance de blancos de la imagen. El primero, se puede calibrar mediante el tiempo de *exposición* de la cámara y, el segundo, gracias al cálculo de las dominantes *RGB* que realiza el módulo de *balance de blancos*.

En un primer momento, se pensó en realizar el proyecto independientemente de los drivers utilizados, pero se propuso hacer un módulo de configuración de cámaras para obtener un proyecto más completo.

Esto se debe tener en cuenta en futuras implementaciones ya que VisionOkII trabaja con diferentes drivers y, dicho módulo comprende única y exclusivamente los CMU Drivers.

Capítulo 7

ANEXO 1: MANUAL DE USUARIO

Para utilizar las herramientas creadas en el proyecto, primero hay que ir a *Archivo* → *Abrir* y seleccionar el proyecto o arrastrando el icono correspondiente del proyecto a la ventana de marco de la aplicación. También se puede crear un proyecto nuevo en el menú *Archivo* → *Nuevo*.

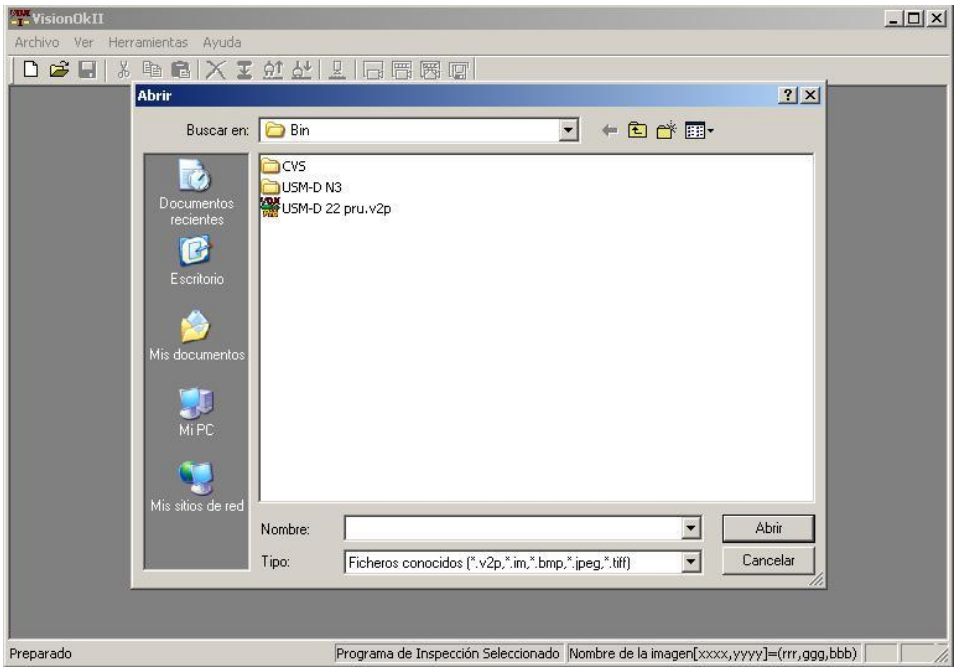

Figura 7. 1 Cargar Proyecto

Una vez cargado el proyecto, en la se puede visualizar la vista de proyecto encargada de mostrar los elementos que contiene este y sus respectivos parámetros. En la **Figura 7.2** se puede apreciar la vista de árbol del proyecto que contiene dispositivos, con cámaras asociadas (en color rojo), vistas con las que se trabaja en este proyecto, programas y controles.

En la parte derecha de la vista de proyecto, se pueden apreciar los parámetros de la cámara seleccionada por el usuario, siendo modificables por éste haciendo *click* sobre el campo a editar.

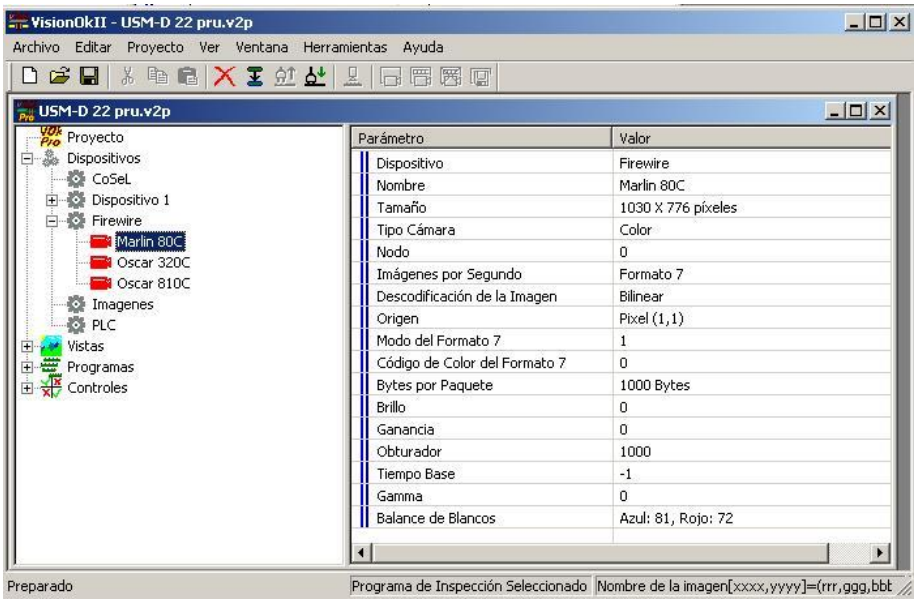

Figura 7. 2 Elección de la cámara

En este proyecto, existe la posibilidad de crear una cámara auto configurable. Para crear una cámara mediante este método, primero debemos establecer el proyecto en línea tal y como se muestra a continuación.

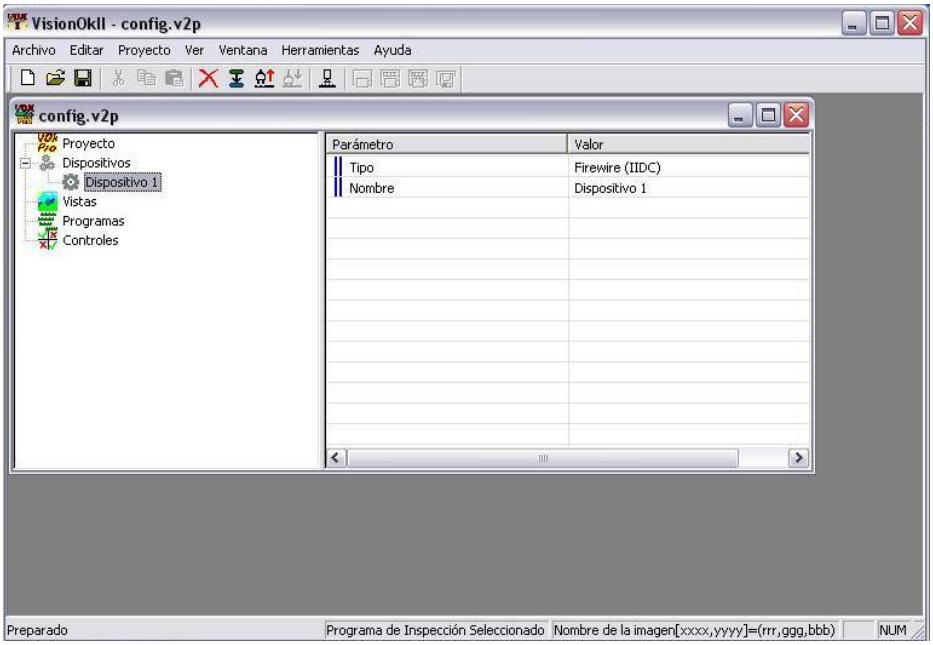

Figura 7. 3 Proyecto en línea

La creación de la cámara se efectúa mediante la selección del dispositivo y pulsando con el botón derecho del ratón. Seguidamente, aparece un menú desplegable con diferentes acciones a realizar. Seleccionar la opción de *Nueva cámara Autoconfig*.

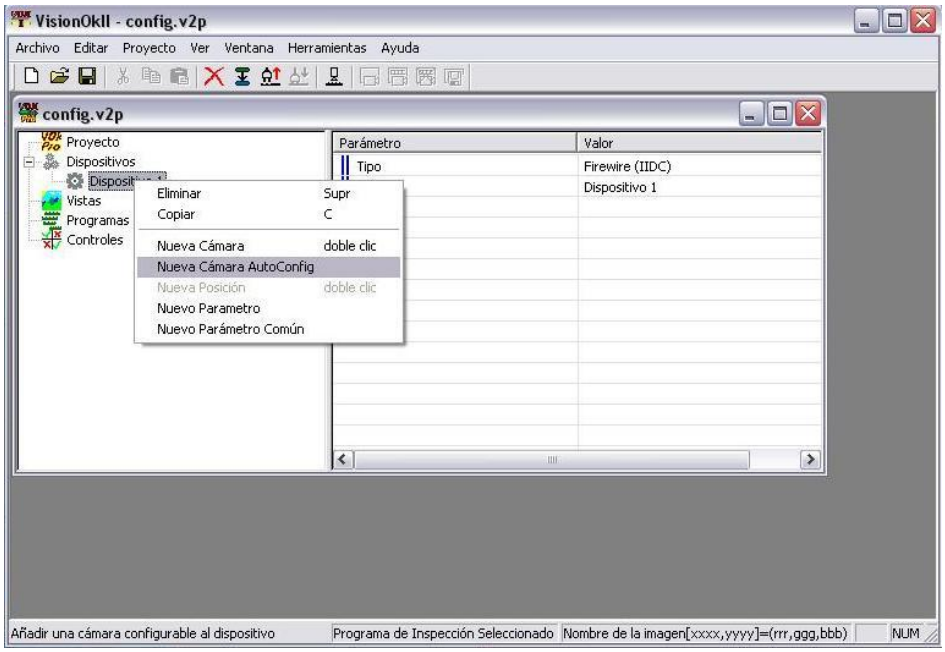

Figura 7. 4 Selección Nueva cámara auto configurable

A continuación, aparece el siguiente diálogo de nueva cámara. Se debe seleccionar un nombre para la cámara y pulsar el botón *Aceptar*.

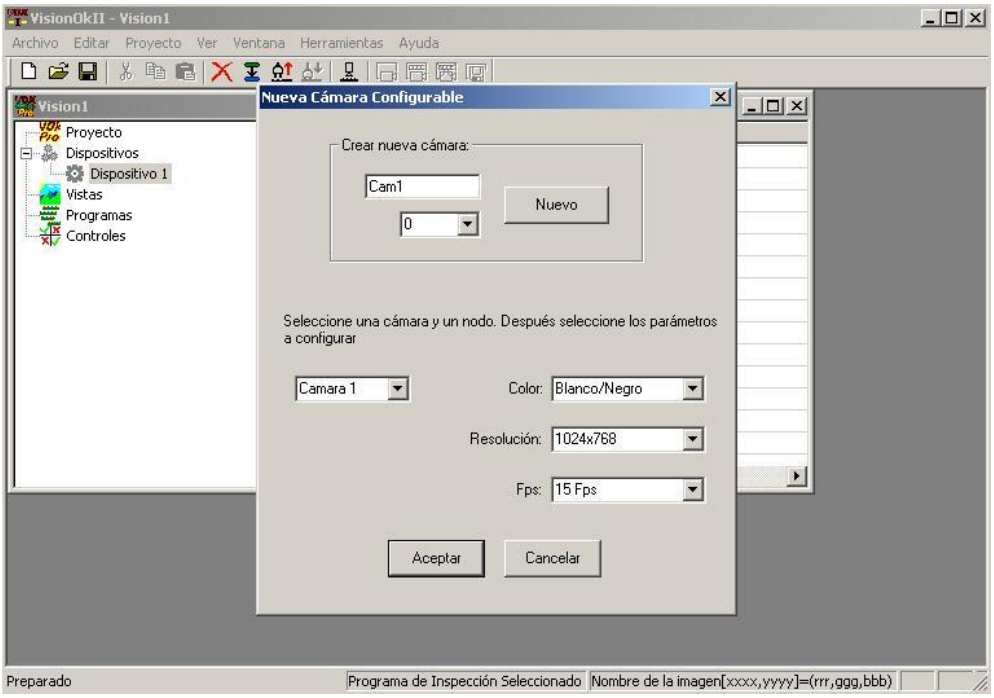

Figura 7. 5 Menú nueva cámara auto configurable

Una vez creada la cámara, se debe realizar la configuración de la cámara. Para ello, se selecciona la cámara y se pulsa con el botón derecho. Acto seguido, se debe escoger la opción *Automatic Configuration*. En la siguiente captura se muestra el paso realizado.

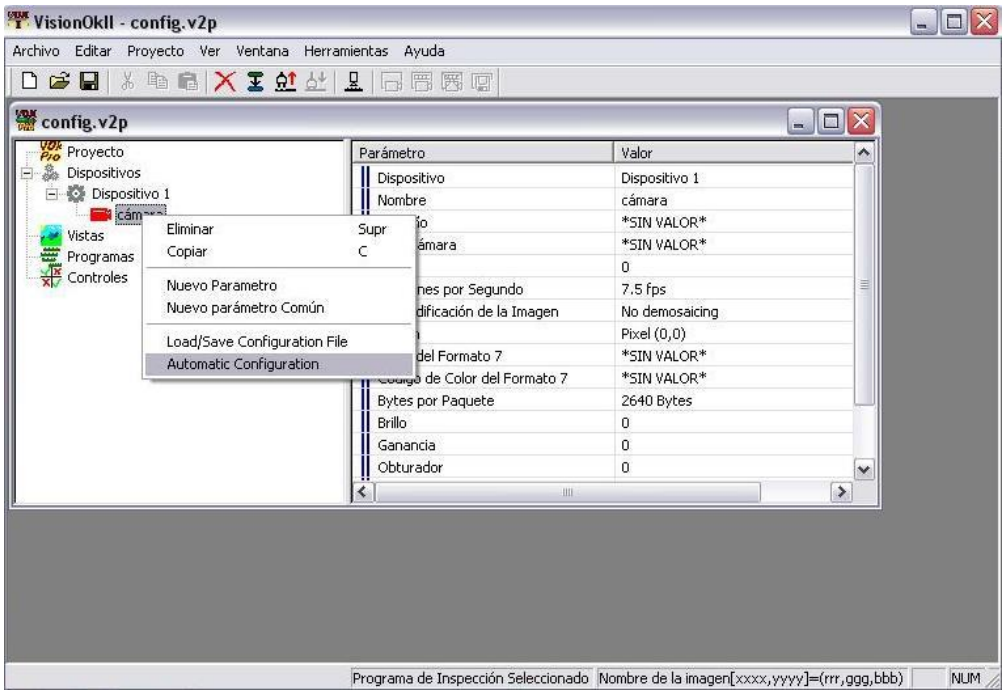

Figura 7. 6 Configuración automática

Una vez pulsada la opción, se carga un formulario con tres campos que el usuario puede configurar: color, resolución y *frames* por segundo. Para establecer la configuración se debe hacer *clicar* el botón *Aceptar*.

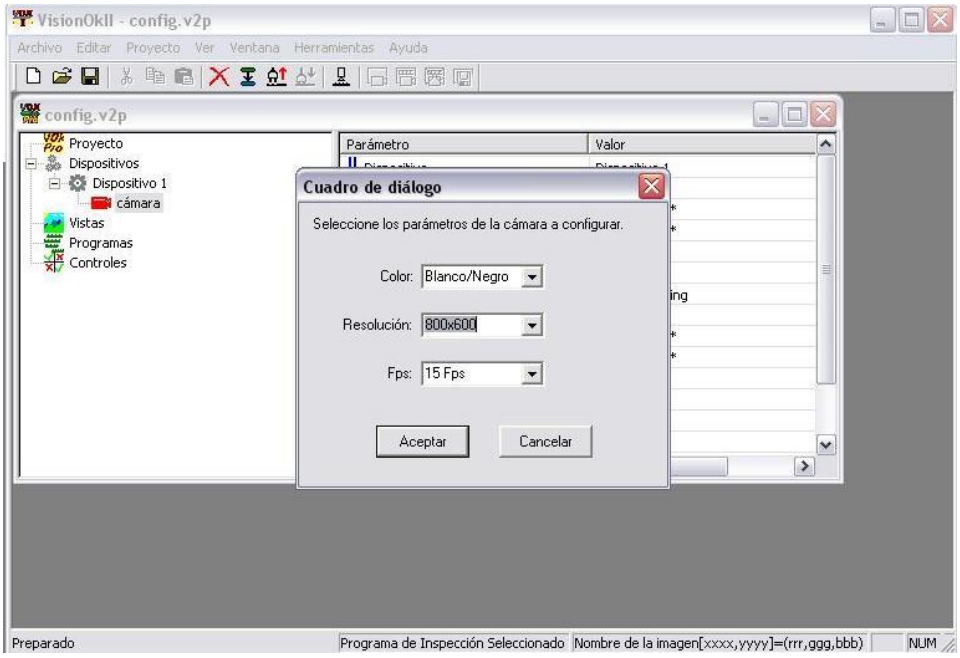

Figura 7. 7 Diálogo de configuración de cámara

Una vez vistos los pasos para crear una nueva cámara y ponerla en funcionamiento, se procede a crear una nueva vista para la cámara. Para ello, hay que pulsar con el botón derecho del ratón y seleccionar *Crear nueva vista*. A continuación, aparece un nuevo diálogo como el de la **Figura 7.8**, y se debe seleccionar la cámara que verá asociada esta nueva vista.

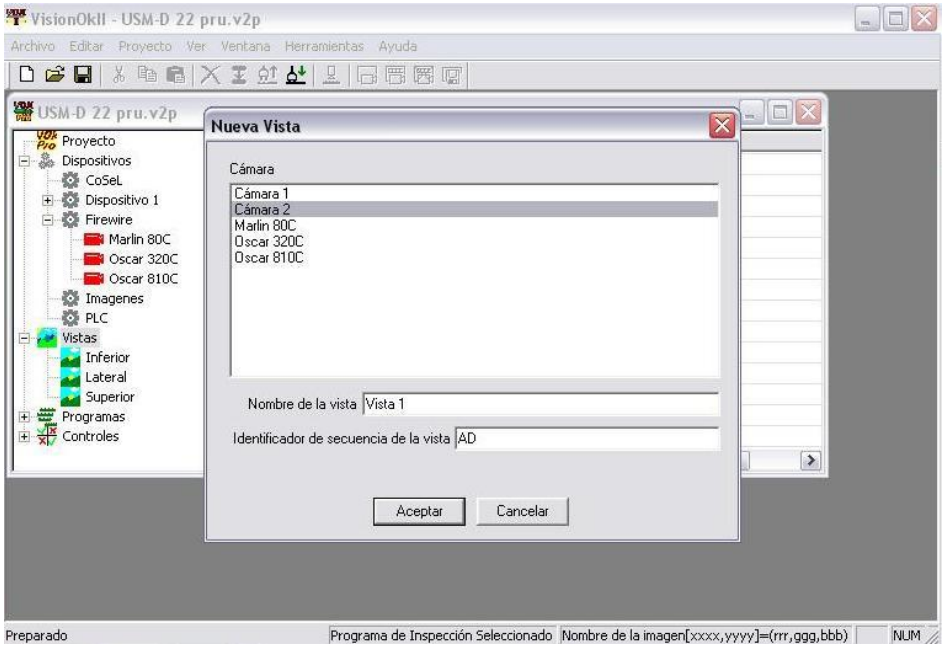

Figura 7. 8 Nueva vista

La visualización del contenido de la cámara se realiza mediante las vistas creadas y la realización de una captura o una captura continua se realiza mediante los botones indicados en la siguiente imagen:

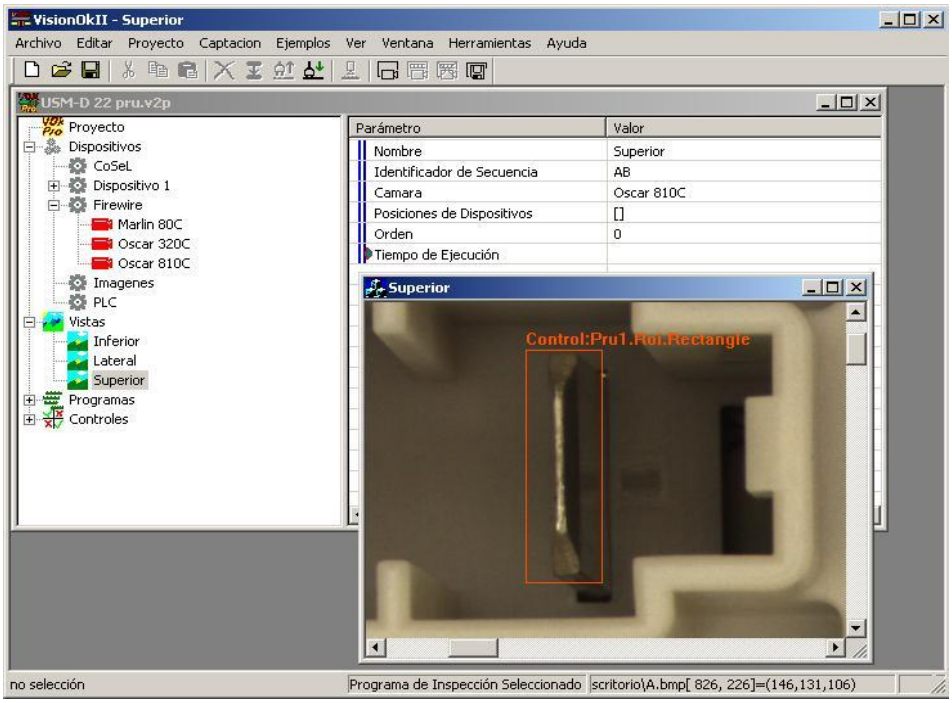

Figura 7. 9 Elección de la vista

Una vez abierta la vista, se puede realizar los diferentes ajustes desarrollados sobre la imagen seleccionada. Para ello, se debe ir al menú *Ver* y, en el submenú *Ajustes,* se debe seleccionar una de las opciones tal y como aparece en la **Figura 7.10**.

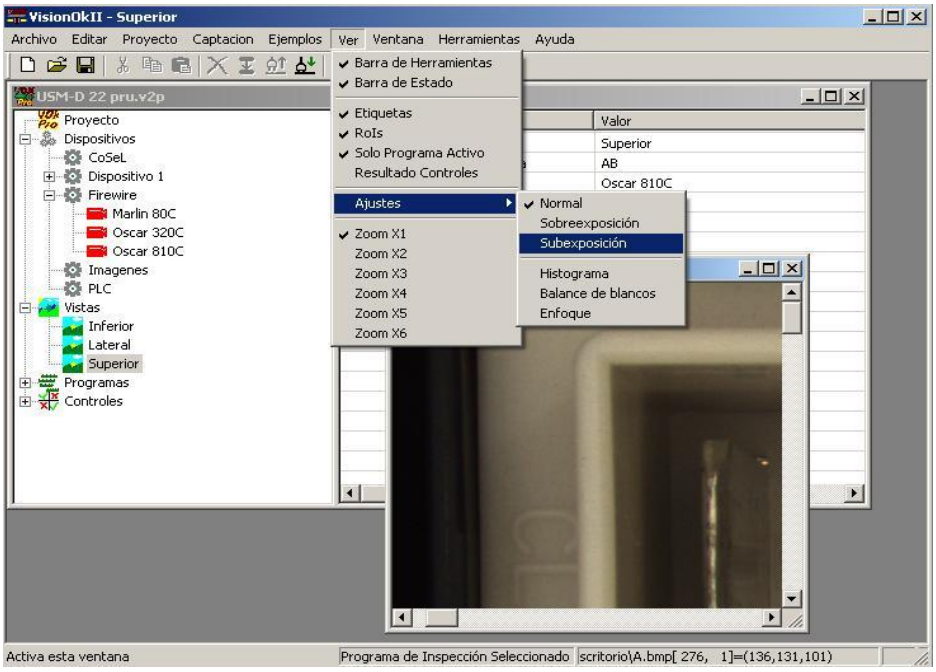

Figura 7. 10 Ajustes de la imagen

En la **Figura 7.11**, se muestra la imagen después de haber aplicado Ajustes→ Sobreexposición. En esta captura se observa los colores sobreexpuestos de los píxeles de la imagen seleccionada. Sobresalen del negro aquellos píxeles que contienen uno o más colores con este defecto.

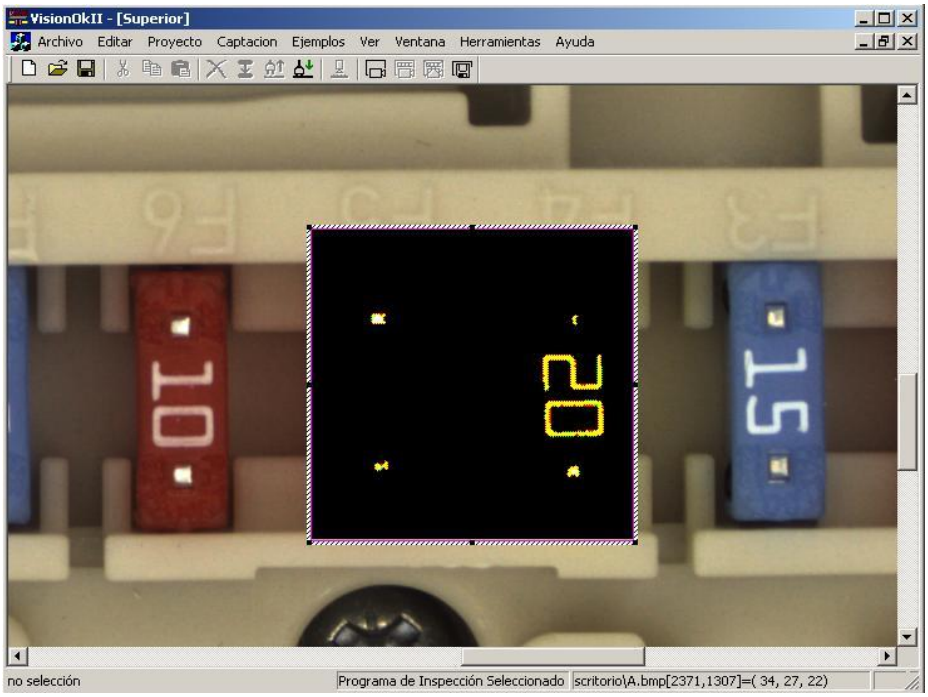

Figura 7. 11 Imagen sobreexpuesta

El histograma recoge información de toda la imagen seleccionada mediante el recuadro de ajustes. En la **Figura 7.13**, se puede observar el histograma RGB en el cual al situar el cursor del ratón encima del histograma se puede ver la cantidad de pixeles que hay de la tonalidad seleccionada. También se puede observar el valor de dicha tonalidad. Además, hay una lista desplegable para elegir el tipo de histograma que se desea visualizar (RGB, Rojo, Azul, Verde).

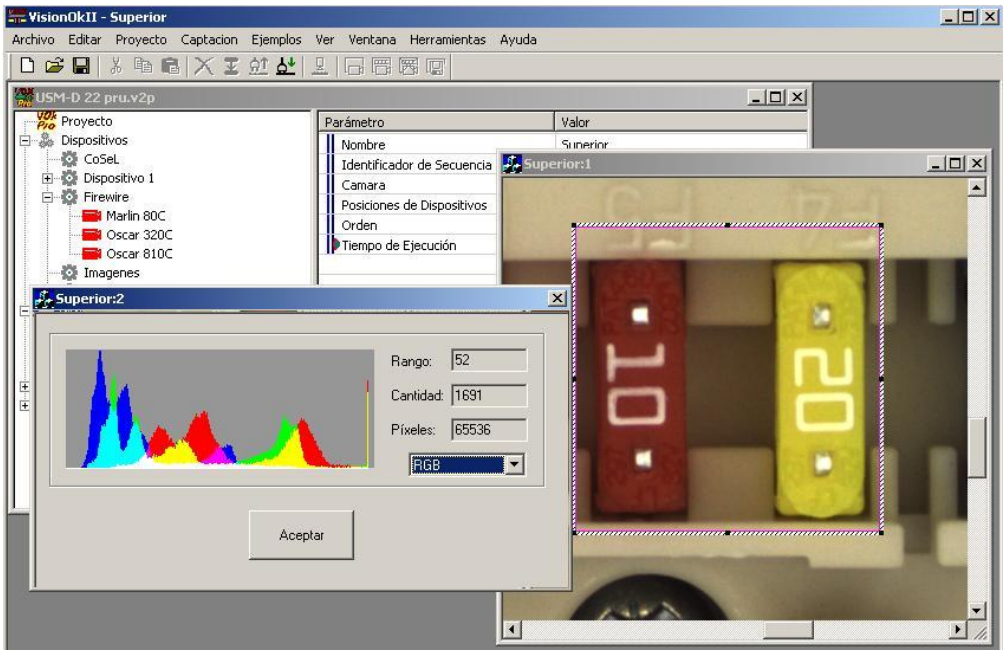

Figura 7. 12 Histograma de la imagen

La siguiente figura nos muestra el módulo de configuración del balance de blancos. Se puede observar que el *frame* captado tiende a tener un dominante de color verde ya que los valores rojo y azul son bastante más pequeños que el anterior.

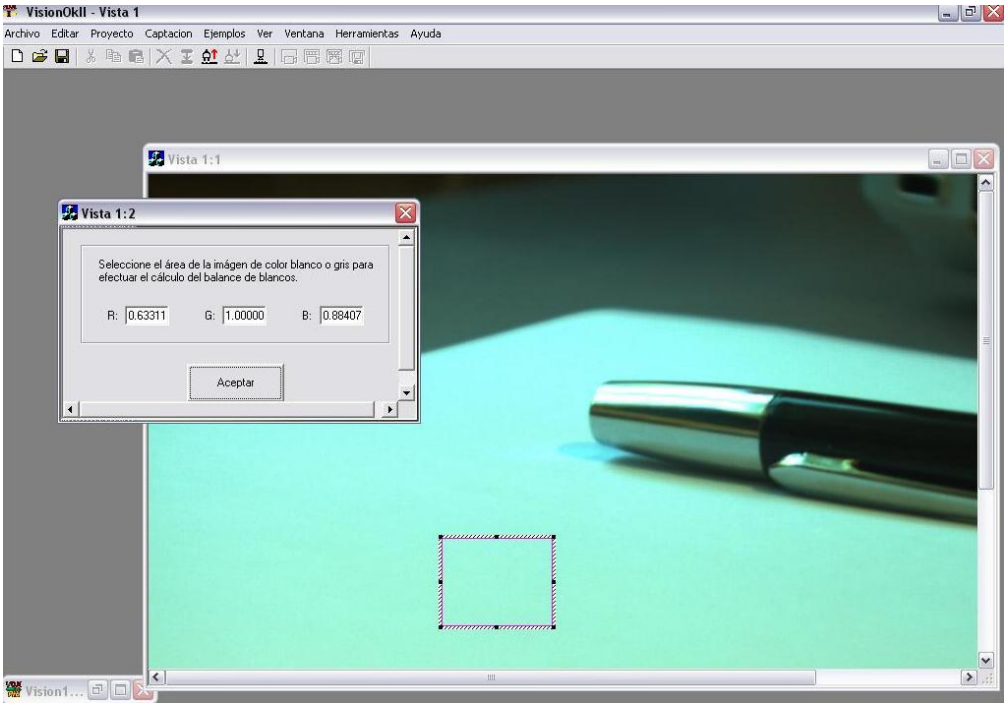

Figura 7. 13 Balance de blancos

En la **Figura 7.15** se puede ver el módulo de enfoque en acción. Se observa que el *frame* actual no se encuentra bien enfocado y, por tanto, la barra de color azul no se muestra a la misma altura que la línea vertical verde que marca el máximo enfoque logrado en esa área de ajuste.

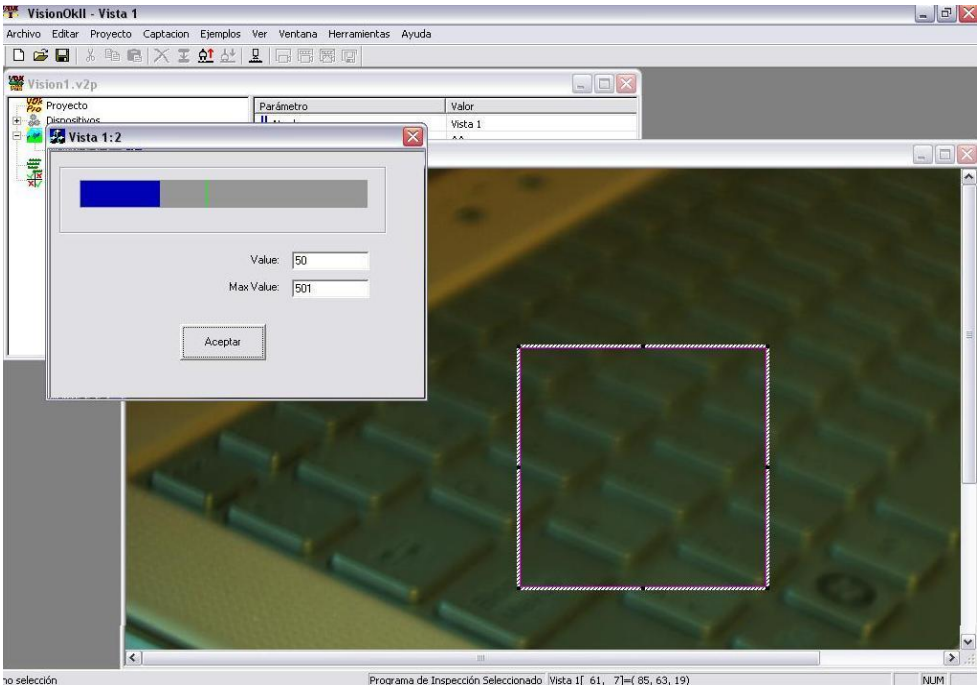

Figura 7. 14 Exposición

En la **Figura 7.16**, se muestra como se mueve el recuadro de ajustes, al mover del ratón de la posición inicial a la posición deseada, donde se situara el recuadro. También es posible moverlo mediante las flechas del teclado.

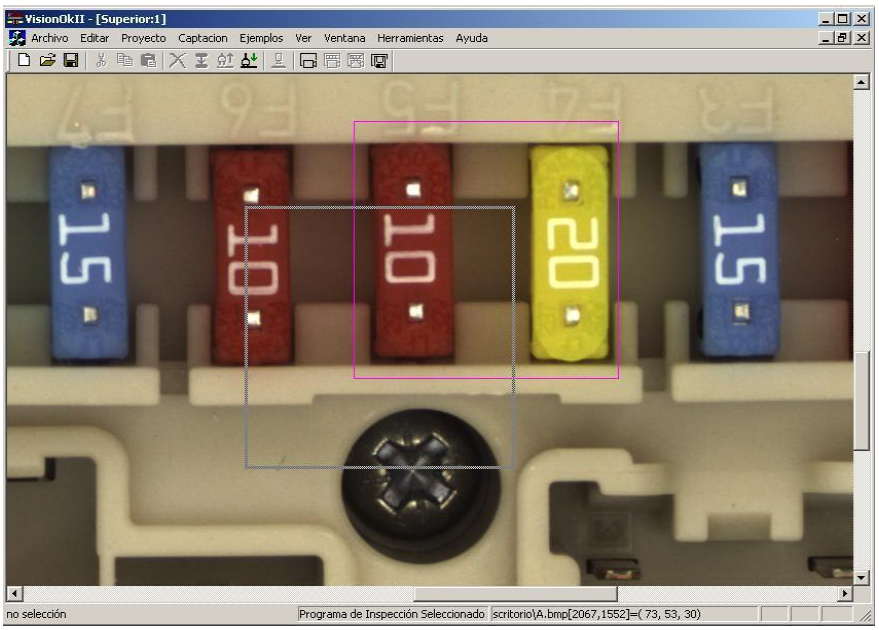

Figura 7. 15 Movimiento rectángulo de ajuste

Una vez realizados todos los ajustes necesarios, se puede volver a visualizar la imagen original seleccionando la opción Normal del submenú Ajustes, tal y como se puede ver en la **Figura 7.17**.

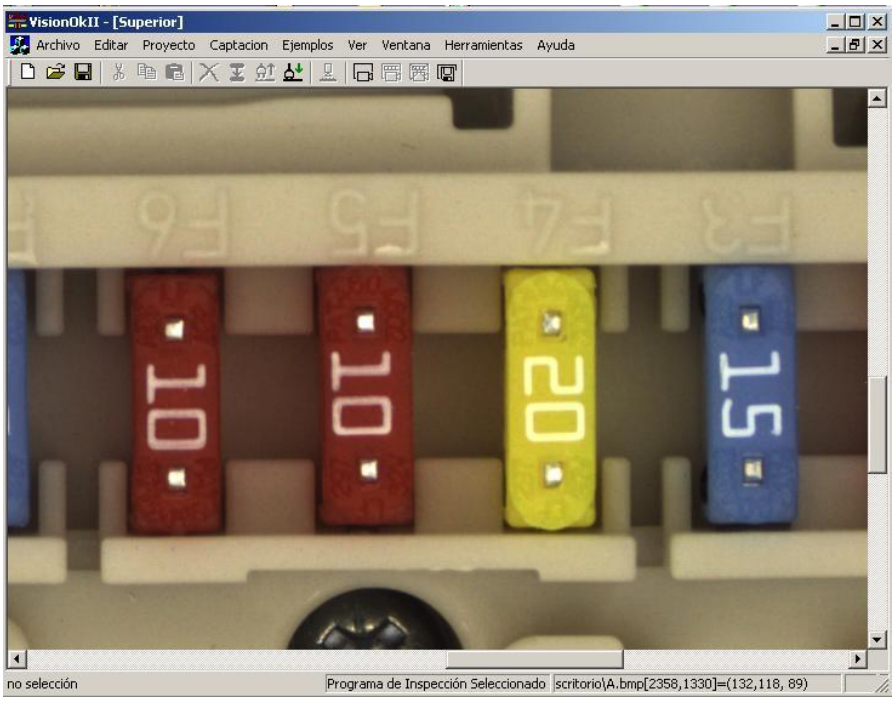

Figura 7. 16 Imagen normal

Offset	Name	Field	Bit	Description
100h	V_FORMAT_INQ	Format_0	[0]	VGA non-compressed format. (Maximum 640x480)
		Format_1	[1]	VGA non-compressed format (1)
		Format_2	[2]	VGA non-compressed format (2)
		Format_x	[3..5]	Reserved for other format
		Format_6	[6]	Still Image Format
		Format_7	[7]	Partial Image Size Format
		-	[8..31]	Reserved (All zero)

Figura 7. 17 Inquiry Registrer for video format

Offset	Name	Field	Bit	Description
180h	V_MODE_INQ_0 (Format_0)	Mode_0	[0]	160x120 YUV (4:4:4) Mode (24bit/pixel)
		Mode_1	[1]	320x240 YUV (4:2:2) Mode (16bit/pixel)
		Mode_2	[2]	640x480 YUV (4:1:1) Mode (12bit/pixel)
		Mode_3	[3]	640x480 YUV (4:2:2) Mode (16bit/pixel)
		Mode_4	[4]	640x480 RGB Mode (24bit/pixel)
		Mode_5	[5]	640x480 Y (Mono) Mode (8bit/pixel)
		Mode_6	[6]	640x480 Y (Mono 16) Mode (16bit/pixel)
		Mode_x	[7]	Reserved for another Mode
		-	[8-31]	Reserved (All zero)
184h	V_MODE_INQ_1 (Format_1)	Mode_0	[0]	800x600 YUV (4:2:2) Mode (16bit/pixel)
		Mode_1	[1]	800x600 RGB Mode (24bit/pixel)
		Mode_2	[2]	800x600 Y (Mono) Mode (8bit/pixel)
		Mode_3	[3]	1024x768 YUV (4:2:2) Mode (16bit/pixel)
		Mode_4	[4]	1024x768 RGB Mode (24bit/pixel)
		Mode_5	[5]	1024x768 Y (Mono) Mode (8bit/pixel)
		Mode_6	[6]	800x600 Y (Mono 16) Mode (16bit/pixel)
		Mode_x	[7]	1024x768 Y (Mono 16) Mode (16bit/pixel)
		-	[8-31]	Reserved (All zero)
188h	V_MODE_INQ_2 (Format_2)	Mode_0	[0]	1280x960 YUV (4:2:2) Mode (16bit/pixel)
		Mode_1	[1]	1280x960 RGB Mode (24bit/pixel)
		Mode_2	[2]	1280x960 Y (Mono) Mode (8bit/pixel)
		Mode_3	[3]	1600x1200 YUV (4:2:2) Mode (16bit/pixel)
		Mode_4	[4]	1600x1200 RGB Mode (24bit/pixel)
		Mode_5	[5]	1600x1200 Y (Mono) Mode (8bit/pixel)
		Mode_6	[6]	1280x960 Y (Mono 16) Mode (16bit/pixel)
		Mode_x	[7]	1600x1200 Y (Mono 16) Mode (16bit/pixel)
		-	[8-31]	Reserved (All zero)
18Ch ... 197h	Reserved for other V_MODE_INQ_x for Format _x			
198h	V_MODE_INQ_6 (Format_6)	Mode_0	[0]	Exit format
		Mode_x	[1..7]	Reserved for another Mode
		-	[8..31]	Reserved (All zero)
19Ch	V_MODE_INQ_7 (Format_7)	Mode_0	[0]	Format_7 Mode_0
		Mode_1	[1]	Format_7 Mode_1
		Mode_2	[2]	Format_7 Mode_2
		Mode_3	[3]	Format_7 Mode_3
		Mode_4	[4]	Format_7 Mode_4
		Mode_5	[5]	Format_7 Mode_5
		Mode_6	[6]	Format_7 Mode_6
		Mode_x	[7]	Format_7 Mode_7
		-	[8-31]	Reserved (All zero)

Figura 7. 18 Inquiry Registrer for video mode

Offset	Name	Field	Bit	Description
200h	V_RATE_INQ_0_0 (Format_0, Mode_0)	FrameRate_0	[0]	Reserved
		FrameRate_1	[1]	Reserved
		FrameRate_2	[2]	7.5 fps
		FrameRate_3	[3]	15 fps
		FrameRate_4	[4]	30 fps
		FrameRate_x	[5..7]	Reserved for another frame rate
		-	[8..31]	Reserved (All zero)
204h	V_RATE_INQ_0_1 (Format_0, Mode_1)	FrameRate_0	[0]	Reserved
		FrameRate_1	[1]	3.75 fps
		FrameRate_2	[2]	7.5 fps
		FrameRate_3	[3]	15 fps
		FrameRate_4	[4]	30 fps
		FrameRate_x	[5..7]	Reserved for another frame rate
		-	[8..31]	Reserved (All zero)
208h	V_RATE_INQ_0_2 (Format_0, Mode_2)	FrameRate_0	[0]	Reserved
		FrameRate_1	[1]	3.75 fps
		FrameRate_2	[2]	7.5 fps
		FrameRate_3	[3]	15 fps
		FrameRate_4	[4]	30 fps
		FrameRate_x	[5..7]	Reserved for another frame rate
		-	[8..31]	Reserved (All zero)
20Ch	V_RATE_INQ_0_3 (Format_0, Mode_3)	FrameRate_0	[0]	Reserved
		FrameRate_1	[1]	3.75 fps
		FrameRate_2	[2]	7.5 fps
		FrameRate_3	[3]	15 fps
		FrameRate_4	[4]	30 fps
		FrameRate_x	[5..7]	Reserved for another frame rate
		-	[8..31]	Reserved (All zero)
210h	V_RATE_INQ_0_4 (Format_0, Mode_4)	FrameRate_0	[0]	Reserved
		FrameRate_1	[1]	3.75 fps
		FrameRate_2	[2]	7.5 fps
		FrameRate_3	[3]	15 fps
		FrameRate_4	[4]	30 fps
		FrameRate_x	[5..7]	Reserved for another frame rate
		-	[8..31]	Reserved (All zero)
214h	V_RATE_INQ_0_5 (Format_0, Mode_5)	FrameRate_0	[0]	Reserved
		FrameRate_1	[1]	3.75 fps
		FrameRate_2	[2]	7.5 fps
		FrameRate_3	[3]	15 fps
		FrameRate_4	[4]	30 fps
		FrameRate_5	[5]	60 fps
		FrameRate_x	[6..7]	Reserved for another frame rate
218h	V_RATE_INQ_0_6	FrameRate_0	[0]	Reserved

	(Format_0, Mode_6)	FrameRate_1	[1]	3.75 fps
		FrameRate_2	[2]	7.5 fps
		FrameRate_3	[3]	15 fps
		FrameRate_4	[4]	30 fps
		FrameRate_x	[5..7]	Reserved for another frame rate
		-	[8..31]	Reserved (All zero)
21Ch … 21Fh	Reserved V_RATE_INQ_0_x (for other MODE_x of Format _0)			
220h	V_RATE_INQ_1_0 (Format_1, Mode_0)	FrameRate_0	[0]	Reserved
		FrameRate_1	[1]	3.75 fps
		FrameRate_2	[2]	7.5 fps
		FrameRate_3	[3]	15 fps
		FrameRate_4	[4]	30 fps
		FrameRate_x	[5..7]	Reserved for another frame rate
		-	[8..31]	Reserved (All zero)
224h	V_RATE_INQ_1_1 (Format_1, Mode_1)	FrameRate_0	[0]	Reserved
		FrameRate_1	[1]	Reserved
		FrameRate_2	[2]	7.5 fps
		FrameRate_3	[3]	15 fps
		FrameRate_x	[4..7]	Reserved for another frame rate
		-	[8..31]	Reserved (All zero)
228h	V_RATE_INQ_1_2 (Format_1, Mode_2)	FrameRate_0	[0]	Reserved
		FrameRate_1	[1]	Reserved
		FrameRate_2	[2]	7.5 fps
		FrameRate_3	[3]	15 fps
		FrameRate_4	[4]	30 fps
		FrameRate_5	[5]	60 fps
		FrameRate_x	[6..7]	Reserved for another frame rate
		-	[8..31]	Reserved (All zero)
22Ch	V_RATE_INQ_1_3 (Format_1, Mode_3)	FrameRate_0	[0]	1.875 fps
		FrameRate_1	[1]	3.75 fps
		FrameRate_2	[2]	7.5 fps
		FrameRate_3	[3]	15 fps
		FrameRate_x	[4..7]	Reserved for another frame rate
		-	[8..31]	Reserved (All zero)
230Ch	V_RATE_INQ_1_4 (Format_1, Mode_4)	FrameRate_0	[0]	1.875 fps
		FrameRate_1	[1]	3.75 fps
		FrameRate_2	[2]	7.5 fps
		FrameRate_x	[3..7]	Reserved for another frame rate
		-	[8..31]	Reserved (All zero)
234h	V_RATE_INQ_1_5 (Format_1, Mode_5)	FrameRate_0	[0]	1.875 fps
		FrameRate_1	[1]	3.75 fps
		FrameRate_2	[2]	7.5 fps
		FrameRate_3	[3]	15 fps
		FrameRate_4	[4]	30 fps

		FrameRate_x	[5..7]	Reserved for another frame rate
		-	[8..31]	Reserved (All zero)
238h	V_RATE_INQ_1_6 (Format_1, Mode_6)	FrameRate_0	[0]	Reserved
		FrameRate_1	[1]	3.75 fps
		FrameRate_2	[2]	7.5 fps
		FrameRate_3	[3]	15 fps
		FrameRate_4	[4]	30 fps
		FrameRate_x	[5..7]	Reserved for another frame rate
		-	[8..31]	Reserved (All zero)
23Ch	V_RATE_INQ_1_7 (Format_1, Mode_7)	FrameRate_0	[0]	1.875 fps
		FrameRate_1	[1]	3.75 fps
		FrameRate_2	[2]	7.5 fps
		FrameRate_3	[3]	15 fps
		FrameRate_x	[4..7]	Reserved for another frame rate
		-	[8..31]	Reserved (All zero)
240h	V_RATE_INQ_2_0 (Format_2, Mode_0)	FrameRate_0	[0]	1.875 fps
		FrameRate_1	[1]	3.75 fps
		FrameRate_2	[2]	7.5 fps
		FrameRate_3	[3]	15 fps
		FrameRate_x	[4..7]	Reserved for another frame rate
		-	[8..31]	Reserved (All zero)
244h	V_RATE_INQ_2_1 (Format_2, Mode_1)	FrameRate_0	[0]	1.875 fps
		FrameRate_1	[1]	3.75 fps
		FrameRate_2	[2]	7.5 fps
		FrameRate_x	[3..7]	Reserved for another frame rate
		-	[8..31]	Reserved (All zero)
248h	V_RATE_INQ_2_2 (Format_2, Mode_2)	FrameRate_0	[0]	1.875 fps
		FrameRate_1	[1]	3.75 fps
		FrameRate_2	[2]	7.5 fps
		FrameRate_3	[3]	15 fps
		FrameRate_x	[4..7]	Reserved for another frame rate
		-	[8..31]	Reserved (All zero)
24Ch	V_RATE_INQ_2_3 (Format_2, Mode_3)	FrameRate_0	[0]	1.875 fps
		FrameRate_1	[1]	3.75 fps
		FrameRate_2	[2]	7.5 fps
		FrameRate_x	[3..7]	Reserved for another frame rate
		-	[8..31]	Reserved (All zero)
250h	V_RATE_INQ_2_4 (Format_2, Mode_4)	FrameRate_0	[0]	1.875 fps
		FrameRate_1	[1]	3.75 fps
		FrameRate_x	[2..7]	Reserved for another frame rate
		-	[8..31]	Reserved (All zero)
254h	V_RATE_INQ_2_5 (Format_2, Mode_5)	FrameRate_0	[0]	1.875 fps
		FrameRate_1	[1]	3.75 fps
		FrameRate_2	[2]	7.5 fps
		FrameRate_3	[3]	15 fps

		FrameRate_x	[4..7]	Reserved for another frame rate
		-	[8..31]	Reserved (All zero)
258h	V_RATE_INQ_2_6 (Format_2, Mode_6)	FrameRate_0	[0]	1.875 fps
		FrameRate_1	[1]	3.75 fps
		FrameRate_2	[2]	7.5 fps
		FrameRate_x	[3..7]	Reserved for another frame rate
		-	[8..31]	Reserved (All zero)
25Ch	V_RATE_INQ_2_7 (Format_2, Mode_7)	FrameRate_0	[0]	1.875 fps
		FrameRate_1	[1]	3.75 fps
		FrameRate_2	[2]	7.5 fps
		FrameRate_x	[3..7]	Reserved for another frame rate
		-	[8..31]	Reserved (All zero)

Figura 7. 19 Inquiry Registrer for video frame rate

BIBLIOGRAFÍA

[1] http://msdn.microsoft.com

[2] The annotated C++ Reference Manual, de Margaret A. Ellis y Bjarne Stroustrup, publicado por Addison-Wesley Publishing Company. Inc., Reading, Massachusetts, E.U.A. 1990.

[3] http://www.codeguru.com

[4] http://www.codeproject.com

[5] IIDC 1394-based Digital Camera Specification v1.30

[6] http://www.cs.cmu.edu/~iwan/1394/index.html

[7] Image analysis and recognition, 5th International Conference, ICAR 2008 by Aurélio Campilho and Mohamed Kamel (Eds.)

[8] Advances in Electrical Engineering and Computational Science by Sio-long Ao and Len Gelman Editors.

[9] http://es.wikipedia.org/wiki/Balance_de_blancos

[10] http://en.wikipedia.org/wiki/Autofocus

[11] http://es.wikipedia.org/wiki/Operador_Sobel

[12] http://es.wikipedia.org/wiki/Caja_blanca_%28sistemas%29

[13] http://www.quesabesde.com/camdig/articulos.asp?articulo=48

[14] Sistema de adquisición automática de imágenes para microscopio óptico (2007) por Said D. Pertuz Arroyo y Hector R. Ibanez Grandas. Universidad del Norte, Colombia.

ÍNDICE

A

área de ajuste, 32
autoenfoque, 27

B

balance de blancos, 13, 26, 27, 28, 32, 35, 36

C

CDocument, 19, 21, 22, 24, 36
CMultiDocTemplate, 9, 19, 21, 22
CView, 19, 23, 39, 41
CVOkImageDoc, 7, 21, 22, 39, 41
CVOkImageView, 7, 21, 22, 23, 24, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42

D

detección de bordes., 28, 29, 30

E

enfoque, 12, 13, 14, 15, 27, 28, 30, 32, 40, 41, 42
exposición, 12, 15, 25, 26, 32, 34
Exposición, 8, 9, 14, 25, 26, 33

F

framework, 18

H

histograma, 12, 13, 14, 15, 24, 25, 32, 37, 38, 39, 40

M

MFC, 7, 18, 19, 45, 47

MDI, 18, 19

O

ON_COMMAND, 9, 24
ON_UPDATE_COMMAND_UI, 9, 24

P

Prewitt, 9, 29

R

RGB, 20, 25, 33, 34, 35, 37, 39, 40, 43, 47

S

Sobel, 9, 29, 30, 46
sobreexposición, 13, 34
Sobreexposición, 25
subexposición, 13, 34
Subexposición, 25

T

temperatura de calor, 26, 47
Tenengrad, 30, 31, 41, 42

V

VisionOkII, 7, 17, 18, 19, 21, 24, 28, 44

W

WINDOWPLACEMENT, 9, 20

Firmado: Daniel Haro Ruiz

Bellaterra, 2 de Febrero de 2010

RESUMEN

Este proyecto está destinado a proporcionar herramientas que ayuden en la configuración de cámaras para uso industrial pero también se pueden exportar a otros campos relacionados con el vídeo y la fotografía digital. Las herramientas desarrolladas ayudan a la configuración del balance de blancos, el grado de exposición al que está sometida la escena capturada por la cámara y a calcular el mejor enfoque posible. Además, se ha desarrollado una herramienta que permita al usuario una configuración más cómoda de los parámetros de la cámara. Estos módulos se han desarrollado sobre el software VisionOkII.

RESUM

Aquest projecte està destinat a proporcionar eines que ajudin en la configuració de càmeres per a ús industrial però també es pot exportar a altres camps relacionats amb el vídeo i la fotografia digital. Les eines desenvolupades ajuden a la configuració del balanç de blancs, el grau d'exposició al que està sotmesa la escena capturada per la càmera i el càlcul del millor enfoc possible. A més, s'ha desenvolupat una eina que permet a l'usuari dur a terme una configuració més còmode dels paràmetres de la càmera. Aquests mòduls s'han desenvolupat sobre el software VisionOkII.

ABSTRACT

This project is destined to provide tools to firewire cameras. The developed tools help to the configuration of white balance, the degree of exposure and to calculate the best possible approach. Moreover, it has developed a tool that allows to the user a more comfortable configuration of the parameters of the camera. These modules have developed on the software VisionOkII.