PROYECTO FIN DE GRADO

ENTORNO DE PROGRAMACIÓN GRÁFICO OPENCV

AUTOR:

Daniel Chamizo Alberto

INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TUTORES:

José María Armingol Moreno

César Hernan Rodríguez Garavito

DEPARTAMENTO DE INGENIERÍA DE SISTEMAS Y AUTOMÁTICA

ESCUELA POLITÉCNICA

UNIVERSIDAD CARLOS III DE MADRID

SEPTIEMBRE 2013

Agradecimientos

Quería agradecer a mi familia, en especial a mi madre, que siempre ha estado ahí dando esos empujones finales cuando más lo he necesitado. Agradecer a César por su gran ayuda y colaboración que ha servido para guiarme durante todo el proyecto y a José María por darme la oportunidad de realizar este proyecto.

ENTORNO DE PROGRAMACIÓN GRÁFICO OPENCV

Índice General.

Índice General

ENTORNO DE PROGRAMACIÓN GRÁFICO OPENCV

Índice de Figuras.

Índice de Figuras

ENTORNO DE PROGRAMACIÓN GRÁFICO OPENCV

Índice de Tablas.

Índice de Tablas

ENTORNO DE PROGRAMACIÓN GRÁFICO OPENCV

Capítulo 1. Motivación y Objetivos

1. Motivación y Objetivos

1.1. Motivación

La motivación del alumno para realizar este trabajo de fin de grado se debe a su interés en la visión por computador. La programación con las librerías OpenCV (Open Source Computer Vision Library) está a la orden del día, por lo que surge la idea de crear una herramienta para que los usuarios que utilizan estas librerías, puedan ahorrarse esfuerzo y tiempo.

Con esta idea se diseña el entorno gráfico en código abierto, de manera que en un futuro, se podrá seguir aprovechando esta herramienta añadiéndola más funciones para poder llegar a desarrollar una entorno potente.

1.2. Objetivos

El objetivo del trabajo de fin de grado es el diseño e implementación de un entorno gráfico para que un usuario pueda desarrollar algoritmos de visión por computador.

Se pretende que el entorno gráfico sea fácil e intuitivo para que el usuario pueda trabajar con las librerías OpenCV de una manera práctica y rápida. Tendrá implementado unas funciones básicas de dichas librerías y una vez terminadas, se dejarán en código abierto para que se puedan mejorar en un futuro.

El entorno gráfico cuenta con una serie de bloques o módulos en donde el usuario puede utilizar las funciones básicas de OpenCV.

Los bloques o módulos constarán de tres pestañas. Cada pestaña tendrá una función diferente:

- Entrada de información.
- Procesamiento.
- Salida de datos.

La entrada de información se realiza mediante la captura de imágenes desde archivo o cámaras web. La salida de datos, es decir, los resultados del procesamiento se muestran de manera gráfica a través de reportes.

El lenguaje con el que se desarrolla el trabajo de fin de grado es C++ y se utiliza el IDE (entorno de desarrollo integrado) Qt Creator.

La herramienta diseñada está orientada a los usuarios que estén familiarizados con las librerías OpenCV.

ENTORNO DE PROGRAMACIÓN GRÁFICO OPENCV

Capítulo 2. Introducción

2. Introducción

2.1. Introducción a la visión por computador

La visión por computador o visión técnica es un campo de la inteligencia artificial. El propósito de esta materia es programar un computador para que "entienda" una escena o las características de una imagen. Este término ha sido muy utilizado en los últimos años. En la figura 1 se puede ver las materias relacionadas con la visión por computador.

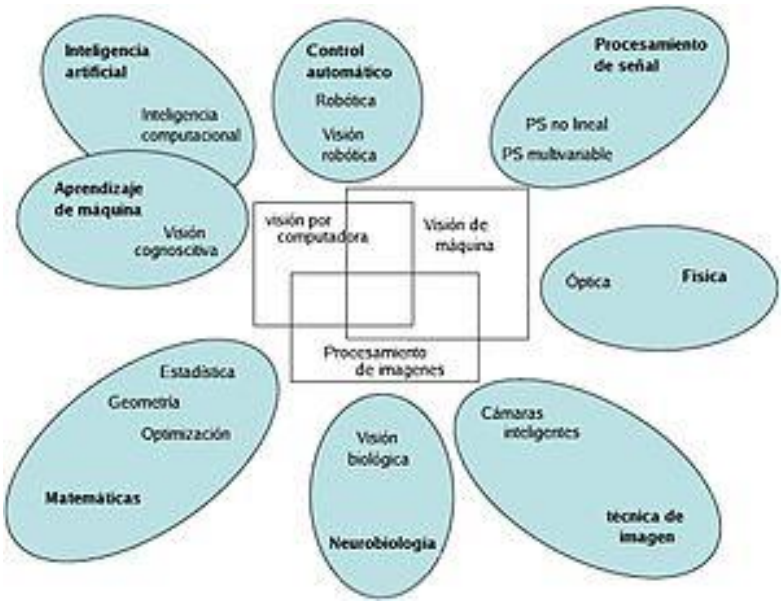


Figura 1. ***Esquema de las relaciones entre la visión por computador y otras áreas afines. [9]***

Los objetivos típicos de la visión por computador incluyen:

- La detección, segmentación, localización y reconocimiento de ciertos objetos en imágenes, por ejemplo, caras humanas.
- La evaluación de los resultados, por ejemplo, segmentación, registro, etc.
- Registro de diferentes imágenes de una misma escena u objeto, es decir, hacer concordar un mismo objeto en diversas imágenes.
- Seguimiento de un objeto en una secuencia de imágenes.

- Mapeo de una escena para generar un modelo tridimensional de la misma; este modelo podría ser usado por un robot para navegar por la escena.
- Estimación de las posturas tridimensionales de humanos.
- Búsqueda de imágenes digitales por su contenido.

Estos objetivos se consiguen por medio del reconocimiento de patrones, aprendizaje estadístico, geometría de proyección, procesamiento de imágenes, teoría de grafos y otros campos.

En los capítulos siguientes se puede observar que el entorno gráfico OpenCV no utiliza todos los campos en los que la visión por computador se relaciona. Este entorno se enfoca principalmente en una de las partes básicas que conforman la materia, que es el procesamiento de imágenes. Un ejemplo de procesamiento de imagen se puede ver en la figura 2.

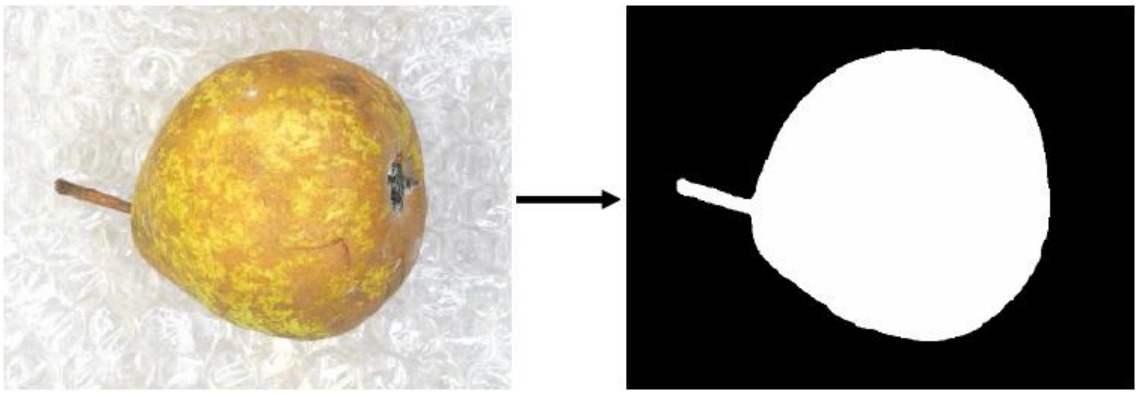


Figura 2. ***Procesamiento de imágenes: Segmentación. [10]***

2.2. Programación G

G es un lenguaje de alto nivel, cuyo flujo de datos gráficos de programación está diseñado para el desarrollo de aplicaciones que cumplan las siguientes propiedades:

- Interactividad
- Ejecuciones en paralelo
- Multi-core

Este lenguaje, en lugar de consistir en declaraciones textuales, utiliza componentes gráficos y una especie de diagrama de flujo de datos para describir la lógica del programa.

El lenguaje de programación G es un lenguaje de programación completo, es decir, se puede programar cualquier cosa con él. Ofrece la típica selección de construcciones, tales como bucles "for" y "while", estructuras de casos de ejecución condicional, y secuencias para la ejecución lineal.

Los operadores habituales utilizados en las expresiones están presentes para operaciones como las comparaciones, operaciones lógicas (booleanas), tales como AND, OR, NOT, aritmética, y muchos más.

Durante el desarrollo de este lenguaje de programación, ha habido aplicaciones que lo han usado como base tales como LabVIEW[12], Titere[11] y MatLab[13]. La aplicación de referencia que se ha seguido para el diseño del entorno gráfico OpenCV ha sido Titere. Esta aplicación es una herramienta bajo licencia GPL para tratamiento de imágenes digitales. Es una herramienta 100% Java, con una finalidad puramente didáctica para que cualquier usuario pueda ver la aplicación de diferentes algoritmos a las imágenes deseadas.

Un ejemplo de uso del lenguaje de programación G se puede ver en la figura 3 donde se puede ver como se usa en el entorno LabView:

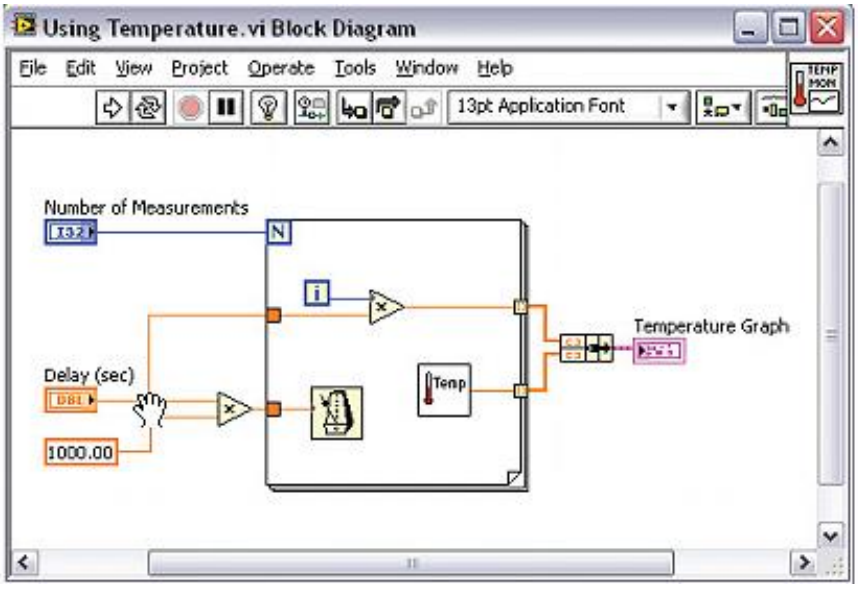


Figura 3. *Diagrama de bloques en LabView. [12]*

2.3. Librerías OpenCV

Las librerías OpenCV (Open Source Computer Vision Library)[13] se utilizan como una biblioteca libre de visión artificial. Originalmente estas bibliotecas fueron desarrolladas por Intel y su primera aparición fue en 1999 con su versión alfa, que se ha utilizado en infinidad de aplicaciones.

Los sistemas más utilizados con estas librerías son sistemas de seguridad con detección de movimiento, y aplicativos de control de procesos donde se requiere reconocimiento de objetos. Su publicación se da bajo licencia BSD, que permite que sea usada libremente para propósitos comerciales y de investigación con las condiciones en ella expresadas.

OpenCV se ejecuta en sistemas operativos como Windows, Android, Maemo, FreeBSD, OpenBSD, iOS, BlackBerry 10, Linux y OS X. Hay más de 500 funciones que abarcan una gran gama de áreas en el proceso de visión.

OpenCV está escrito en C++ y su interfaz principal se encuentra en C++, pero aún conserva una extensa interfaz aunque menos completa en C. En la actualidad, hay interfaces también en otros lenguajes como Python, Java y MATLAB/Octave.

El API de las librerías se puede consultar en la documentación en línea. Todos los nuevos desarrollos y algoritmos de las librerías OpenCV se desarrollan ahora en la interfaz de C++. Estas librerías se pueden descargar de su propia página web y se puede elegir la versión deseada para su posterior uso.

OpenCV trabaja mediante una estructura modular, lo que significa que el paquete incluye varias bibliotecas compartidas o estáticas, que son las siguientes:

- **core** : Un módulo compacto que define las estructuras de datos básicos, como la densa Mat matriz multi-dimensional y las funciones básicas utilizadas por el resto de módulos.
- **imgproc** : Un módulo de procesamiento de imágenes que incluye la imagen lineal y no lineal de filtrado, transformaciones geométricas en imagenes (cambio de tamaño, deformación afín y la perspectiva, genérico reasignación basada en tablas), conversión de espacio de color, histogramas, etc.
- **video** : Un módulo de análisis de vídeo que incluye estimación del movimiento, la sustracción del fondo, y algoritmos de seguimiento de objetos.
- **calib3d** : Este módulo se encarga de las vistas múltiples de algoritmos geométricos básicos, calibración individual y estéreo de cámara, algoritmos de correspondencia estéreo, y los elementos de la reconstrucción 3D.

- **features2d** : Este paquete maneja los detectores de características salientes, descriptores y comparadores.
- **Objdetect** : Detección de objetos e instancias de las clases predefinidas (por ejemplo, las caras, los ojos, las tazas, la gente, los coches, y así sucesivamente).
- **highgui** : Una interfaz de fácil uso con de captura, imagen y video codecs, así como capacidades de interfaz de usuario sencilla.
- **gpu** : Algoritmos acelerados por GPU de diferentes módulos OpenCV.
- Algunos otros módulos auxiliares, como FLANN y Google test wrappers, los enlaces de Python.

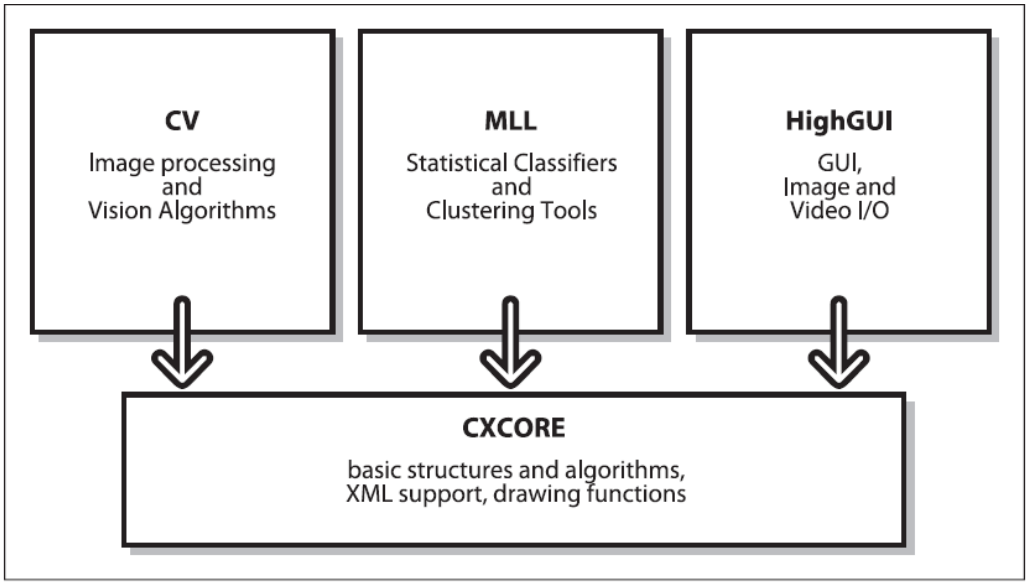


Figura 4. *Esquema de la arquitectura OpenCV. [2]*

2.4. Entorno de desarrollo Qt Creator

Qt Creator es un IDE (entorno de desarrollo integrado), multiplataforma que se ajusta a las necesidades de los desarrolladores Qt.

Qt Creator se centra en proporcionar características que ayudan a los nuevos usuarios a aprender y comenzar a desarrollar rápidamente sus proyectos. También aumenta la productividad de los desarrolladores con experiencia en Qt. Este IDE está compuesto de las siguientes herramientas:

- Editor de código con soporte para C++, QML y ECMAscript
- Herramientas para la rápida navegación del código
- Resaltado de sintaxis y auto-completado de código
- Control estático de código y estilo a medida que se escribe
- Soporte para refactoring de código
- Ayuda sensitiva al contexto
- Plegado de código (code folding)
- Paréntesis coincidentes y modos de selección

Una de las herramientas más importantes que se ha utilizado durante el diseño del proyecto es el depurador visual (visual debugger).

En C++ el debugger es consciente de la estructura de muchas clases de Qt, lo que aumenta la capacidad de mostrar los datos de Qt con claridad.

Gracias a esta herramienta en Qt Creator podemos ver la información en bruto procedente de GDB, de una manera clara y concisa. Se puede realizar las siguientes operaciones si el usuario las estima necesarias:

- Interrupción de la ejecución del programa.
- Ejecución línea por línea o instrucción a instrucción.
- Puntos de interrupción (breakpoints).
- Examinar el contenido de llamadas a la pila (stack), los observadores y de la variables locales y globales.

Otra herramienta crucial es el entorno integrado para la creación y diseño de forms para proyectos C++, que permite diseñar rápidamente widgets y diálogos. Los forms son totalmente funcionales y pueden ser previsualizados.

El nombre de esta herramienta es Qt Designer y ha sido la clave para el desarrollo del proyecto ya que es la base para poder realizar el diseño del mismo.

Qt Creator tiene la posibilidad de realizar diferentes tipos de proyectos en función de lo que el usuario desee diseñar e implementar. Para la realización del trabajo de fin de grado, el alumno ha escogido el proyecto tipo GUI (Graphical User Interface). Con este tipo de proyecto se pueden implementar programas informáticos que actúan de interfaz de usuario, utilizando un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz.

Su principal uso, consiste en proporcionar un entorno visual sencillo para permitir la comunicación con el sistema operativo de una máquina o computador. En la figura 5 se pueden ver los diferentes tipos de proyectos que Qt Creator puede realizar y la elección del proyecto tipo GUI:

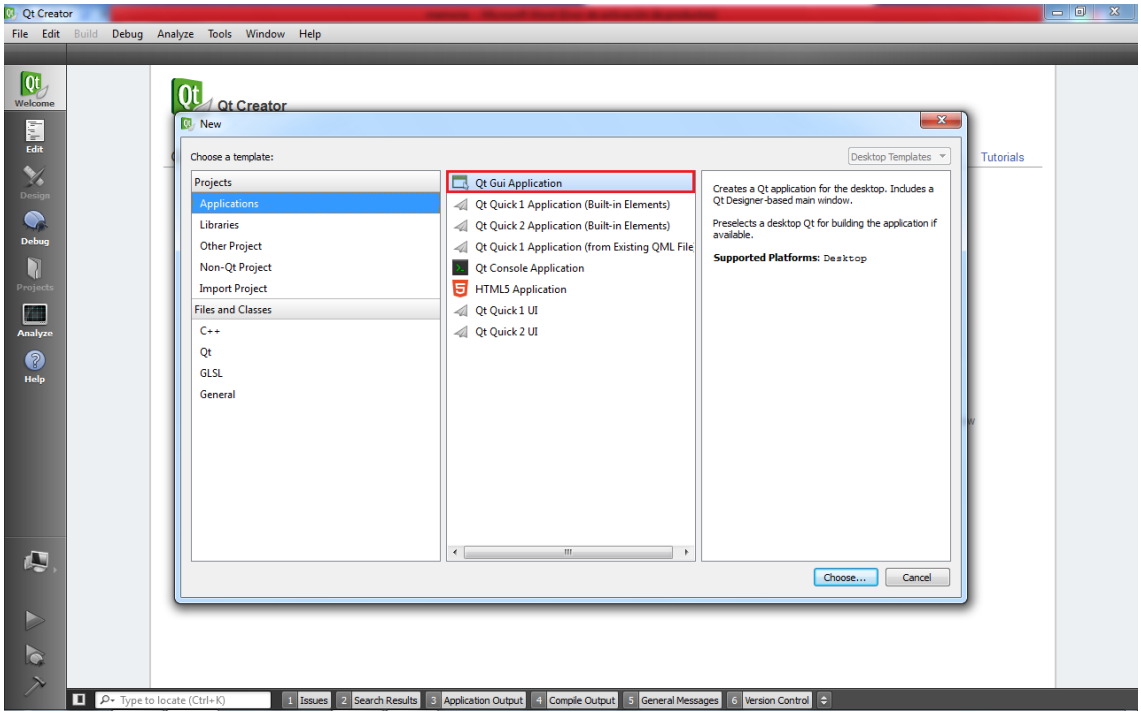


Figura 5. ***Selección de proyecto tipo GUI.***

La elección de este tipo de proyecto se debe a que la implementación del entorno gráfico OpenCV es un conjunto de widgets y cuadros o cajas de diálogos que como se ha mencionado anteriormente, se forman a partir de imágenes y objetos gráficos.

Los widgets son muy variados y de diferentes tipos:

- **Widgets de escritorio**: son los más conocidos. Son herramientas interactivas descargables que se insertan en el escritorio de nuestro ordenador. Suelen ser aplicaciones para: el tiempo, el reloj, buscadores, etc.
- **Web widgets**: partes de código que representan funcionalidades o contenidos que pueden ser instalados y ejecutados en una página web de manera sencilla. El código puede programarse desde: Javacript, Flash, Silverlight y Windows Media Player entre otros. Su objetivo es enriquecer los contenidos y funcionalidades de tu Web sin necesidad de programar y crear nuevos contenidos.

- **Widgets para móviles**: similares a los de escritorio, pero, en este caso, para la interfaz del teléfono. A pesar de lo pequeña que pueda ser su pantalla, los Widgets se adaptan perfectamente a la misma y prestan servicios interactivos de gran calidad.
- **Widgets físicos**: mecanismos compactos interactivos que integran varias funciones típicas de los Widgets utilizados en un ordenador. Sus funcionalidades más comunes pueden ser alarmas despertador, información del tiempo, de Internet, etc.

Figura 6. *Escritorio con diferentes widgets.*

Por otro lado, los cuadros o cajas de diálogos sirven para insertar texto de diferentes maneras en una ventana que puede tener funcionalidades diferentes.

Un cuadro o caja de diálogo es una ventana de nivel superior utilizada principalmente para tareas de corto plazo y comunicaciones breves con el usuario. Estas ventanas pueden ser modales o no modales.

- **Cuadros de diálogo modales**: bloquea la entrada a otras ventanas visibles en la misma aplicación. Suelen ser los diálogos que se utilizan para solicitar un nombre de archivo del usuario o que se utilizan para establecer preferencias de la aplicación. Cuando se abre un cuadro de diálogo modal aplicación, el usuario debe terminar la interacción con el diálogo y cerrarlo antes de que puedan acceder a cualquier otra ventana de la aplicación.
- **Cuadros de diálogo no modales**: funciona de forma independiente de las demás ventanas de la misma aplicación. Buscar y reemplazar diálogos en procesadores de textos son a menudo no modales permitiendo al usuario interactuar tanto con la ventana principal de la aplicación y con el diálogo.

En el proyecto el alumno ha utilizado los cuadros de diálogos modales. Esta elección sirve para que avise al usuario de cualquier tipo de aspecto dentro del entorno gráfico mediante la aparición de una ventana. Dicha ventana bloquea el uso del entorno hasta que el usuario la cierra. Por ello, la intención del alumno, es que el usuario detecte o lea el aviso que se le está dando a través de la ventana. Un ejemplo de ello se puede ver en la figura 7:

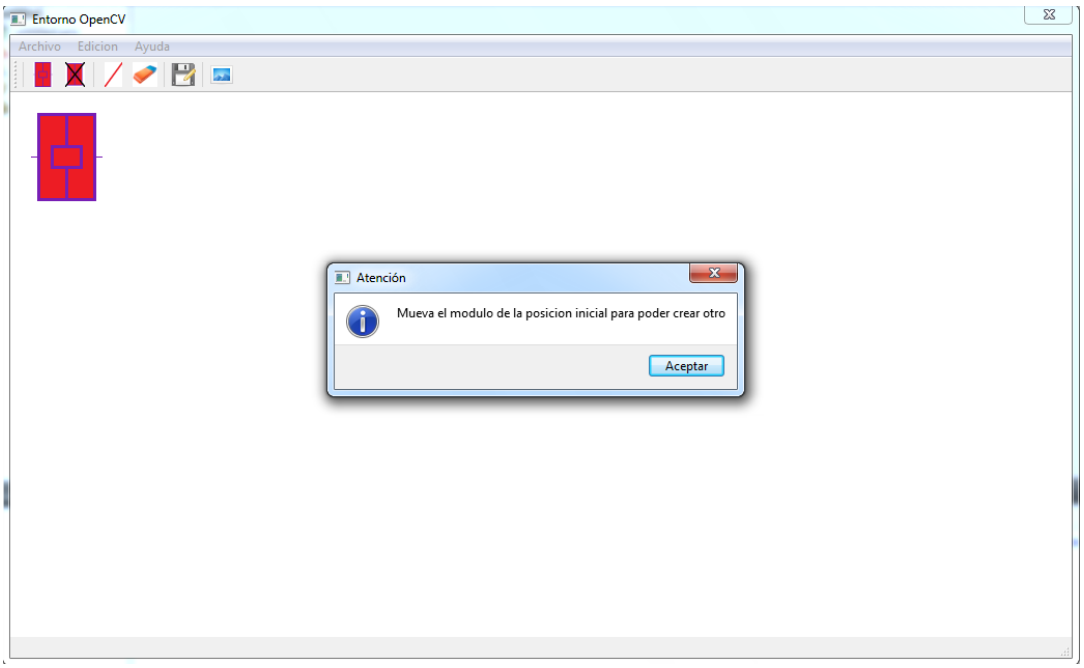


Figura 7. ***Cuadro de diálogo.***

Todos estos recursos los podemos diseñar, utilizar y modificar con el Qt Designer. Se puede ver un ejemplo de cómo usar estos recursos con la herramienta Qt Designer en la figura 8, donde se observa la UI (User Interface) de la clase MainWindow que se explicará en los capítulos siguientes:

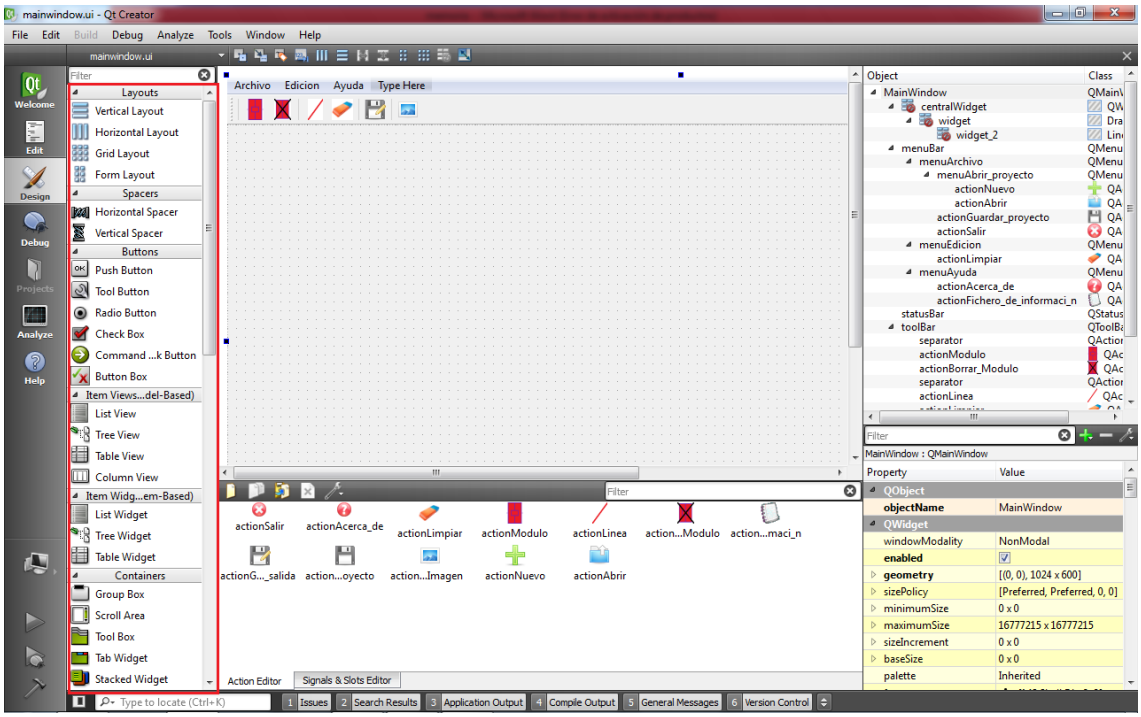


Figura 8. ***Qt Designer.***

2.5. Aplicación de las librerías OpenCV en Qt Creator

Para poder diseñar e implementar el entorno gráfico OpenCV, se necesita la integración de las librerías OpenCV con las herramientas de Qt Creator. Todo esto es necesario ya que el proyecto es un conjunto de widgets y diálogos. Este conjunto sirve para que el usuario pueda trabajar las diferentes funcionalidades que proporciona las librerías OpenCV a modo de interacción con el entorno gráfico.

Qt Creator permite que se puedan anexionar librerías diferentes, a parte de las propias que ya posee, que son las librerías Qt. Por ello, para anexionar las librerías OpenCV, solo se modifica el archivo .pro del proyecto GUI. La explicación más detallada se puede ver en el Anexo I.

ENTORNO DE PROGRAMACIÓN GRÁFICO OPENCV

Capítulo 3.
Entorno gráfico OpenCV

3. Entorno gráfico OpenCV

3.1. Descripción y características generales

El entorno gráfico OpenCV se ha desarrollado con el IDE Qt Creator y las librerías OpenCV en la plataforma o sistema operativo Windows. Este entorno está pensado para usuarios de OpenCV. El lenguaje utilizado es C++.

Su propósito es facilitar el trabajo a los usuarios programadores en el ámbito de procesamiento de imágenes, de una manera rápida y gráfica. Rápida porque el usuario no tiene que escribir ningún tipo de código para realizar el proceso deseado a la imagen, y gráfica porque se puede ver los procesos por los que se va sometiendo a la imagen, a la vez que se puede consultar los reportes de cada proceso.

El entorno tiene unas dimensiones fijas de 1024x600. Son fijas debido a que no se puede maximizar o minimizar la ventana, es decir, la ventana con dichas dimensiones, solo posee el botón cerrar. El alumno tomó esta decisión debido a que es un proyecto en código abierto y la posibilidad de maximizar o minimizar la ventana se considera una mejora futura. En la figura 9, se puede ver la ventana del entorno gráfico OpenCV y en la esquina superior derecha el botón cerrar de la ventana.

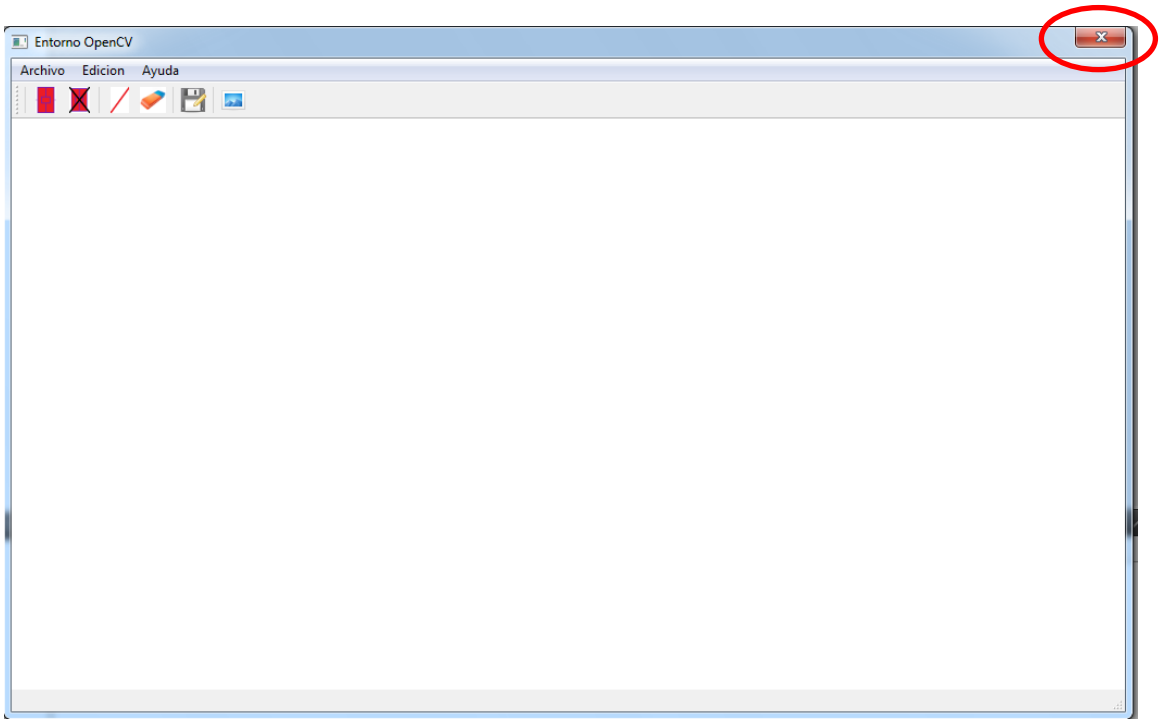


Figura 9. *Ventana del entono gráfico OpenCV.*

En el entorno se pueden diferenciar las siguientes partes:

- CentralWidget
- MenuBar
- ToolBar
- StatusBar

Cada una de estas partes desempeña una función diferente. En los siguientes capítulos se explicarán cada una de éstas partes.

La arquitectura interna del entorno está formada por diferentes clases. Dichas clases se explicarán en el capítulo siguiente. En la figura 10, se puede ver un esquema de la estructura del entorno gráfico OpenCV:

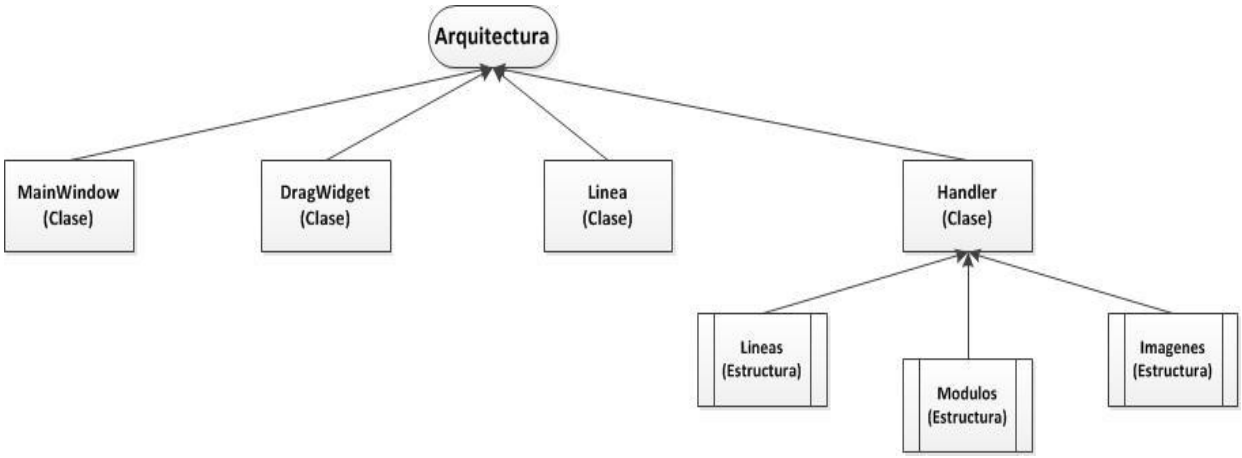


Figura 10. *Arquitectura del entono gráfico OpenCV.*

Aunque las estructuras Lineas, Modulos, e Imágenes son creadas dentro de la clase Handler, la información que contienen puede ser modificada, borrada o creada desde las otras tres clase debido a que dichas estructuras son estáticas.

3.2. Diagramas

Para tener una visión completa de la programación y el uso que el usuario puede dar al entorno se van a mostrar los diagramas de casos de uso y el diagrama de clases UML. En el diagrama de casos se pueden ver los diferentes actores que participan en este proyecto. En el diagrama de clases UML se pueden ver las variables y funciones de cada clase, así como las asociaciones de las mismas.

3.2.1 Diagrama de clases UML

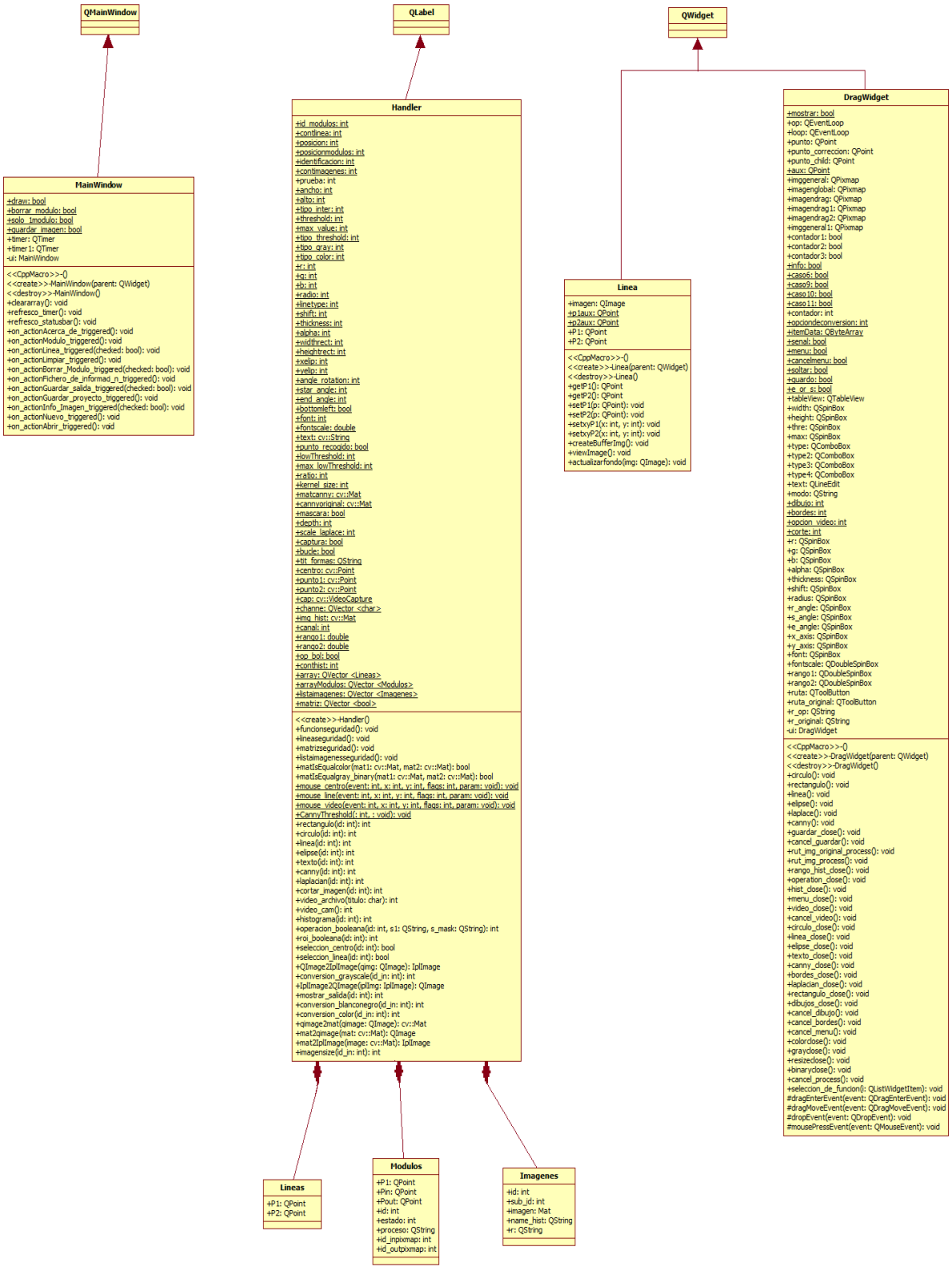


Figura 11. ***Diagrama de clases UML.***

En la figura 11 se puede ver como hay tres estructuras asociadas a la clase Handler. Estas estructuras son accesibles para las otras clases, es decir, las clases MainWindow, DragWidget y Linea pueden utilizar su información. También podemos ver que la clase QWidget es el padre de las clases Linea y DragWidget, la clase QMainWindow es el padre de MainWindow y QLabel es el padre de Handler.

3.2.2. Diagrama de casos de usos

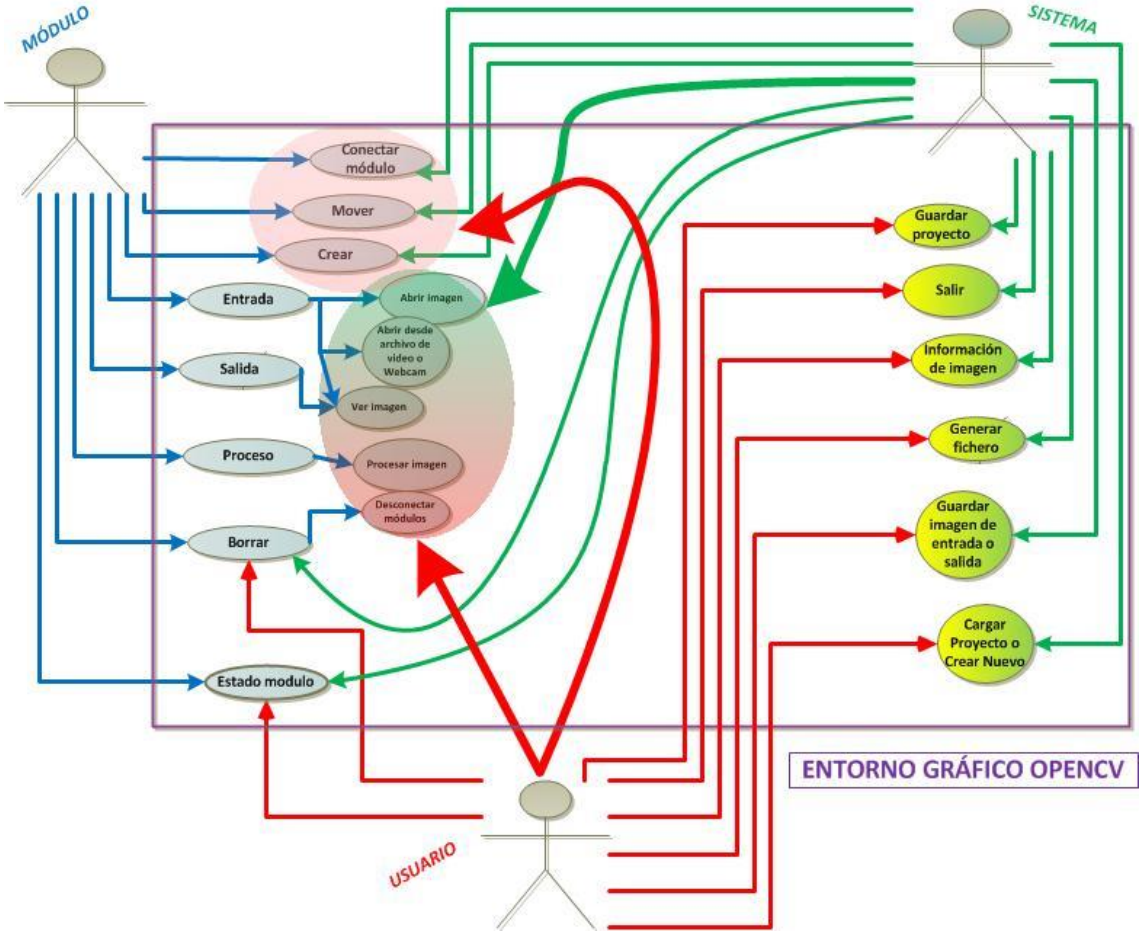


Figura 12. *Diagrama de casos de usos.*

En el diagrama de la figura 12 se puede observar que hay tres actores, el sistema, el módulo y el usuario. La clase Linea y DragWidget actúan en todas las acciones que realiza el módulo (flechas azules). La clase MainWindow actúa en todas las acciones que realiza el sistema (flechas verdes). En flechas de color rojo se puede ver todas las acciones que el usuario puede realizar en el entorno gráfico OpenCV. Dichas acciones, provocan que las tres clases mencionadas en las líneas anteriores interactúen entre ellas intercambiándose información. Esta información se encuentra en la clase Handler.

3.3. Clases y funciones de las mismas

Para la implementación del entorno gráfico OpenCV se han utilizado cuatro clases diferentes. Cada una de las clases desempeña una función diferente, aunque la información es compartida por todas.

3.3.1. Handler

Esta clase es la única que no tiene UI (User Interface) a diferencia de las otras tres. Esto se debe a que no es necesario utilizar una UI para esta clase, ya que su propósito es almacenar la información del entorno en diferentes estructuras. Dichas estructuras se crean en esta clase que también, se encarga del procesamiento de imágenes con las librerías OpenCV, a través de las diferentes funciones que el alumno ha creado.

Otro dato importantes de esta clase, es que se diferencia del resto porque no tiene ningún padre, es decir, no hereda nada de ninguna otra clase ya que no es necesario debido al propósito que desempeña.

Las estructuras creadas para guardar la información son las siguientes:

- Lineas
- Modulos
- Imagenes

En la figura 10 (Arquitectura del entorno gráfico OpenCV) se puede observar dichas estructuras. Cada una de ellas tiene diferentes variables para poder gestionar la información del entorno. Dichas variables se analizan en los capítulos siguientes.

Para usar las estructuras se crean arrays de las mismas. Esta operación, se consigue utilizando la clase QVector, la cual permite crear arrays de cualquier tipo de clase o estructura. En la clase Handler hay cuatro definiciones con la clase QVector:

- `static QVector <Imagenes> listaimagenes;`
- `static QVector <Modulos> arrayModulos[100];`
- `static QVector <Lineas> array[100];`
- `static QVector <bool> matriz[100][100];`

Cada posición del vector tendrá las características de las estructuras por las que el vector está formado. El vector matriz es el único que no está formado por ninguna estructura ya que como se puede observar está formado por booleanos (true o false), es decir, cada posición del vector es un booleano.

Este vector nos servirá para saber las conexiones entre los módulos, por ejemplo, si queremos saber las conexiones del módulo con identificación 2, tendremos que recorrer el vector escribiendo matriz[2][posición que se recorre]. Si en alguna de las posiciones que se recorren, el booleano es "true", entonces el módulo con identificación 2 está conectado a otro modulo con identificación igual a dicha posición.

El valor máximo de posiciones que puede alcanzar los vectores arrayModulos, array y matriz son 99. Esto se debe a que el alumno ha decidido poner de límite el uso de 100 módulos y 100 líneas o uniones entre módulos. En cambio, el vector listaimagenes almacena todas las imágenes que el usuario utiliza o trata durante el ciclo de uso del entorno gráfico OpenCV. Si el usuario realiza dos o más veces el mismo proceso a una imagen, el entorno realiza dicho proceso deseado por el usuario. Antes de añadir la imagen tratada al vector, el entorno la busca en el dicho vector por si la imagen procesada ya existiera, y si es el caso, devuelve su identificador.

En esta clase también se encuentran todas las funciones que el alumno ha creado para realizar los diferentes tratamientos a las imágenes. Las funciones que el alumno ha creado son:

- bool matIsEqualcolor(const cv::Mat mat1, const cv::Mat mat2)
- bool matIsEqualgray_binary(const cv::Mat mat1, const cv::Mat mat2)
- static void mouse_centro(int event, int x, int y, int flags, void* param)
- static void mouse_line(int event, int x, int y, int flags, void* param)
- static void mouse_video(int event, int x, int y, int flags, void* param)
- static void CannyThreshold(int, void *)
- int rectangulo(int id)
- int circulo(int id)
- int linea(int id)
- int elipse(int id)
- int texto(int id)
- int canny(int id)
- int laplacian(int id)
- int cortar_imagen(int id)
- int video_archivo(const char *titulo)
- int video_cam()
- int histograma(int id)
- int operacion_booleana(int id, QString s1, QString s_mask)
- int roi_booleana(int id)
- bool seleccion_centro(int id)
- bool seleccion_linea(int id)
- IplImage* QImage2IplImage(QImage *qimg)
- int conversion_grayscale(int id_in)
- QImage* IplImage2QImage(IplImage *iplImg)
- int mostrar_salida(int id)
- int conversion_blanconegro(int id_in)
- int conversion_color(int id_in)
- int imagensize(int id_in)

En estas funciones es en donde se ha utilizado las librerías OpenCV. Cada una de las funciones realiza un tratamiento a las imágenes, y para realizar dicho tratamiento se ha utilizado funciones que posee la librería OpenCV. En los capítulos siguientes se explicarán cada una de las funciones anteriores de manera más detallada.

3.3.1.1. Estructura Lineas

Esta estructura se encarga de guardar la información de los puntos que forman la línea de unión entre los módulos y está compuesta por las siguientes variables:

- P1 : es una variable de tipo QPoint que nos sirve para saber el primer punto (x,y) de la línea, es decir, el primer punto de conexión entre módulos.
- P2 : es una variable de tipo QPoint que nos sirve para saber el segundo punto (x,y) de la línea, es decir, el segundo punto que completa la conexión entre módulos.

3.3.1.2. Estructura Modulos

Esta estructura es la más compleja ya que tiene toda la información del procesado de la imagen además del flujo de ésta a otros módulos. Está formada por las siguientes variables:

- P1 : es una variable de tipo QPoint que sirve para saber el punto (x,y) de la esquina superior izquierda del módulo.
- Pin : es una variable de tipo QPoint que sirve para saber el punto (x,y) de entrada del módulo. Este punto coincide con el segundo punto de la estructura Lineas (P2).
- Pout : es una variable de tipo QPoint que sirve para saber el punto (x,y) de salida del módulo. Este punto coincide con el primer punto de la estructura Lineas (P1).
- id : es una variable de tipo int que sirve para identificar el modulo.
- estado : es una variable de tipo int que sirve para saber cuál es el estado del módulo. Los diferentes estados en los que puede estar el modulo son:
    - estado inicial, de creación.
    - estado de módulo activo.
    - estado de módulo pendiente de líneas, es decir, pendiente de unión.
- proceso : es una variable de tipo QString que sirve para escribir una cadena de caracteres con el proceso realizado en el módulo. Se utiliza para la StatusBar.
- id_inpixmap : es una variable de tipo int que sirve para saber cuál es el identificador de la imagen asociada a la entrada del módulo.
- id_outpixmap : es una variable de tipo int que nos sirve para saber cuál es el identificador de la imagen asociada a la salida del módulo.

3.3.1.3. Estructura Imágenes

Esta estructura se encarga de almacenar las imágenes que utiliza el usuario en el entorno. Está formada por las siguientes variables:

- id : es una variable de tipo int que sirve para dar un número de identificación a la imagen. Este número será el que almacenará en la variable id_inpixmap o id_outpixmap de la estructura Modulos. Se dan dos casos:
  - Si id >0 la imagen es original o tratada.
  - Si id <= -3 la imagen es un histograma.
- sub_id : es una variable de tipo int que sirve para saber si es una imagen es:
  - Original (imagen cargada sin ningún tratamiento).
  - Captura de video.
  - Tratada.
- imagen : es una variable de tipo Mat donde se guarda la imagen cargada o a tratar. Las variables de tipo Mat son estructuras de las librerías OpenCV que almacenan toda la información de las imágenes.
- name_hist : es una variable de tipo QString que sirve para saber cuál es el canal de la imagen que ha elegido el usuario. Esta variable es nula si el tratamiento que se realiza a la imagen es diferente a Histograma.
- r : es una variable de tipo QString que almacena la ruta de la imagen. Se necesita para el uso de guardar y cargar proyecto.

3.3.2. MainWindow

Esta clase es la más importante de todas, el propósito de la misma es el control gráfico del entorno, es decir, maneja la información que se necesita para realizar las tareas gráficas necesarias cuando el usuario lo precisa, como por ejemplo, el uso de la barra de herramientas del entorno cuando el usuario quiere unir dos módulos.

Para que esta clase pueda realizar lo que se acaba de comentar en el párrafo anterior, tiene su propia UI (User Interface) de diseño de forms. No es la única clase que lo posee, pero la UI de esta clase maneja directamente las UIs de las otras clases. Esta UI hereda todo de la clase QMainWindow.

La UI de esta clase se distingue porque es la ventana del propio entorno junto con todas sus partes, es decir, maneja las diferentes partes que conforman el mismo:

- CentralWidget
- MenuBar
- ToolBar
- StatusBar

El manejo de estas partes se realiza mediante señales y slots que se activan cuando hay un trigger (disparo) o evento. Una señal es una notificación que un objeto emite cuándo cambia su estado, de manera que podría interesarle a otros objetos. Un slot es una función que se ejecuta cuándo una señal se emite.

Cualquier objeto en el que deseemos implementar señales y slots debe de heredar de la clase QObject. En el entorno los widgets que se utilizan cumplen con esta condición ya que heredan de QWidget que a su vez hereda de QObject. Los widgets que nos proporciona Qt poseen señales y slots predefinidos, pero también es posible crear un widget personalizado con el fin de añadir nuestras propias señales y slots.

Los slots utilizados en esta clase son los siguientes:

- void refresco_timer();
- void refresco_statusbar();
- void on_actionAcerca_de_triggered
- void on_actionModulo_triggered()
- void on_actionLinea_triggered(bool checked)
- void on_actionLimpiar_triggered()
- void on_actionBorrar_Modulo_triggered(bool checked)
- void on_actionFichero_de_informaci_n_triggered()
- void on_actionGuardar_salida_triggered(bool checked)
- void on_actionGuardar_proyecto_triggered()
- void on_actionInfo_Imagen_triggered(bool checked)
- void on_actionNuevo_triggered()
- void on_actionAbrir_triggered()

Todos ellos serán explicados con detenimiento en los capítulos siguientes.

Los triggers o eventos se pueden crear o generar de manera externa o interna.

Los eventos internos que se generan en esta clase son creados por dos timers (temporizadores). Estos timers están programados de manera que cuando llegan a su final de cuenta generan un evento. Cada evento que genera un timer, está conectado a slots diferentes, y esto sirve para que cada final de cuenta de los timers se ejecuten los slots asociados a cada timer. En resumen, cada timer está realizando un refresco de información cada fin de cuenta.

Los eventos externos los crea el usuario al utilizar el mouse (ratón) con el entorno gráfico. En esta clase solo se generan los eventos externos para las siguientes partes:

- MenuBar
- ToolBar

En la clase MainWindow solo hay una función, que es void cleararray() y sirve para poner a cero toda la información del entorno gráfico.

Esta información, se guarda por medio del uso de la clase QVector, que pertenece a la biblioteca propia de Qt. La explicación de la clase y su uso en el entorno se redactará en el capítulo de la clase Handler.

El CentralWidget de la UI de la clase MainWindow va a ser la base para las clases DragWidget y Linea. La función de estas clases se explica en los capítulos siguientes. En la figura 13 se puede ver la UI de la clase MainWindow y de que está compuesta:

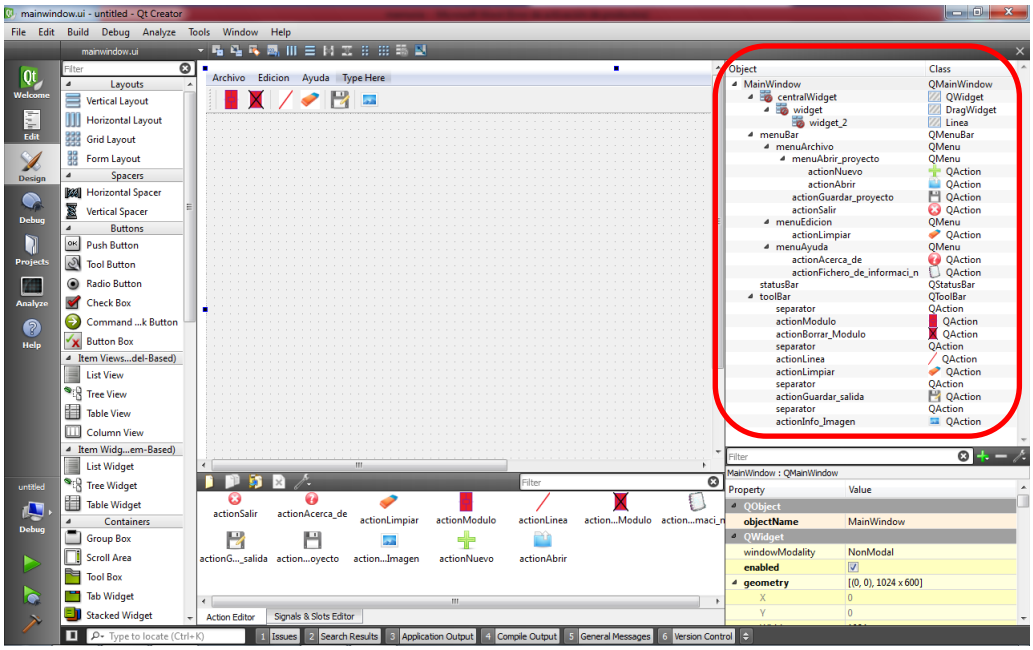


Figura 13. *UI (User Interface) de la clase MainWindow.*

3.3.3. DragWidget

La clase DragWidget es la clase que se encarga de la controlar la interacción del usuario con los módulos o bloques. Su UI (User Interface) tiene el propósito de controlar las posiciones, las conexiones y los modos que puede tener un módulo o varios de ellos.

El control sobre estos módulos se consigue gracias a los eventos que se crean con el ratón al cliquear sobre la UI de la clase. Hay cuatro funciones que manejan dichos eventos y son cruciales para que la información se guarde en el momento justo.

Dichas funciones son las siguientes:

- void *dragEnterEvent*(QDragEnterEvent *event)
- void *dragMoveEvent*(QDragMoveEvent *event)
- void *dropEvent*(QDropEvent *event)
- void *mousePressEvent*(QMouseEvent *event)

Para utilizar estas funciones, es necesario activar los Drops en la UI de la clase. Esto quiere decir que la UI permite utilizar el método arrastrar y soltar (drag and drop).

Este método es una expresión informática que se refiere a la acción de mover con el ratón objetos de una ventana a otra o entre partes de una misma ventana. Los objetos arrastrados en la UI de la clase DragWidget son de tipo Pixmap (mapa de pixeles). Estos objetos son los módulos, que se posicionan al antojo del usuario.

La secuencia de las funciones citadas anteriormente es:

1. Presionar con el mouse sobre el módulo (*mousePressEvent)*.
2. Mantener el botón del mouse u otro dispositivo apuntador, para "tomar" el módulo (*dragEnterEvent*).
3. ''Arrastrar'' el módulo a la ubicación deseada (*dragMoveEvent*).
4. ''Soltar'' el módulo soltando el botón del mouse (*dropEvent*).

Esta secuencia solo puede realizar con un módulo, es decir, el usuario no puede mover varios módulos debido a que el entorno gráfico OpenCV no tiene esa función. Cuando el usuario quiere realizar la secuencia ''Arrastrar y soltar'' para mover un módulo, éste no se modifica de posición hasta que el usuario no suelta el botón del ratón, es decir, hasta que no se produce el paso 4 de la secuencia.

Mientras que el usuario está en el paso 3 de la secuencia, se producen dos aspectos importantes:

- En el módulo de la posición inicial se crea una máscara grisácea para indicar cuál es el módulo a mover.
- Se crea una copia del módulo de la posición inicial, la cual, se mueve con el puntero del ratón.

Estos dos aspectos se van a cumplir mientras que el usuario no dé lugar al paso 4. Se produce el borrado de la UI del objeto en la posición inicial (módulo en la posición inicial) y donde se crea el nuevo objeto tipo Pixmap (nuevo módulo) en la nueva posición dada por el puntero del ratón. En la figura 14 se puede observar la secuencia del paso 3 (imagen de la izquierda) al paso 4 (imagen de la derecha).

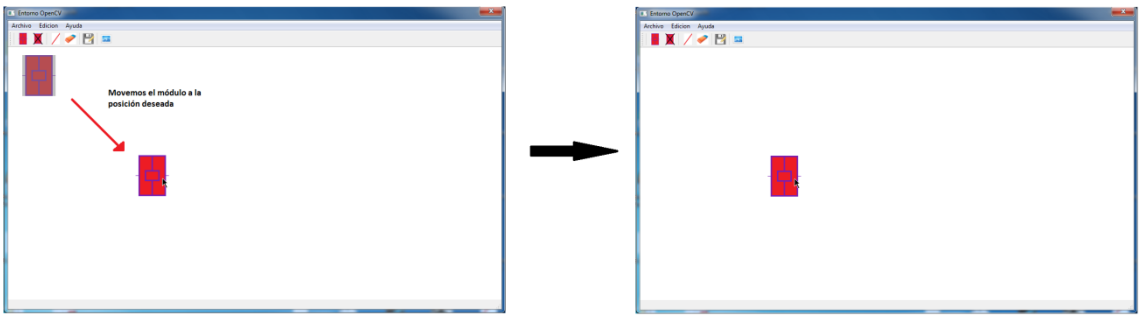


Figura 14. *Clase DragWidget: Transición del paso 3 al paso 4.*

Se puede observar que se utiliza un único módulo. En el caso de que el propio módulo a mover estuviera unido a uno o varios módulos, dichas uniones se rectificarían de manera que las líneas que unen a dichos módulos se vuelven a redibujar. En la figura 15 se puede ver dicho proceso:

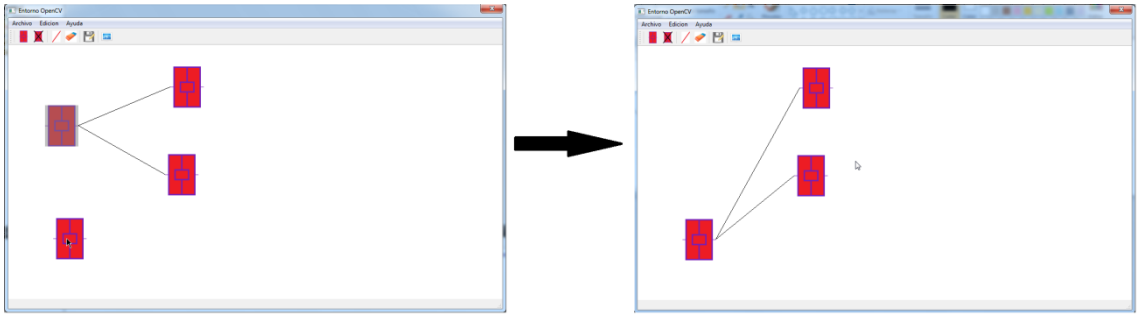


Figura 15. *Clase DragWidget: Transición del paso 3 al paso 4 con módulos unidos.*

Para entender de manera más precisa la secuencia, en la figura 16 se muestra un diagrama de flujo:

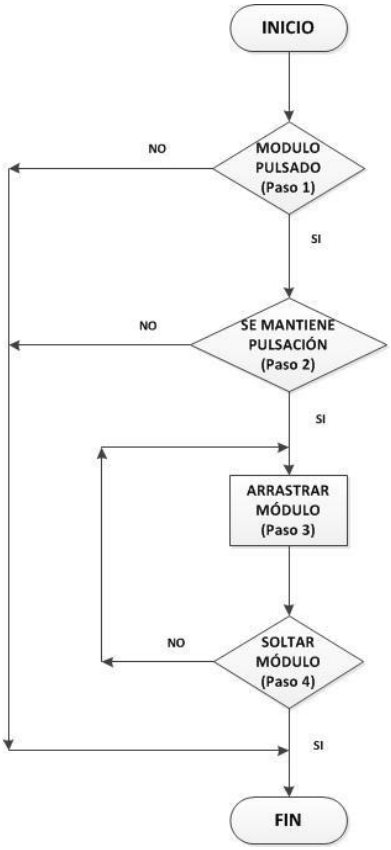


Figura 16. *Esquema de la secuencia "arrastrar y soltar".*

De las funciones citadas anteriormente las más importantes son *mousePressEvent y dropEvent.* En estas funciones es donde se guarda, se modifica y se borra toda la información del entorno.

La UI de esta clase solo posee CentralWidget a diferencia de MainWindow que está formada por otras tres más (StatusBar, MenuBar, ToolBar). Esto se debe a que la UI de esta clase no hereda de QMainWindow sino de QWidget.

Esto provoca que no disponga de las partes que posee la clase MainWindow, es decir, la UI de la clase DragWidget es un simple widget. Para entender esta explicación, en la figura 17 se puede ver la UI de la clase DragWidget donde sale recuadrado la composición de la misma:

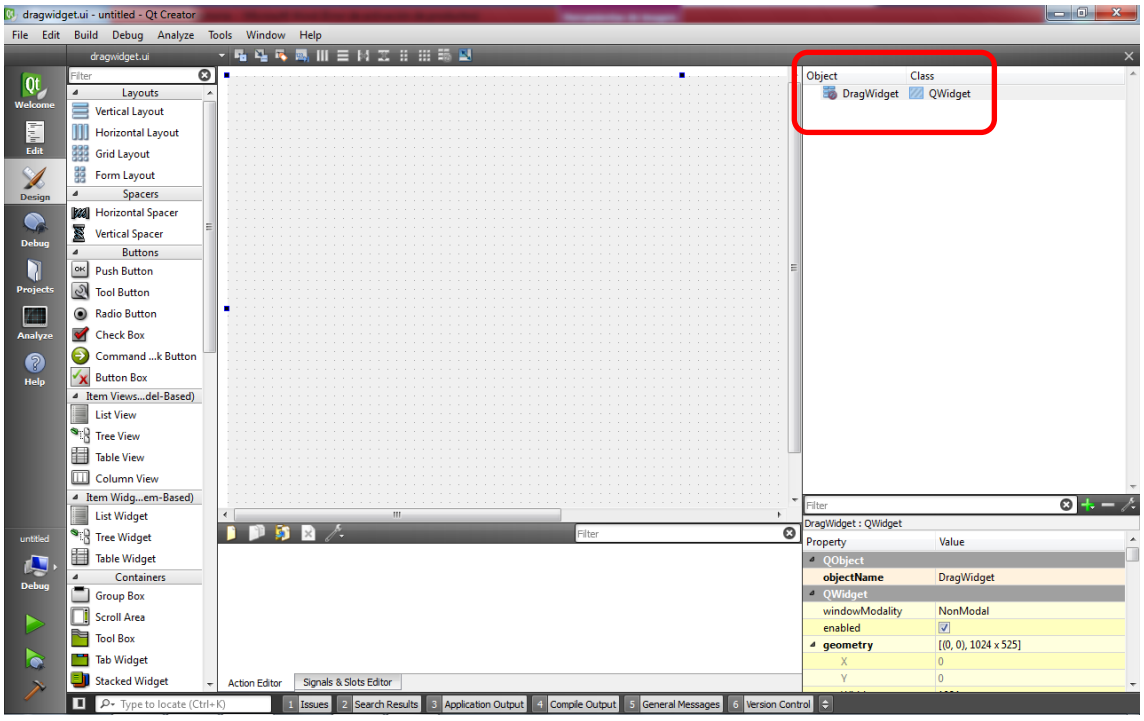


Figura 17. UI (User Interface) de la clase DragWidget.

3.3.4. Linea

Esta clase es la más simple de las cuatro que tiene el entorno gráfico OpenCV. El propósito de la clase Linea es dibujar las conexiones entre los módulos. Esta es la tercera de las tres clases que poseen UI.

La diferencia de la UI de esta clase y las UIs de la clase MainWindow y DragWidget, es que no hereda nada de la clase QWidget, sino que hereda todo de la clase QLabel.

Esto se debe a que para realizar el propósito descrito, se utiliza un objeto de tipo QImage cuyo padre es QLabel, es decir, la clase QImage hereda todo de QLabel.

En el objeto QImage se realiza las conexiones de manera que este actúa como un lienzo blanco en el cual se dibujan líneas. Dicho lienzo actúa como fondo.

La UI de la clase Linea tiene unas dimensiones de 1024x525, exactamente iguales que las dimensiones de la UI de DragWidget.

Para pintar las líneas se usa la clase QPainter. Esta clase realiza "pintura" de bajo nivel en los widgets y objetos de tipo QImage y QPixmap.

QPainter proporciona funciones altamente optimizadas para hacer la mayor parte de los programas de interfaz gráfica de dibujo requeridos. Se puede pintar desde líneas simples a formas complejas. También puede dibujar texto alineado.

El proceso de pintado de la línea se realiza utilizando uno de los dos objetos tipo QTimer que posee la clase MainWindow. Cuando el temporizador llega a final de cuenta, se pinta sobre un objeto tipo QImage todas las líneas almacenas y este objeto se establece como fondo en la UI de la clase Linea. Para entender mejor el proceso, se muestra un esquema del proceso en la figura 18:

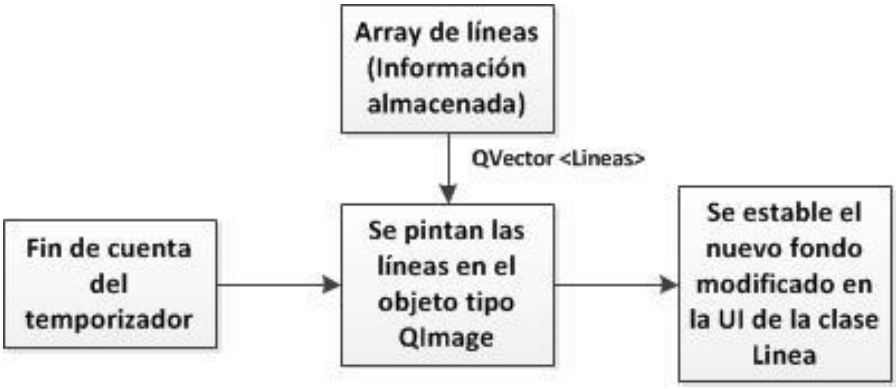


Figura 18. Esquema del proceso pintar líneas.

La UI de la clase Linea es igual que la de la clase DragWidget, es decir, hereda todo de QWidget, por lo que es un widget que utiliza la clase MainWindow. La composición de la UI de la clase Linea se puede ver en la figura 19:

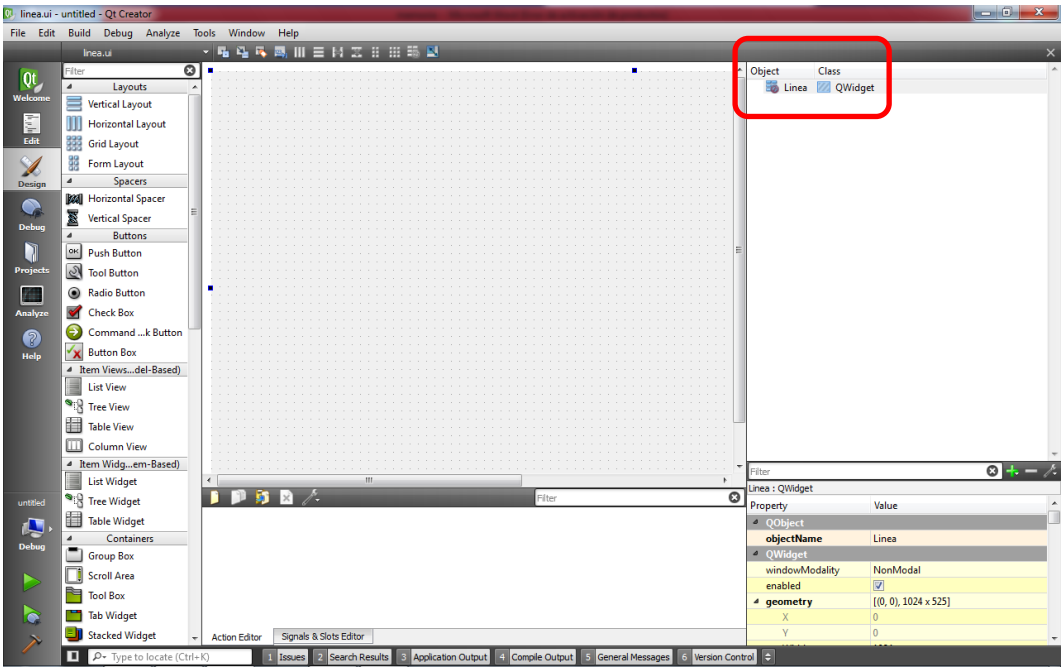


Figura 19. ***Esquema del proceso pintar líneas.***

3.4. Funcionalidades de las partes del entorno

Como se ha explicado en los capítulos anteriores, este entorno está formado por partes que son controladas por la clase MainWindow.

Dichas partes tienen una función concreta que cumplir para que el usuario pueda manejar dicho entorno en sus diferentes proyectos. En este capítulo se explicará de manera detallada las funciones que tiene o realiza cada parte que forma el entorno Gráfico OpenCV.

3.4.1. CentralWidget

En el CentralWidget es donde se va a desarrollar casi toda la interacción entre el usuario y el entorno. Esto se debe a que la zona donde se van a realizar los diferentes tratamientos a las imágenes es el CentralWidget.

Como se ha visto en los capítulos anteriores, la clase MainWindow es la que controla toda la ventana del entorno, por lo tanto también controla el CentralWidget del mismo. Para controlarlo se van a anexionar las UIs de las clases DragWidget y Linea.

Con esto conseguimos juntar las tres clases en el mismo CentralWidget y que cada una de las UIs realice la función para la cual está programada.

Por lo tanto, el CentralWidget del entorno va a tener diferentes capas, las cuales están controladas por diferentes clases y las capas que forman el mismo pertenecen a las siguientes clases:

- MainWindow.
- DragWidget.
- Linea.

Todo este proceso se debe a que las clases se tienen que sincronizar entre ellas para que el usuario pueda realizar lo que desee. La escala de nivel de control de las capas se muestra en la figura 20:

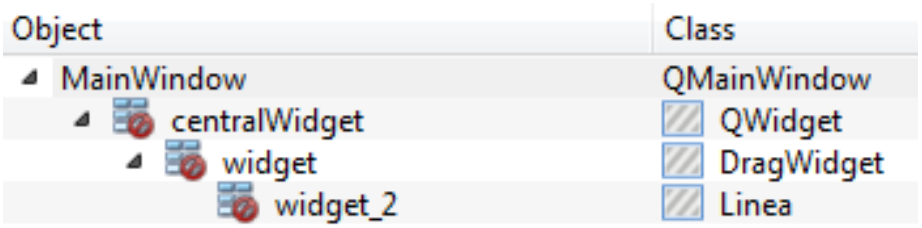


Figura 20. *Composición del CentralWidget.*

Una vez vista la figura 20, se puede ver que el CentralWidget de la clase MainWindow hereda todo de la clase QWidget, y dicho CentralWidget está formado por otros dos Widget que pertenecen a las clases DragWidget y Linea.

3.4.2. MenuBar

Esta parte del entorno no tiene mucha interacción con el usuario, ya que probablemente será la parte del entorno que menos utilice el usuario. En cambio, la función de la misma es muy importante ya que incorpora los siguientes slots:

- Guardar Proyecto
- Nuevo Proyecto
- Abrir Proyecto
- Salir
- Limpiar
- Generar fichero
- Acerca de…

De todas estas funciones, las más importantes son Abrir proyecto y Guardar Proyecto. Estas funciones son las más complejas de todas las anteriores y las que tienen que realizar su función de manera correcta debido a que sino siempre se puede provocará un ‘’bug” (error de software) en el entorno.

3.4.2.1. Guardar Proyecto

Esta función (o slot) es esencial ya que para que el usuario pueda cargar en un futuro un proyecto, se necesitan que todos los datos estén guardados de alguna forma para luego más tarde poder volver a usarlos.

La información que se necesita guardar se hará mediante el uso de un fichero de texto (.txt). En este fichero se guardará toda la información de los cuatro vectores que almacenan toda la información del entorno. El alumno ha decidido guardar la información de esta manera debido a que si hubiera algún bug en el guardado de la información, se podría ver con más claridad.

El formato de guardado creado en el fichero se realiza de manera ordenada. El orden de guardado establecido es el siguiente:

1. Líneas.
2. Módulos.
3. Conexionado de módulos.
4. Imágenes.

Una vez que se ha escrito toda la información en el fichero de texto se procede al guardado de los archivos necesarios para en un futuro poder cargarlos. Estos archivos son las imágenes y se guardan en la misma ruta que el archivo de texto, por lo tanto, cuando el usuario realiza un guardado se crea un fichero y las imágenes usadas en dicho fichero.

Este guardado se puede realizar en la ruta que el usuario quiera. Un ejemplo de guardado se puede observar en la figura 21:

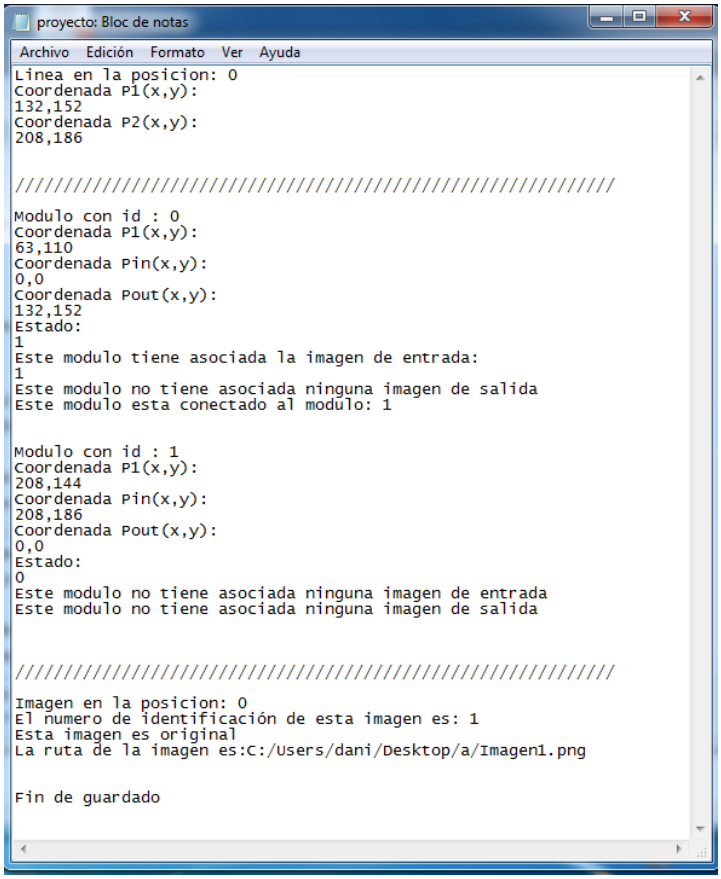


Figura 21. ***Fichero de texto de guardado.***

Se puede observar el título que el usuario ha elegido para su proyecto, en este caso, es ''proyecto''. También se puede ver que hay dos módulos, una imagen y una línea que conecta a dichos módulos. Dicha línea une la salida del módulo con identificador 0 con la entrada del módulo con identificador 1. Además se puede observar que las coordenadas de los puntos de la línea coinciden con los puntos de entrada (Pin) y salida (Pout) de los módulos.

En cuanto a la imagen se pueden observar diferentes datos, como la imagen de entrada del módulo con identificador 0, que es una imagen original, es decir, no se la ha realizado ningún tratamiento, su número de identificación y también se puede observar la ruta donde se encuentra. En la ruta se puede ver que el nombre que se ha dado es ''Imagen1''. Esto se debe a que el alumno ha implementado esta función de manera que cuando se realice el guardado, cada imagen tendrá el nombre de ''Imagen" más su número de identificación, por ejemplo si hubiera una imagen con el número de identificación 6, su nombre en este proceso seria "Imagen6". El formato elegido para el guardado de dichas imágenes es png (Portable Network Graphics).

3.4.2.2. Nuevo Proyecto

Esta función sirve para crear un nuevo proyecto, con ella el usuario puede estar realizando un trabajo, y si en un momento dado necesita reiniciar o empezar desde cero, puede utilizar esta función. Para conseguir esto, se necesita que la información del entorno se quede a cero y para ello se borrar los cuatro vectores (QVector) que posee la clase Handler.

Los vectores que se borran primero son el de imágenes (listaimagenes) y el de conexiones entre módulos (matriz). Una vez realizado el borrado de información de dichos vectores se necesita borrar la información gráfica del entorno, es decir, nos queda borrar los módulos y las líneas que los unen que el usuario está visualizando. En el caso de las líneas se borran automáticamente una vez borrado la información de su vector (QVector <Lineas> array[100]) debido a que como se ha explicado en los capítulos anteriores, la clase MainWindow tiene un timer que realiza el refresco de pantalla con la información que contenga dicho vector.

En el caso de los módulos se necesita saber el valor del parámetro P1 de la estructura de los Modulos debido a que dicho parámetro es el punto de la posición del módulo. Una vez borrados los módulos ya está el entorno para un nuevo uso del usuario.

3.4.2.3. Abrir Proyecto

Para usar esta función antes se debe disponer de los archivos necesarios que se crean cuando se usa la función de Guardar Proyecto, es decir, si no se ha realizado ningún guardado anteriormente no servirá de nada esta función.

La función Abrir Proyecto realiza una tarea sencilla pero muy precisa. La precisión se debe a que para cargar los datos, dicha función lee línea a línea el fichero de texto que contiene toda la información del proyecto que guardo con anterioridad el usuario. El orden de leído del fichero es el mismo que el orden utilizado en la función Guardar Proyecto.

Cuando se empieza a leer el fichero de texto línea a línea se utilizan diferentes ''identificadores'' para que la información no se lea de manera incorrecta. Estos identificadores son la estructura del fichero de texto. En la figura 22 se puede ver recuadrado los diferentes ''identificadores'' del ejemplo utilizado en la figura 21:

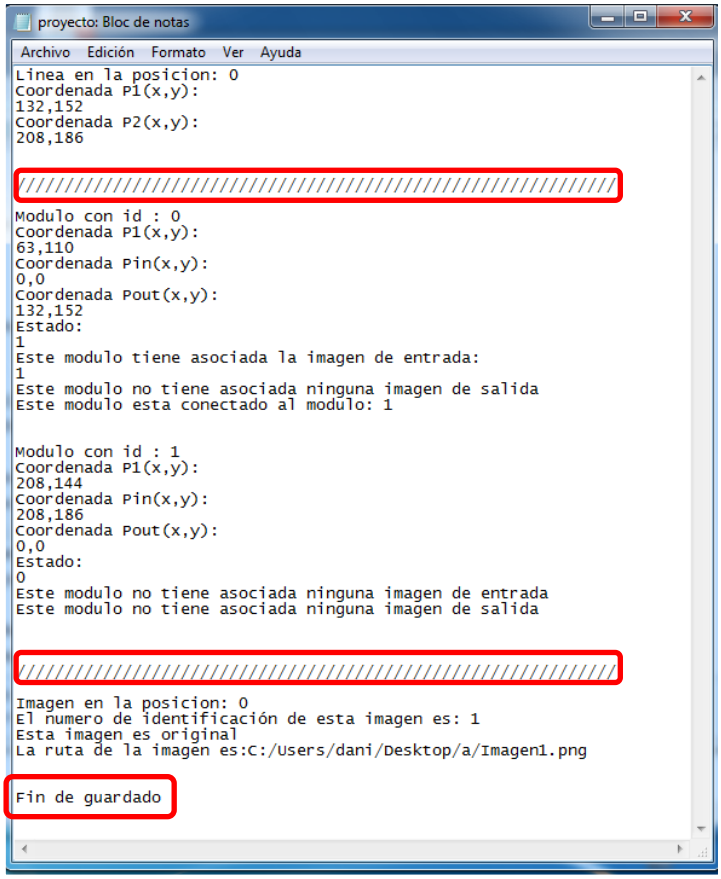


Figura 22. Identificadores del Fichero de texto.

Cuando el usuario quiere abrir un proyecto guardado con anterioridad se pueden dar dos casos:

1. El usuario no ha realizado ningún proceso todavía.
2. El usuario está realizando en ese preciso momento otro proyecto.

Para cualquiera de los dos casos, se realiza primero una puesta a cero del entorno y a continuación la carga de información. En el primer caso simplemente carga la información contenida en el fichero, pero en el segundo caso se produce la puesta a cero del entorno. Por lo que el usuario tiene que ser consciente de que perderá toda la información del proyecto que esté realizando en ese momento.

3.4.2.4. Salir

La función Salir es muy simple, su propósito es únicamente cerrar el entorno gráfico OpenCV. Para ello, cuando el usuario pulsa sobre esta opción de la MenuBar provoca el cierre de la UI de la clase MainWindow, y con ello el cierre del entorno. Es equivalente a pulsar sobre el botón de cerrar de la ventana.

3.4.2.5. Limpiar

El propósito de esta función es borrar todas las conexiones entre todos los módulos que el usuario tenga en el entorno. Para ello, se borran completamente los vectores siguientes:

- static QVector <Lineas> array[100];
- static QVector <bool> matriz[100][100];

En el vector de los módulos (static QVector <Modulos> arrayModulos [100]) solo se borran los parámetros Pin y Pout de cada uno de los módulos que existan, ya que son los puntos de entrada y salida de conexionado entre módulos.

El último vector que queda es el que contiene la información de las imágenes (static QVector <Imagenes> listaimagenes). En este vector no se modifica o elimina ningún tipo de información, excepto en el caso de que no exista ningún módulo en el entorno, es decir, en el caso de que el usuario no tenga ningún módulo visible en la interfaz del entorno. En este caso expuesto, se borraría el vector de las imágenes.

3.4.2.6. Generar Fichero

Esta opción de la MenuBar es para que el usuario pueda ver toda la información que se ha almacenado en los diferentes vectores. Puede ser útil para saber los diferentes parámetros de las estructuras. Esta información contenida en los vectores, se transcribe a un fichero de texto (.txt) al igual que en la función de Guardar Proyecto.

Entre la función Guardar Proyecto y la función Generar fichero hay una pequeña diferencia. Ésta consiste en que en la parte de la información de las imágenes del fichero generado, el parámetro de la ruta de la imagen, puede aparecer vacío.

Esto se debe a que si el usuario no ha realizado ningún guardado o cargado de proyecto, no se puede conocer la ruta de las imágenes, ya que la rutas de la imágenes solo se crea cuando se realizan dichos procesos.

En la figura 23 se puede ver el uso de esta función para el mismo ejemplo de los capítulos anteriores pero sin cargar o guardar proyecto:

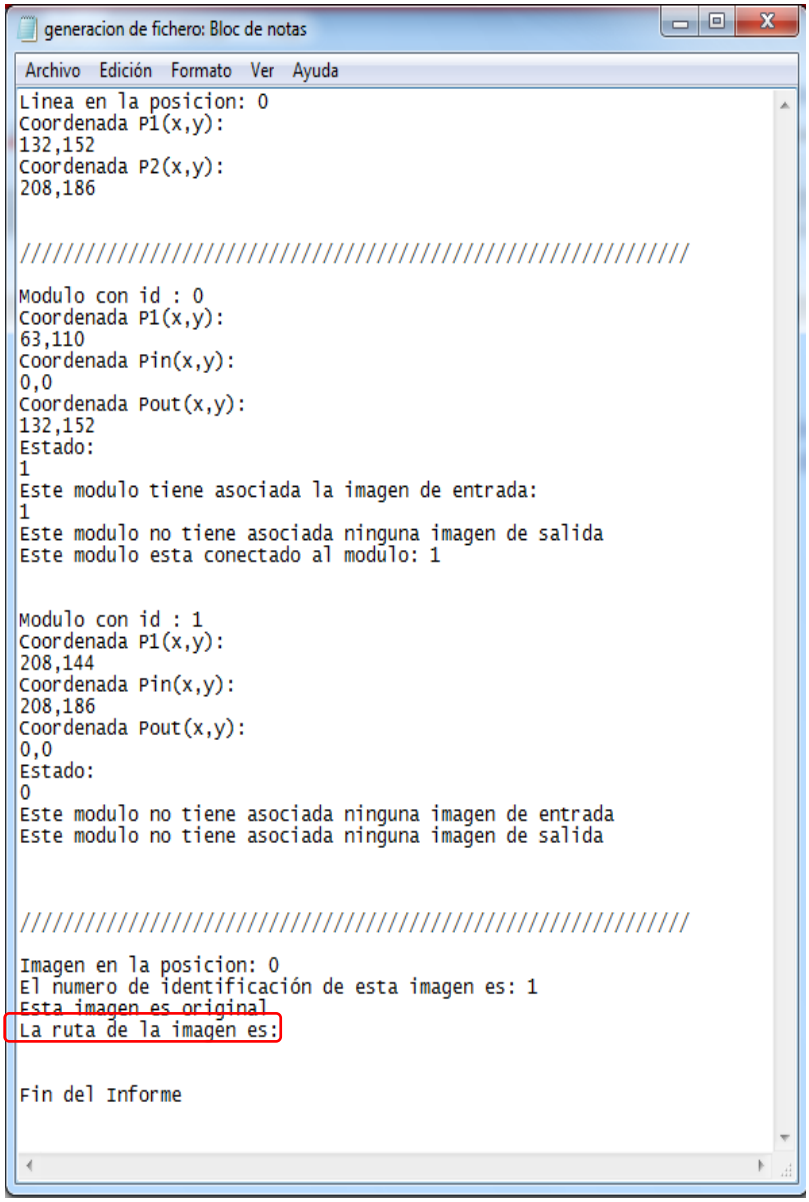


Figura 23. *Generación de Fichero de texto.*

3.4.2.7. Acerca de…

En esta función solo se visualiza información referente al alumno y al entorno. Se puede ver en la figura 24 la ventana que se muestra una vez que el usuario elige esta opción de la MenuBar:

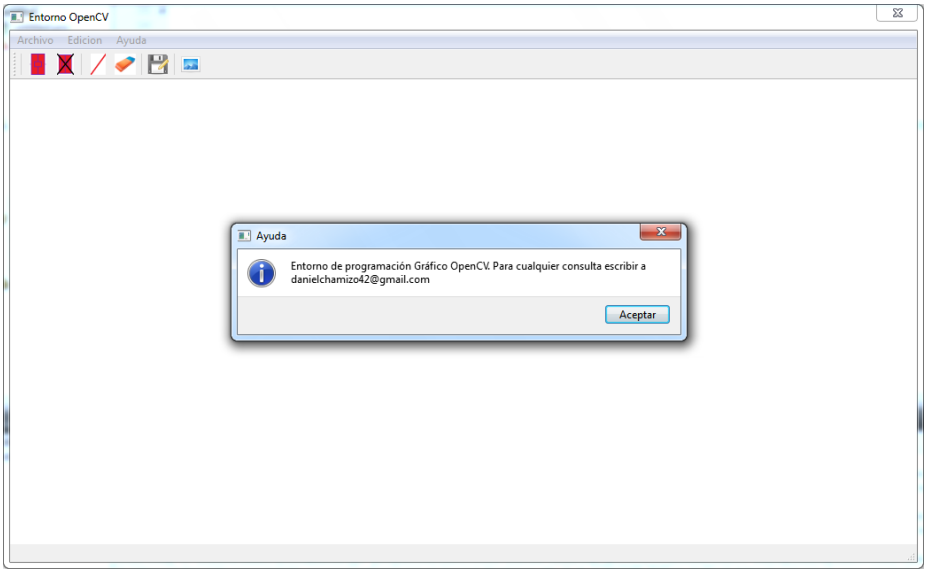


Figura 24. *Acerca de…*

3.4.3. ToolBar

En esta parte del entorno es donde el usuario utiliza todas las herramientas que el entorno le proporciona. Dichas herramientas son unos botones que el usuario puede pulsar con el ratón. Se clasifican por zonas y cada una de ellas está separada por una barra horizontal llamada separador. Estas zonas son las siguientes:

- Zona de acción de módulos.
- Zona de conexionado.
- Zona de guardado de imagen.
- Zona de información de imagen.

En la figura 25 se pueden distinguir dichas zonas:

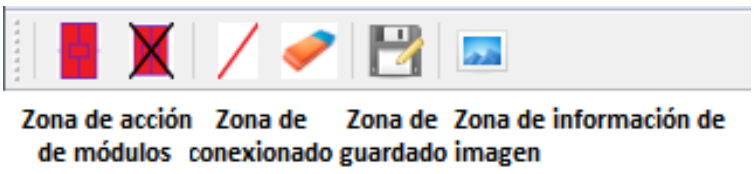


Figura 25. *ToolBar.*

3.4.3.1. Zona de acción de módulos

En esta zona se encuentran dos herramientas que realizan acciones referidas al módulo. Una realiza la acción contraria de la otra. Dichas acciones son:

- Crear módulo
- Borrar módulo

Cuando el usuario pulsa sobre la acción crear módulo provoca que el entorno sitúe en una posición inicial un módulo. La posición de creación es siempre la misma ya que así lo ha decidido el alumno, y no se puede crear un módulo nuevo si está ocupada dicha posición. Si se da este caso, el entorno avisa al usuario mediante una ventana de información. Este caso se observar en la figura 26:

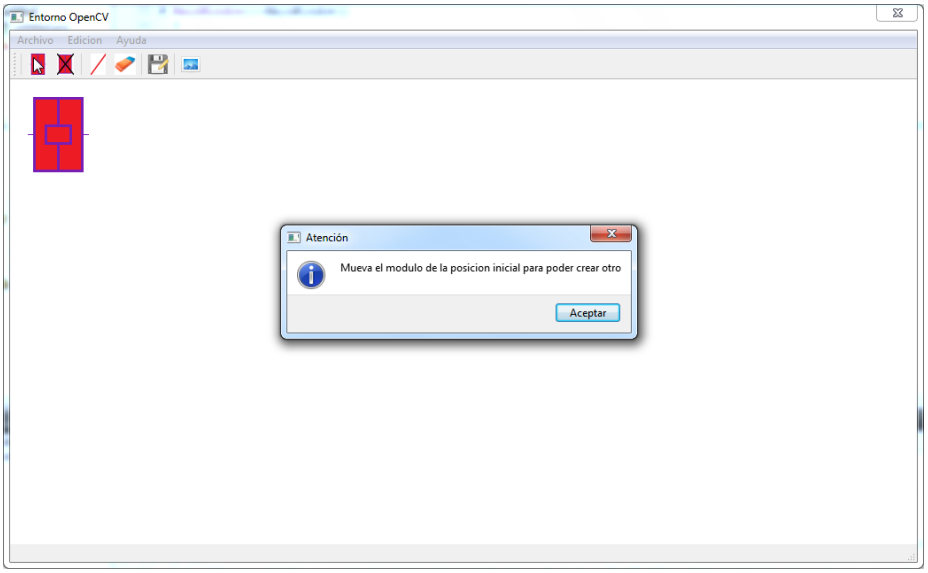


Figura 26. ***Caso crear módulo.***

En este caso, se puede observar el puntero del ratón encima del botón de creación de módulo donde el usuario ha intentado crear otro módulo, teniendo otro creado en la posición de creación inicial. Esta implementación que ha diseñado el alumno, obliga al usuario a retirar el módulo creado de la posición inicial para poder crear otro.

Esto se debe a que la intención del alumno es evitar la creación infinita de módulos provocando una confusión o traspaso erróneo de la información de cada módulo.

En cuanto a la acción de Borrar módulo sirve para borrar el módulo que el usuario elija. El botón tiene dos modos:

1. Modo pulsado
2. Modo no pulsado

El modo pulsado se produce cuando se pulsa una vez el botón. Éste se queda como si se estuviera presionando sobre él continuamente. Cuando se vuelve a pulsar sobre el botón, éste vuelve a su estado normal. A continuación se muestra en la figura 27 los diferentes estados del botón Borrar módulo:

Figura 27. *Borrar módulo.*

Cuando el usuario quiere borrar un módulo tiene que pulsar el botón para que se quede en modo pulsado (ver figura 27) y elegir el módulo a eliminar. Mientras que el botón Borrar módulo esté en modo pulsado, el usuario puede eliminar todos los módulos sobre los que pulse, pero solo puede eliminar de uno en uno.

En el uso de esta herramienta (Borrar módulo) se pueden producir diferentes casos al eliminar un módulo, los cuales se citan a continuación:

1. El módulo no tiene ningún conexionado con uno o varios módulos.
2. El módulo tiene conexión solo en la salida.
3. El módulo tiene conexión solo en la entrada.
4. El modulo tiene conexión en la entrada y salida.

En el caso 1, el módulo simplemente se elimina. En el caso 2, se elimina el modulo sobre el que pulse el usuario y los módulos conectados a la salida de dicho modulo eliminado se reinician, borrando todos los datos que poseen. Solamente se borran las conexiones de las salidas del módulo eliminado. En el caso 3, el modulo sobre el que pulse el usuario se elimina y la conexión con el modulo que esté conectado a su entrada se elimina también. En el caso 4, se produce una mezcla entre el caso 2 y el caso 3. Para entender mejor la explicación se muestran las figuras 28, 29, 30, 31 con los diferentes casos:

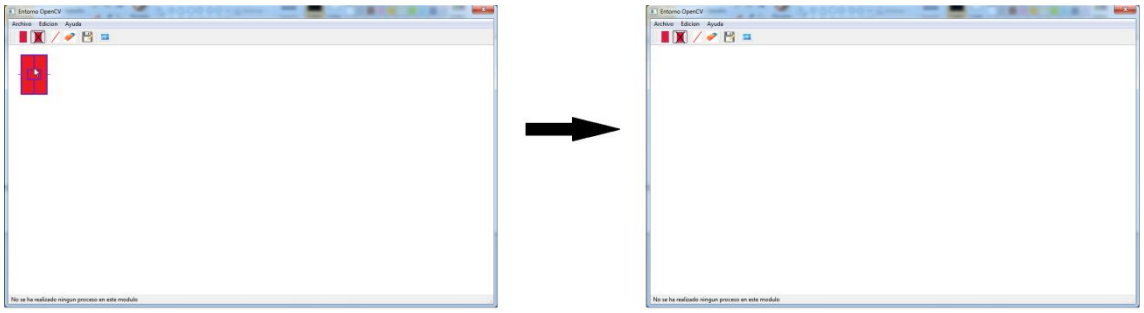


Figura 28. Borrar módulo: caso 1.

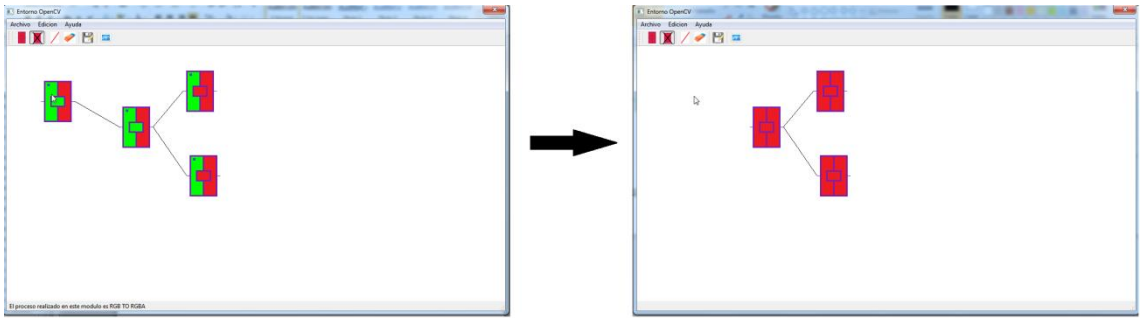


Figura 29. Borrar módulo: caso 2.

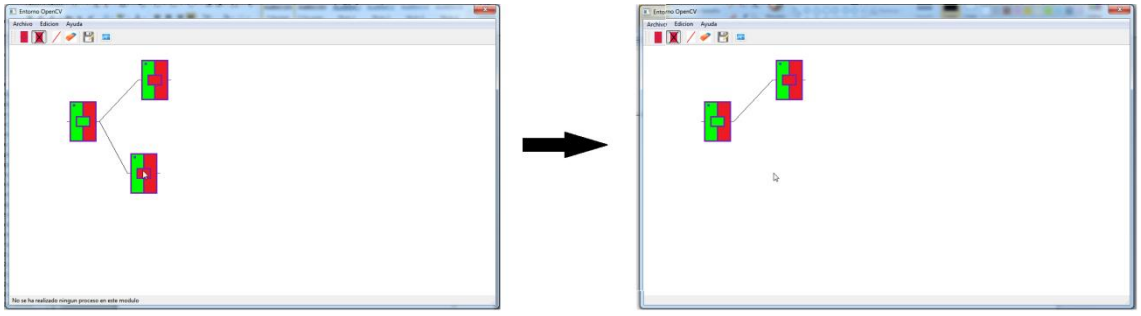


Figura 30. *Borrar módulo: caso 3.*

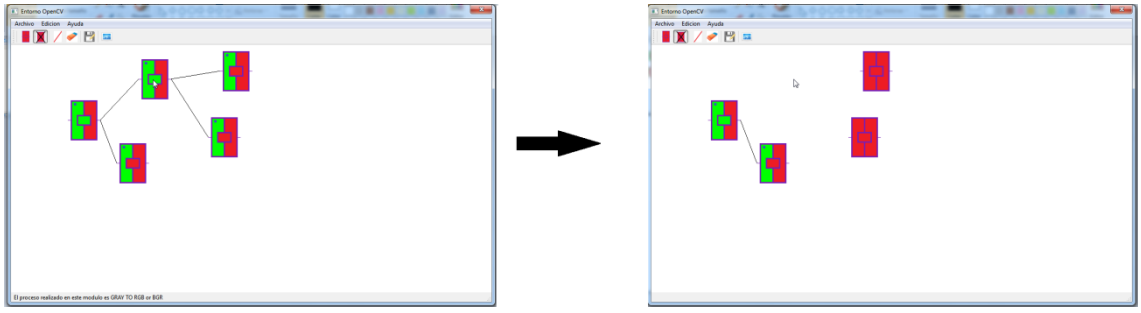


Figura 31. *Borrar módulo: caso 4.*

3.4.3.2. Zona de conexionado

En esta zona, el usuario puede conectar los diferentes módulos que quiera utilizar en el entorno gráfico OpenCV. También hay dos botones como en la zona de acción de módulos. Dichos botones tienen las siguientes funciones:

- Conectar módulos.
- Eliminar todas las conexiones existentes.

En cuanto a la acción de Conectar módulos, sirve realizar el conexionado entre los mismos. El botón tiene dos modos:

1. Modo pulsado
2. Modo no pulsado

El modo pulsado se produce cuando se pulsa una vez el botón. Éste se queda como si se estuviera presionando sobre él continuamente. Cuando se vuelve a pulsar sobre el botón, éste vuelve a su estado normal. En la figura 32 se puede observar los diferentes estados del botón Conectar módulos:

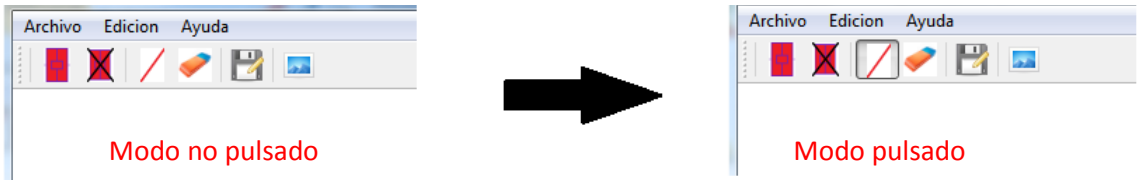


Figura 32. ***Conectar módulos.***

Cuando el botón de Conectar módulos está en modo pulsado, el usuario puede conectar los módulos que quiera pero siempre cumpliendo con las condiciones siguientes:

- No se puede conectar a un módulo cuya entrada ya tenga una imagen seleccionada o procesada.
- Cada módulo puede tener varias salidas pero solo puede tener una entrada.
- No se puede conectar la entrada con la salida de un mismo módulo ni viceversa.

Para conectar los módulos el usuario tiene que pulsar en la mitad correspondiente del módulo, es decir, si el usuario quiere conectar la salida del módulo 1 a la entrada del módulo 2, tiene que pulsar en la mitad derecha del módulo 1 y la mitad izquierda del módulo 2. En la figura 33 se puede observar las mitades de un módulo:

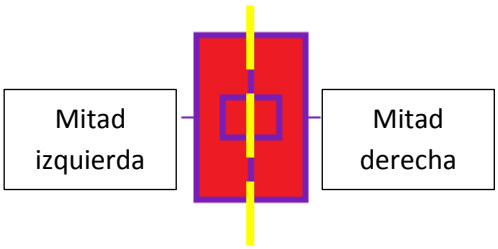


Figura 33. *Conexionado de módulos: mitades.*

El orden de pulsado en el ejemplo anterior es indiferente para el entorno, es decir, para el caso anterior da igual si el usuario pulsa primero sobre el módulo 2 y luego sobre el módulo 1 o viceversa. El entorno detecta cuál es el orden de pulsado y hace los cálculos de las posiciones correspondientes.

Una vez que el usuario ha conectado todos los módulos que quiere, tiene la opción de poder deshacer todas las conexiones existentes. Esta función es la que realiza el otro botón de la zona de conexionado. La opción de poder desconectar la conexión deseada no se puede realizar ya que el alumno lo considera una mejora futura. Para ver cómo funciona, se puede observar en la figura 34:

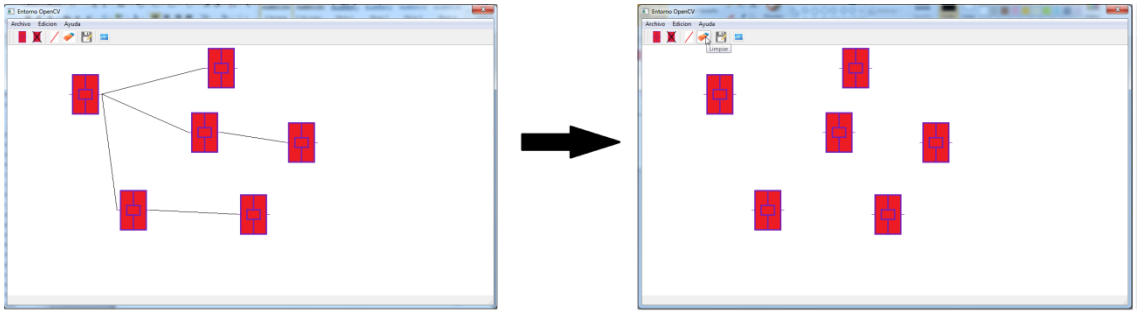


Figura 34. Conexionado de módulos: borrado de conexiones.

3.4.3.3. Zona de guardado de imagen

En esta zona solo tendremos un botón, que sirve para guardar la imagen que el usuario desee del módulo que quiera. El guardado de la imagen, se puede realizar en la ruta que estime el usuario y también se puede seleccionar que imagen guardar, la de entrada o la de salida.

En cuanto a la acción de Guardar Imagen, tiene dos modos al igual que los botones de Borrar módulos y Limpiar o Borrar conexionado de módulos:

1. Modo pulsado
2. Modo no pulsado

El modo pulsado se produce cuando se pulsa una vez el botón. Éste se queda como si se estuviera presionando sobre él continuamente. Cuando se vuelve a pulsar sobre el botón, éste vuelve a su estado normal. A continuación se muestra en la figura 35 los diferentes estados del botón Guardado de imagen:

Figura 35. *Conexionado de módulos: borrado de conexiones.*

Para usar dicha herramienta, el usuario solo tiene que dejar el botón en modo pulsado y a continuación pulsar sobre el modulo que desee. Cuando se pulsa sobre el módulo, el entorno le pide al usuario la ruta, y una vez elegida la ruta, le pide la imagen que contenga el módulo (entrada o salida).

Se pueden dar diferentes casos de aviso al usuario en esta herramienta:

1. Si el módulo no tiene ninguna imagen ni de entrada ni de salida asociada.
2. Si el módulo solo tiene una imagen de entrada asociada.
3. Si el módulo tiene imagen de entrada y salida asociada.

En el caso 1, el usuario no puede guardar ninguna imagen, por lo que si utiliza esta herramienta será avisado por el entorno.

En el caso 2, el usuario solo puede guardar la imagen de entrada, por lo que si intenta guardar la imagen de salida será avisado por el entorno.

En el caso 3, el usuario puede realizar el guardado de las imágenes sin ningún problema. A continuación se expone un ejemplo (figura 36) que se produce en los casos anteriores, concretamente el caso 2:

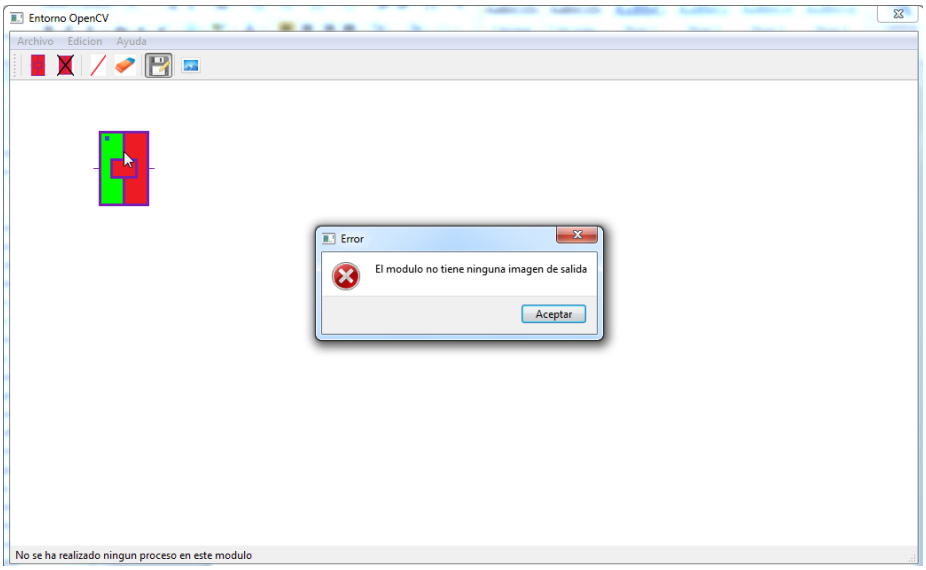


Figura 36. *Guardado de imagen: caso 2.*

3.4.3.4. Zona de información de imagen

Esta herramienta se utiliza para que el usuario pueda saber la información de la imagen que está contenida en la entrada o salida del módulo. Para ello esta zona está formada por un botón al igual que en la zona de Guardado de imagen. Los modos de esta herramienta son:

1. Modo pulsado
2. Modo no pulsado

El modo pulsado se produce cuando se pulsa una vez el botón. Éste se queda como si se estuviera presionando sobre él continuamente. Cuando se vuelve a pulsar sobre el botón, éste vuelve a su estado normal. En la figura 37 se muestra los diferentes estados del botón Información de imagen:

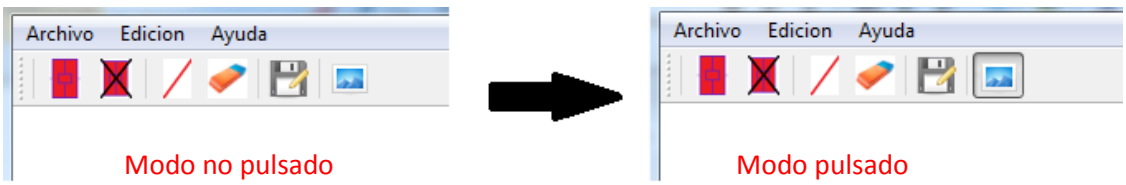


Figura 37. Información de imagen.

Cuando el usuario utiliza esta herramienta se pueden producir los siguientes casos:

1. El módulo seleccionado no tiene imagen de salida.
2. El módulo seleccionado no tiene imagen ni de entrada ni de salida.
3. El módulo seleccionado tiene imagen de entrada y salida.

En el caso 1, el usuario solo recibirá la información de la imagen de entrada. En el caso 2, el usuario no recibirá ningún tipo de información, ya que no hay imágenes. En el caso 3, el usuario recibirá la información tanto de la imagen de entrada como de la imagen de salida. A continuación se exponen los casos comentados en las figuras 38, 39, 40:

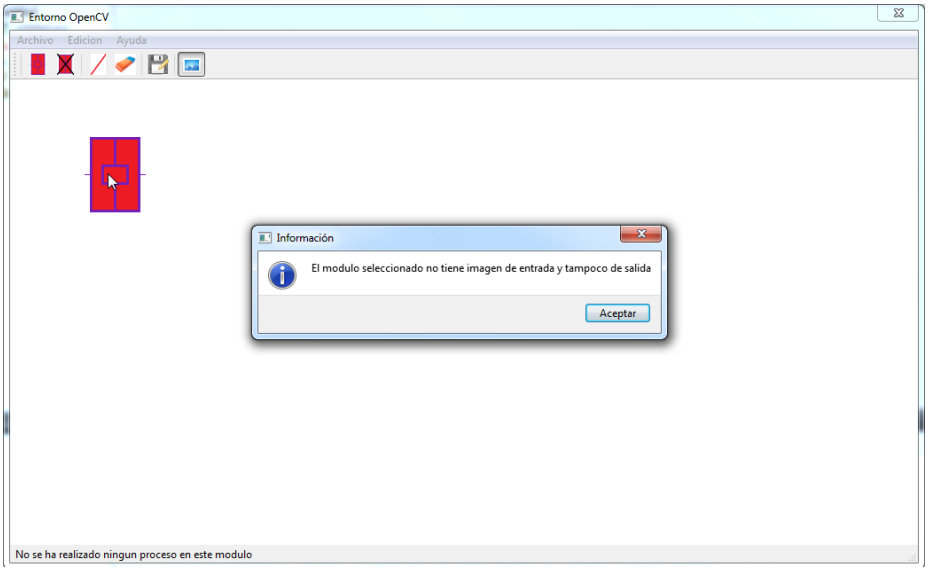


Figura 38. ***Información de imagen: caso 1.***

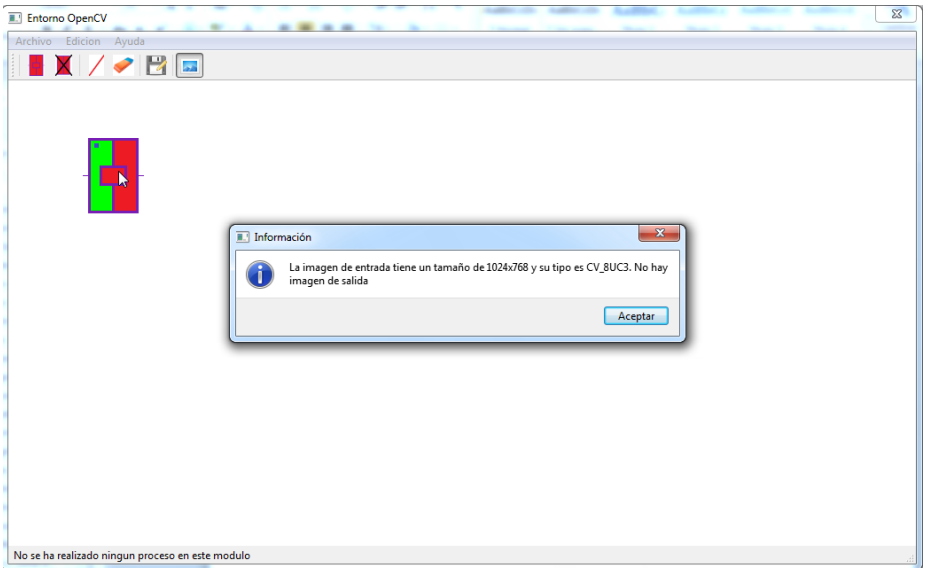


Figura 39. Información de imagen: caso 2.

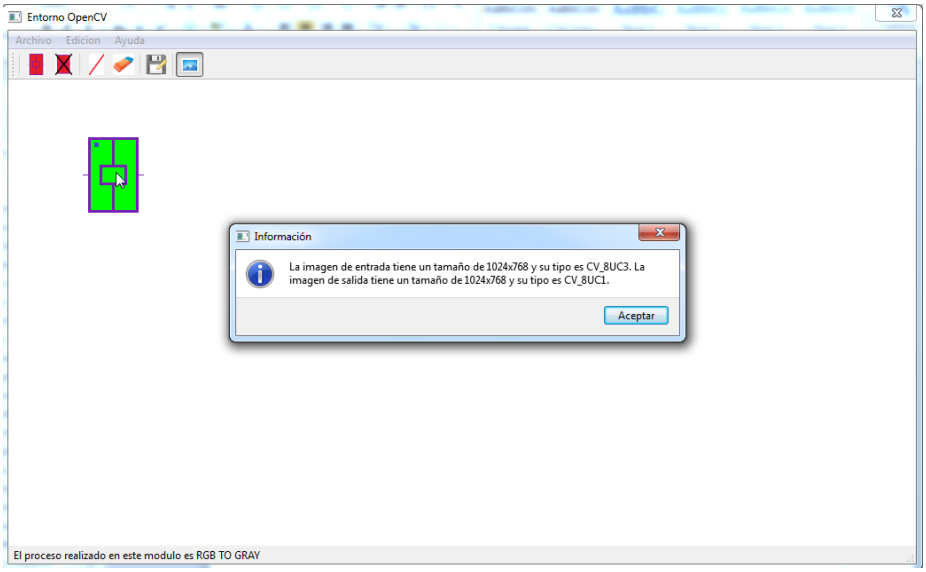


Figura 40. Información de imagen: caso 3.

3.4.4. StatusBar

En esta parte del entorno el usuario puede averiguar los diferentes procesos que va realizando a cada uno de los módulos que se encuentren en la interfaz del entorno. Para ello, el usuario solo tiene que situar el puntero del ratón dentro de los límites del módulo sin necesidad de hacer ningún click sobre el mismo. Para el diseño e implementación de esta parte del entorno se ha utilizado el otro timer que tiene la clase MainWindow, de manera que se está produciendo un continuo refresco de información.

La información que se puede visualizar en la StatusBar es el procesado realizado en el módulo. Para ello se utiliza la variable "proceso" que posee la estructura Modulos. Cuando el timer llega a su final de cuenta se produce un evento que se envía al siguiente slot:

- void refresco_statusbar()

Dentro de este slot se pide la posición del puntero del ratón y si se encuentra dentro de los límites del mapa de pixeles del módulo se busca su variable "proceso".

Este refresco de información depende de la posición del puntero del ratón, es decir, si el puntero está encima y dentro de los límites del mapa de pixeles de un módulo, se podrá visualizar la información del mismo, sino, no se verá nada en la StatusBar. Para entender el funcionamiento de esta parte se muestra un esquema en la figura 41:

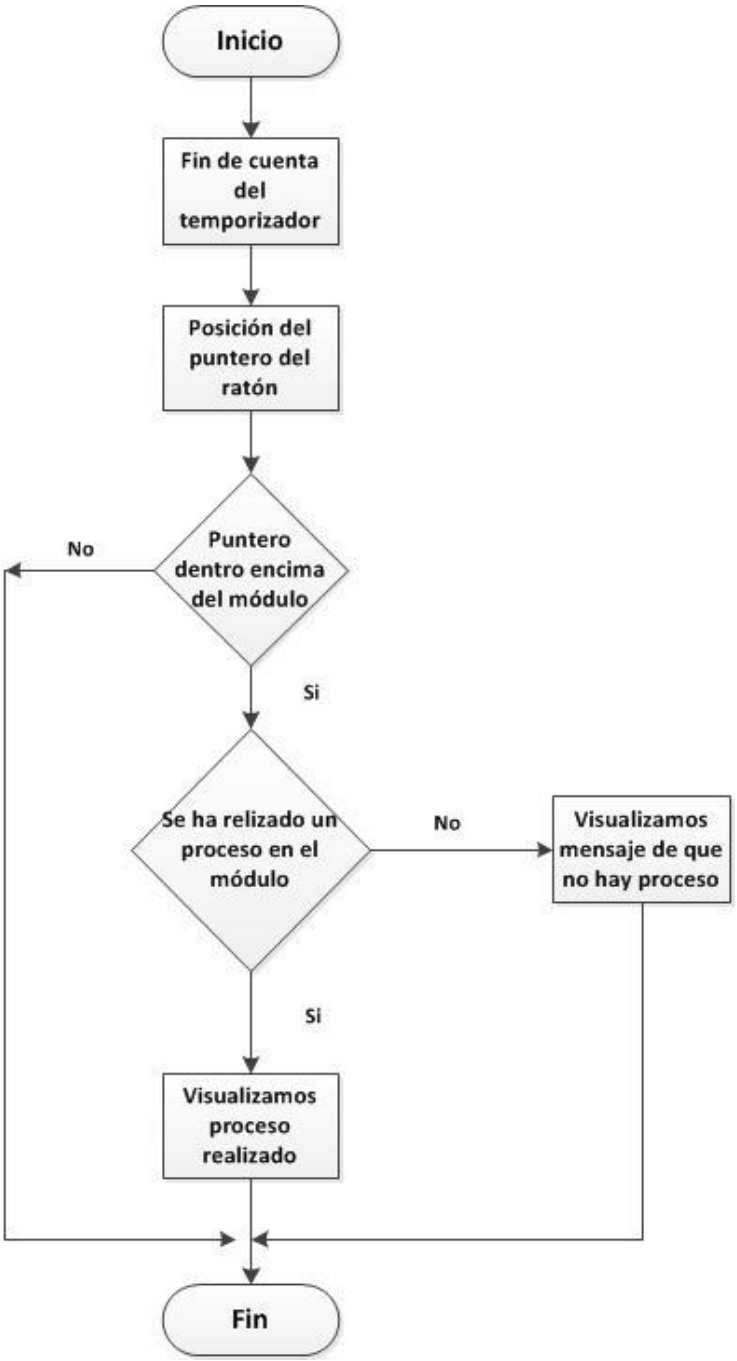


Figura 41. *Esquema de la StatusBar.*

Se puede observar en este esquema que en caso de que en el módulo no tenga ningún proceso realizado por el usuario, el entorno se lo indica. Una vez visto este esquema, se va a mostrar el resultado en el entorno en la figura 42:

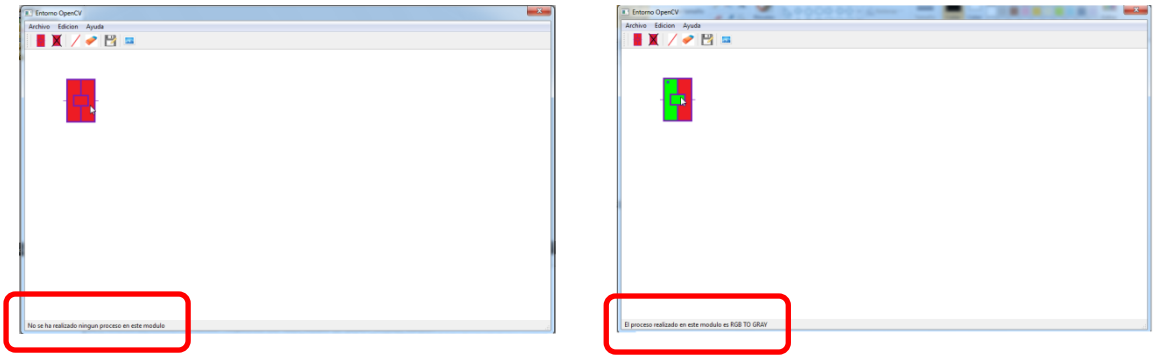


Figura 42. StatusBar.

3.5. Bloque o Módulo

El módulo o bloque es clave para que el usuario pueda utilizar todos los recursos disponibles que el entorno le ofrece. En el módulo se produce la coordinación de toda la información que proviene del usuario y la que el entorno crea para realizar lo que el usuario quiere. La información que proviene del usuario se capta mediante pequeños widgets que están formados a su vez por otros. En estos widgets es donde se produce la interacción más fuerte entre el usuario y el entorno.

Para una mejor compresión de las partes del módulo, se hará un pequeño inciso explicando primero la composición de dichos widgets. Están formados por las siguientes clases:

- QTableView
- QSpinBox
- QDoubleSpinBox
- QStandardItemModel
- QModelIndex
- QPushButton
- QComboBox
- QLineEdit
- QToolButton

Todas estas clases, son las que se han utilizado para crear widgets personalizados para cada una de las diferentes funciones que puede ofrecer el entorno. Un ejemplo de ello se muestra en la figura 43:

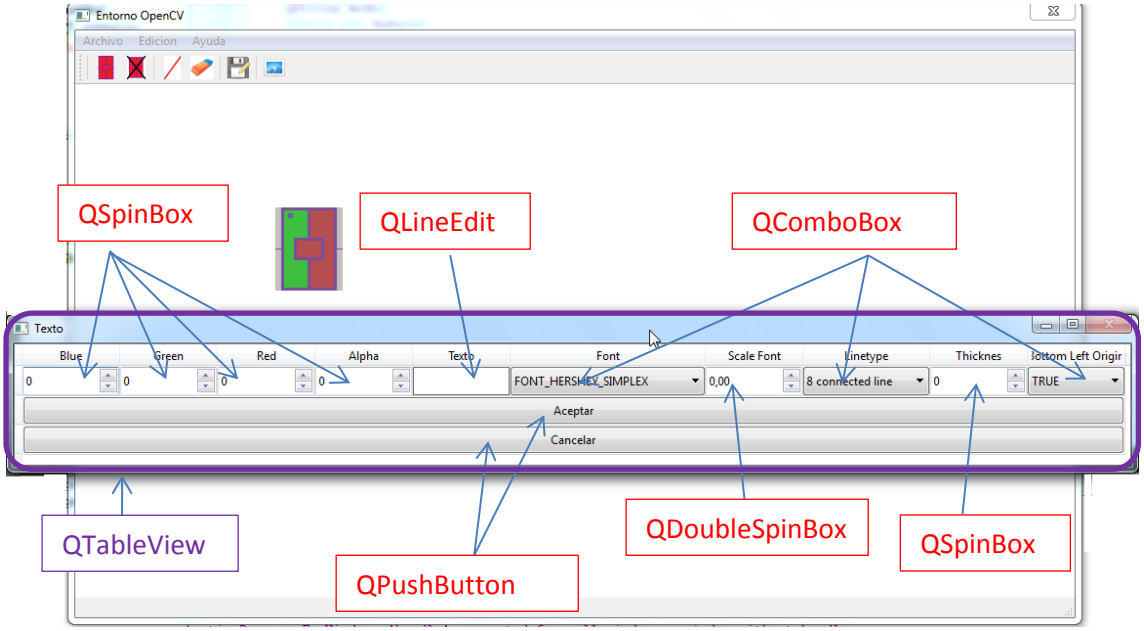


Figura 43. *Bloque o Módulo: Composición de widgets.*

Como se puede observar en la figura anterior, todas las clases utilizadas están contenidas en una principal, que es QTableView. Esta clase es la ventana del widget personalizado y nos permite poder anexionarle cualquier otra clase, pero la forma y orden dentro de la propia ventana depende de las clases QStandardItemModel y QModelIndex.

Estas dos clases dividen en celdas la ventana dejándola ordenada por filas y columnas, de manera que en cada celda es donde se pueden anexionar las otras clases. Para el ejemplo de la figura anterior se puede ver que hay 10 columnas y 3 filas, por lo que hay 30 celdas. Estas celdas creadas se pueden expandir o contraer, un ejemplo de ello son el botón aceptar y cancelar de la figura 43.

La razón de este inciso se debe a que en el módulo se distinguen distintas partes, y según el usuario pulse en cada una de ellas, el entorno le puede pedir información mediante los widgets anteriores para completar la tarea que el propio usuario quiere realizar. Estas partes o pestañas son:

- Entrada.
- Procesamiento
- Salida.

Cada una de estas pestañas tiene dos colores, el color rojo (inactivo) y el color verde (activo). En función de las operaciones que realice el usuario dichos colores de cambiarán.

3.5.1. Entrada

En esta pestaña el usuario puede cargar la imagen de diferentes formas:

1. Abrir imagen
2. Abrir video

En el caso 1, el entorno carga la imagen desde una ruta especificada por el usuario, es decir, carga una imagen desde el disco duro del ordenador.

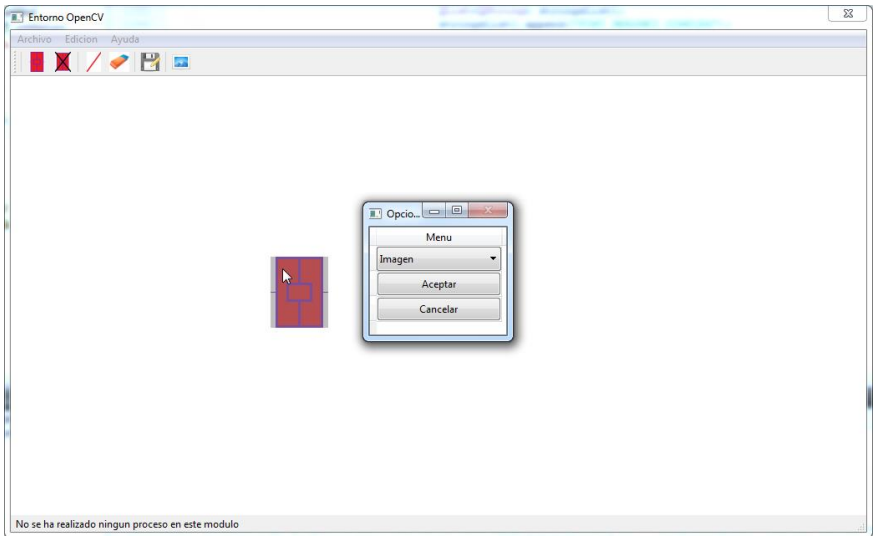


Figura 44. *Módulo: carga de imagen.*

En el caso 2, el usuario tiene la opción de poder realizar una carga de imagen desde un video, es decir, puede realizar una captura de video.

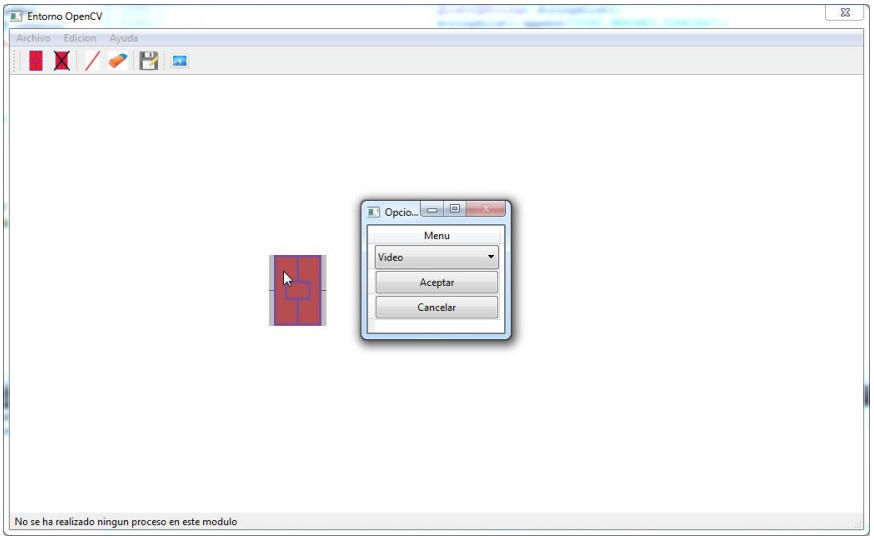


Figura 45. *Módulo: carga de video.*

Se puede observar en las dos figuras anteriores (44 y 45), que el modo de la pestaña de entrada es inactivo o lo que es lo mismo, de color rojo. Esto se debe a que el usuario no ha completado la carga.

En el caso 1, para que se complete la carga, el usuario tiene elegir la imagen a cargar y el entorno comprobar que la imagen elegida por el usuario carga sin ningún error. Si se cumple todo, la pestaña cambia a estado activo (se vuelve de color verde).

En el caso 2, hay dos variantes:

- Carga desde Webcam.
- Carga desde video en disco duro.

Cuando el usuario selecciona la carga por la Webcam, tiene que tener dicho dispositivo conectado al ordenador para que el entorno lo reconozca. En caso de no tenerlo el entorno le envía una señal de aviso al usuario. La captura de imagen se consigue pulsando con el ratón sobre la ventana donde se está visualizando la Webcam. Para la asociar los eventos del ratón con el video se utiliza la función:

- void **setMouseCallback**(const string& **winname**, MouseCallback **onMouse**, void* **userdata**=0)

Esta función es de la librería OpenCV. La variable “winname” es el nombre de la ventana donde se reproduce el video, en este caso siempre es “video”. La variable “onMouse” es el nombre de la función estática a la cual está conectado el evento del ratón.

Con esta función asociamos los eventos del ratón al video, pero en este caso el entorno no utiliza ningún parámetro, como por ejemplo, la posición del puntero dentro de un frame del video. Solo necesita que cuando el usuario pulse sobre el video, se origine una señal que se conecte a la variable “onMouse”.

El nombre de la función estática que se encuentra en la variable “onMouse” es:

- static void mouse_video(int event, int x, int y, int flags, void* param)

Cuando el usuario selecciona la carga por video desde disco duro, solo tiene que elegir el video en la ruta en la que se encuentre e inmediatamente empezará a reproducirse en la ventana de carga. Cuando el video llega al final de su reproducción automáticamente vuelve a empezar a reproducirse entrando así en bucle. Esto se debe a que la intención del alumno es que el usuario pueda ver toda la secuencia del video antes de realizar la captura de imagen en caso de que lo necesite. El bucle de reproducción no se acaba hasta que el usuario pulsa con el ratón sobre la ventana como en el caso de la Webcam. En el único formato que se puede realizar la carga del video es AVI (Audio Video Interleave) ya que es lo que soporta las librerías OpenCV.

En los dos casos se puede pausar la visualización pulsando cualquier tecla del teclado. En la figura 46 se muestra un ejemplo:

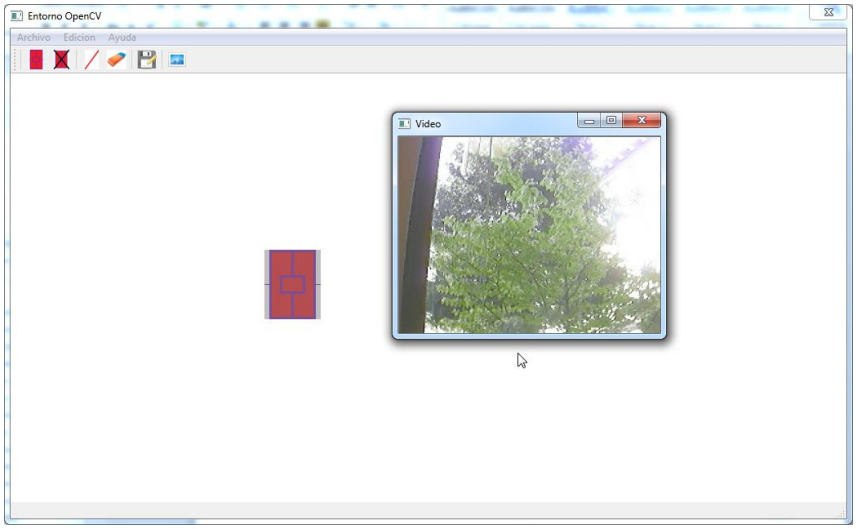


Figura 46. *Módulo: captura de video.*

Cualquiera de los tipos de carga explicados anteriormente provoca el cambio de estado de la pestaña de entrada del módulo. Como se ha comentado, si se produce la carga de manera correcta y sin ningún error se produce el paso de inactiva a activa. Cuando la carga es correcta, la forma que toma el módulo para el ejemplo anterior se puede ver en la figura 47:

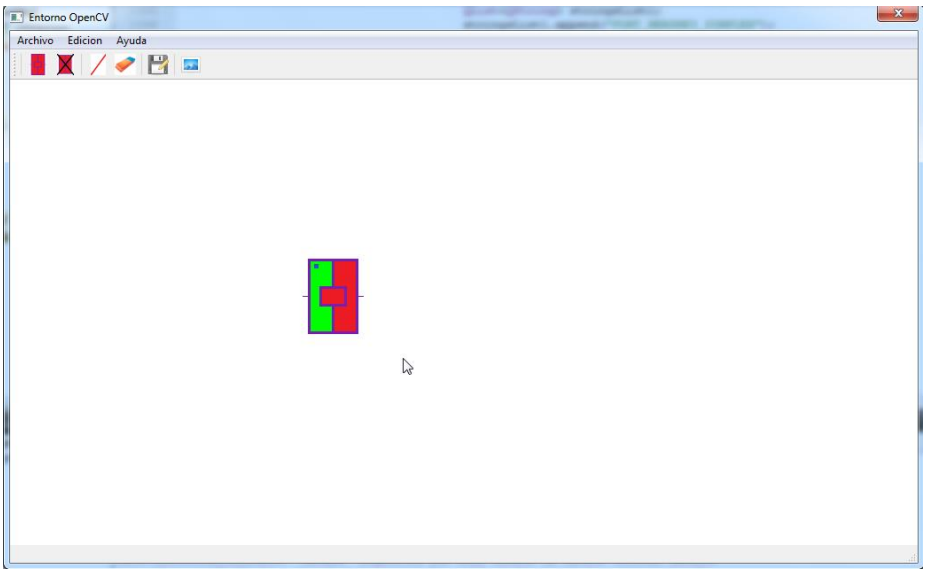


Figura 47. *Módulo: pestaña de entrada activa.*

Cuando la pestaña de entrada del módulo está en estado activo, ésta queda limitada de manera que solo mostrará la imagen cargada cada vez que el usuario pulse sobre ella, es decir, no se podrá volver a cargar otra imagen en el mismo módulo una vez que este ya tenga una imagen de entrada. Para el ejemplo anterior, si el usuario pulsa en la pestaña una vez cargada la imagen, el resultado se puede ver en la figura 48:

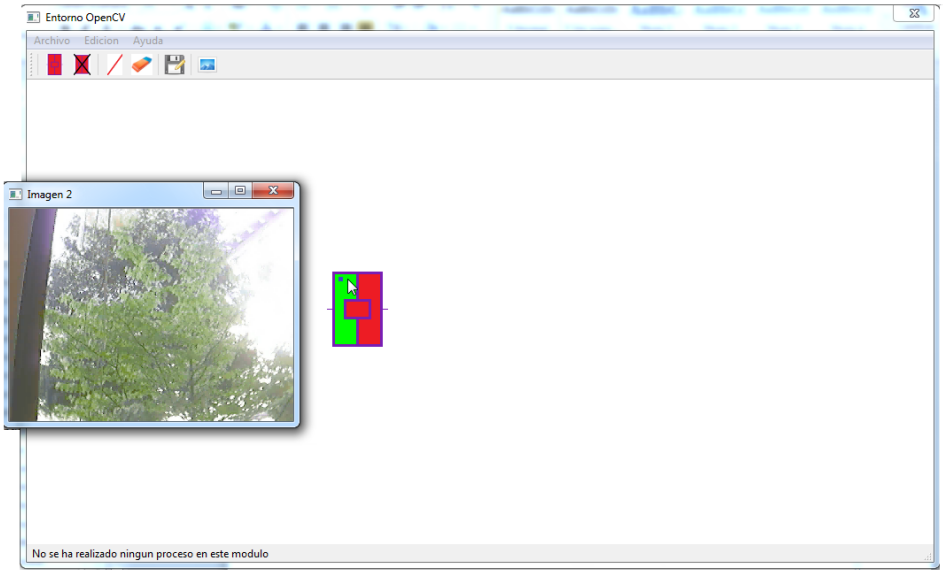


Figura 48. *Módulo: mostrar captura de video.*

3.5.2. Procesamiento

En esta pestaña es donde se va a producir el uso de las librerías OpenCV, ya que es donde el usuario elige que tratamiento realizar a la imagen que se encuentra en la pestaña de entrada. Cada uno de los tratamientos que se aplican a las imágenes se explicará en los capítulos siguientes.

En esta pestaña el usuario se va a encontrar una ventana con una lista de tratamientos que puede realizar. Para la creación de esa ventana se ha utilizado la clase QListWidget. Esta clase permite poder crear una ventana e introducir una lista, pero lo más característico es que la lista se puede conectar mediante señales y slots. Esto es importante, ya que de esta manera el entorno sabe qué ha escogido el usuario dentro de la lista. Cuando el usuario escoge una opción de la lista, el entorno recibe una señal con la posición de la lista donde el usuario ha pulsado.

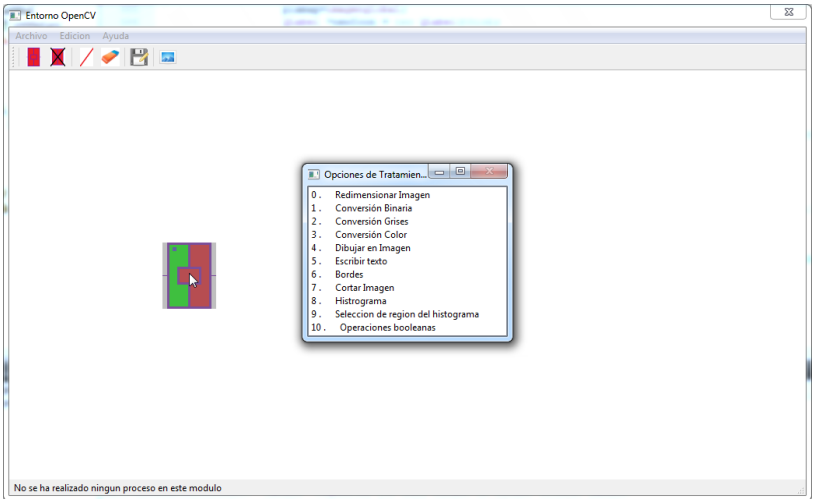


Figura 49. ***Módulo: Lista de tratamientos.***

Para elegir una de las opciones que se pueden visualizar en la figura 49 se debe hacer doble click con el ratón sobre la opción deseada. Si solo se hace un click la ventana de opciones no se cierra, debido a que esta implementado de esta manera.

Si se observa bien la figura 49, se puede ver que el botón cerrar de la ventana está desactivado. Esto se debe a que el alumno quiere obligar al usuario a elegir una de las opciones propuestas. Mientras que el usuario no elija alguna de las opciones propuestas, la ejecución del programa permanecerá pausada.

En caso de que el usuario pulse sobre la pestaña de procesamiento y el módulo no tenga ninguna imagen asociada a su entrada, saldrá igualmente la ventana de opciones de procesamiento, aunque si se pulsa sobre cualquiera de las opciones, el entorno lanzará al usuario un mensaje de error.

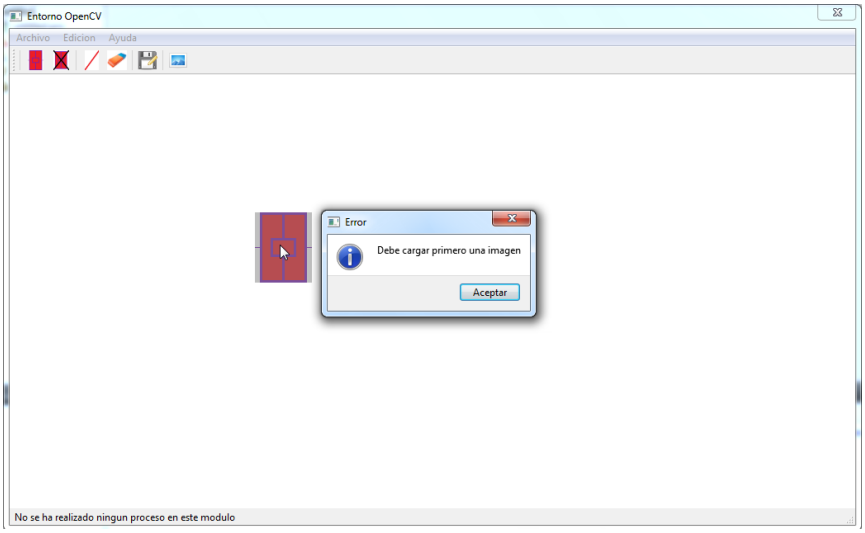


Figura 50. ***Módulo: Procesamiento sin imagen de entrada.***

Una vez que el usuario ha elegido una de las opciones, el entorno comienza con el procesado de la imagen de entrada del módulo. Durante el procesado de la imagen se pueden dar dos casos:

1. El procesado se ha realizado correctamente.
2. Ha habido un error durante el procesamiento.

En el caso 1, la pestaña de procesamiento cambia de estado inactivo (color rojo), a estado activo (color verde).

En el caso 2, el entorno emite un mensaje de error al usuario y la pestaña de procesado no cambia de estado, se mantiene en el estado inactivo. Si ocurre este último caso se puede deber al formato de la imagen de entrada o que la imagen no esté cargada correctamente. En la figura 51 se muestra el estado del módulo para el caso 1:

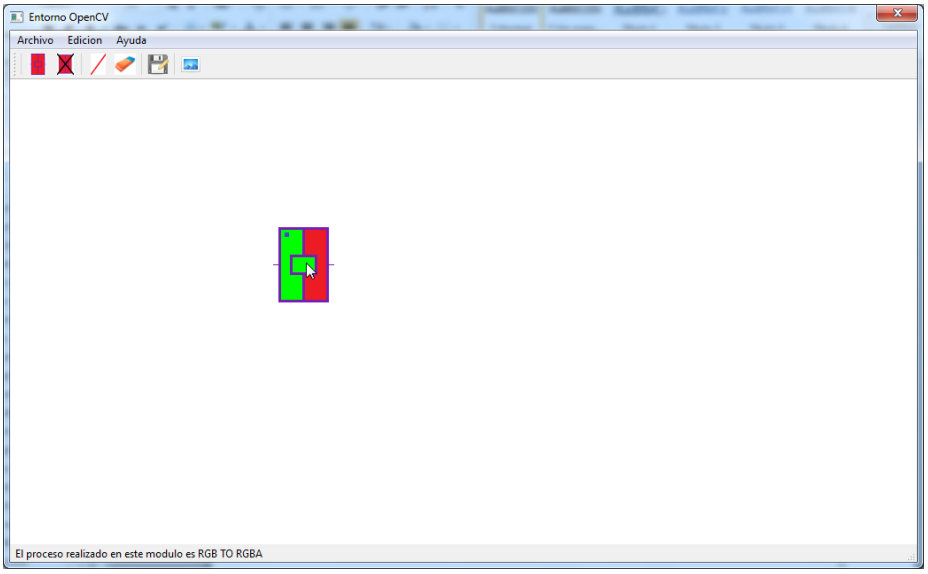


Figura 51. *Módulo: Procesamiento correcto.*

3.5.3. Salida

Esta pestaña del módulo es la más sencilla ya que su función es mostrar los resultados del procesamiento. Para que esto ocurra se debe haber realizado correctamente el procesamiento de la imagen ya que se provocaría un error.

Cuando el usuario quiere ver el resultado del procesamiento realizado, pulsa sobre la pestaña y sale una ventana con la imagen tratada. Los modos de esta pestaña al igual que los de las otras, son los mismos, es decir, hasta que el usuario no pulsa sobre la pestaña por primera vez, ésta no cambia de modo.

Cuando se da este caso la pestaña pasa de estar en modo inactivo a modo activo y una vez que está activa ya no vuelve a cambiar de modo. Si se pulsa una vez ya jamás volverá al estado inactivo. Si se produjese algún error, el entorno avisa al usuario mediante un mensaje de error.

Se va a suponer un ejemplo, en el que el usuario tiene que convertir una imagen cualquiera a escala de grises, por lo que:

1. El usuario elige una imagen cualquiera.
2. El usuario realiza el proceso de conversión de grises que aparece en la lista de opciones.
3. El usuario quiere ver el resultado del tratamiento realizado.

Una vez realizado estos tres pasos la salida debería ser la misma que la de la figura 52:

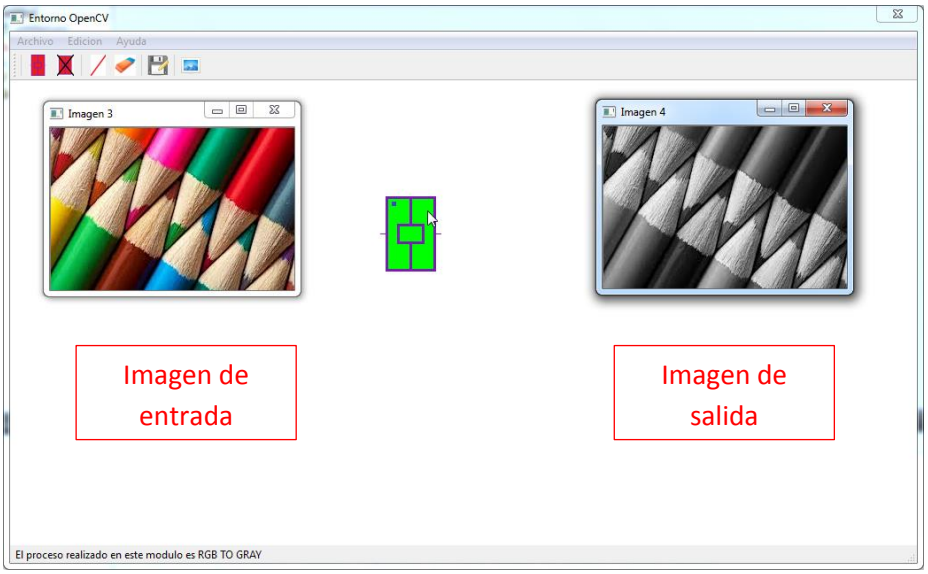


Figura 52. *Módulo: ejemplo.*

3.6. Funciones OpenCV

En el entorno se utilizan varias funciones de las librerías OpenCV. Dichas funciones están orientadas a programadores de las librerías OpenCV, por lo que el usuario debe poder identificar todos los parámetros de las funciones. Estos parámetros son los mismos que se utilizan en la API de la librería OpenCV. Estas funciones son los posibles tratamientos que el usuario puede realizar a las imágenes que utilice.

Las funciones que el alumno ha desarrollado para su utilización por parte del usuario son las siguientes:

- int rectangulo(int id)
- int circulo(int id)
- int linea(int id)
- int elipse(int id)
- int texto(int id)
- int canny(int id)
- int laplacian(int id)
- int cortar_imagen(int id)
- int histograma(int id)
- int operacion_booleana(int id, QString s1, QString s_mask)
- int roi_booleana(int id)
- int conversion_blanconegro(int id_in)
- int conversion_color(int id_in)
- int imagensize(int id_in)
- int conversion_grayscale(int id_in)

Estas funciones son los procesos principales del entorno, pero también se han utilizado funciones auxiliares que son necesarias para que las funciones anteriores se ejecuten de forma correcta. Dichas funciones se citan a continuación:

- static void mouse_centro(int event, int x, int y, int flags, void* param)
- static void mouse_line(int event, int x, int y, int flags, void* param)
- static void CannyThreshold(int, void *)
- bool seleccion_centro(int id)
- bool seleccion_linea(int id)

Se puede observar que todas las funciones principales devuelven un entero (int). Esto se debe a que en cada una de ellas se realiza el tratamiento a la imagen y a continuación envía su identificador para que la imagen tratada se quede anexionada al módulo. Cada una de las funciones anteriores, se explicara con más detalle en los subcapítulos siguientes.

3.6.1. Redimensionar imagen

Esta función sirve para redimensionar la imagen que el usuario quiera con las dimensiones dadas por él mismo.

Para ello se utiliza la siguiente función de la librería OpenCV:

- void **resize**(InputArray **src**, OutputArray **dst**, Size **dsize**, double **fx**=0, double **fy**=0, int **interpolation**=INTER_LINEAR)

Esta función permite redimensionar la imagen con el tamaño deseado y con el método de interpolación deseado. Los diferentes tipos de interpolación que el usuario puede escoger son:

- INTER_NEAREST
- INTER_LINEAR
- INTER_AREA
- INTER_CUBIC
- INTER_LANCZOS4

Para realizar el proceso redimensionar imagen, el usuario debe seleccionar la altura, anchura y el tipo de interpolación. La altura y la anchura están limitadas por el alumno a un rango de [1,2000]. El alumno ha decidido que el valor máximo que puede llegar a tener una imagen es 2000x2000.

El widget personalizado de esta función se puede ver en la figura 53:

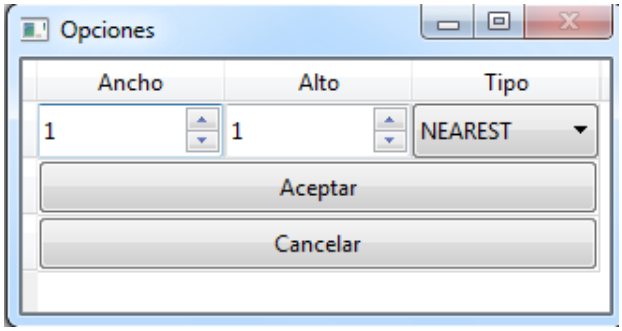


Figura 53. *Funciones OpenCV: redimensionar imagen.*

Si el usuario decide no realizar el proceso, puede pulsar sobre el botón cancelar y el modulo se queda en el mismo estado que estaba antes de pulsar sobre el proceso.

3.6.2. Conversión binaria

Esta función permite al usuario convertir una imagen con varios niveles de gris a una nueva con solo dos, de manera que los objetos quedan separados del fondo. Para conseguir esto, el usuario debe elegir el valor de umbral correspondiente. Este valor no es fácil de encontrar por dos razones:

- Hay ruido en las imágenes.
- El objeto y el fondo no tienen un único valor de gris, sino un intervalo que provoca el solapamiento de valores.

Básicamente, filtra los píxeles de una imagen teniendo en cuenta para su discriminación la intensidad de cada píxel. En una imagen de 8 bits sin signo, un píxel puede tomar valores desde 0 hasta 255, siendo 0 el negro absoluto y 255 el blanco absoluto.

Esta función sirve para filtrar una imagen por medio de un umbral dado, si el valor del píxel es mayor al umbral hace algo o si el valor del pixel es menor que el umbral hace otra cosa. Lo que haga depende del tipo de umbral que se use.

La función que se ha utilizado de la librería OpenCV es:

- double **threshold**(InputArray **src**, OutputArray **dst**, double **thresh**, double **maxval**, int **type**)

El usuario puede seleccionar los valores de:

- Thresh
- Maxval
- Type

Los valores de Thresh y Maxval se encuentran dentro del rango de [1,255]. Por otro lado, la variable Type puede ser:

- THRESH_OTSU: la función determina el valor umbral óptimo usando el algoritmo de Otsu y lo utiliza en lugar del umbral especificado. La función devuelve el valor de umbral calculado. Actualmente, el método de Otsu se lleva a cabo sólo para imágenes de 8 bits.
- THRESH_BINARY: Si la intensidad del píxel en la posición (x, y) de la matriz de la imagen es mayor que Thresh, se establece el valor de Maxval como nueva intensidad del píxel. De lo contrario, se establece el valor 0.
- THRESH_BINARY_INV: Si la intensidad del píxel en la posición (x, y) de la matriz de la imagen es mayor que Thresh, se establece el valor 0 como nueva intensidad del píxel. De lo contrario, se establece el valor de Maxval.
- THRESH_TRUNC: El valor máximo de la intensidad de los píxeles es Thresh, es decir, si la intensidad del pixel en la posición (x, y) de la matriz de la imagen es mayor, entonces su valor se trunca.
- THRESH_TOZERO: Si la intensidad del píxel en la posición (x, y) de la matriz de la imagen es menor que Thresh, se establecerá el valor 0 como nueva intensidad del píxel.
- THRESH_TOZERO_INV: Si la intensidad del píxel en la posición (x, y) de la matriz de la imagen es mayor que Thresh, se establecerá el valor 0 como nueva intensidad del píxel.

Para entender mejor los tipos de umbrales se puede observar en la figura 54 el efecto de cada uno se ellos sobre una imagen cualquiera. La linea azul es el umbral seleccionado.

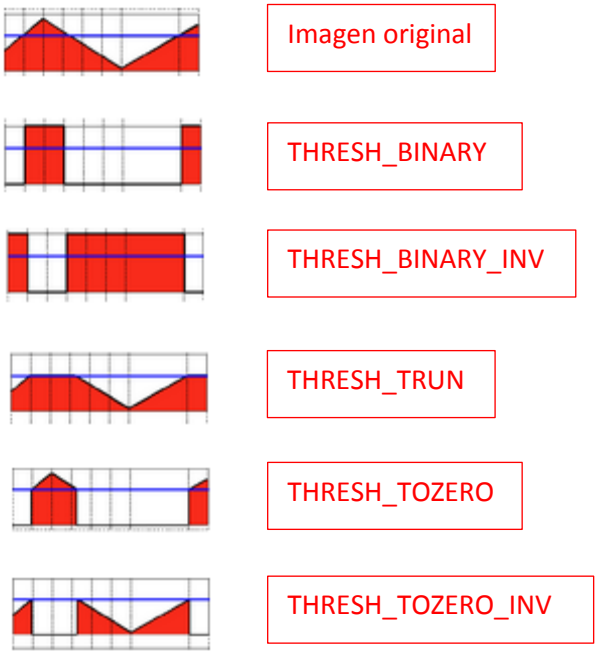


Figura 54. . *Funciones OpenCV: Gráficas de los tipos de umbrales. [14]*

El usuario debe tener en cuenta que esta función extrae un canal de cualquier definición de espacio de color en la que se encuentre la imagen (RGB, HSV, etc…). Por ello, si la imagen tiene tres canales no se podrá usar la función threshold. Si se produce este caso, el entorno avisará al usuario mediante un mensaje sugiriéndole que realice primero el tratamiento de conversión de grises ya que realizando dicho tratamiento la imagen pasará a tener un solo canal. El caso comentado se puede ver en la figura 55, en donde se intenta usar esta función a una imagen de tres canales:

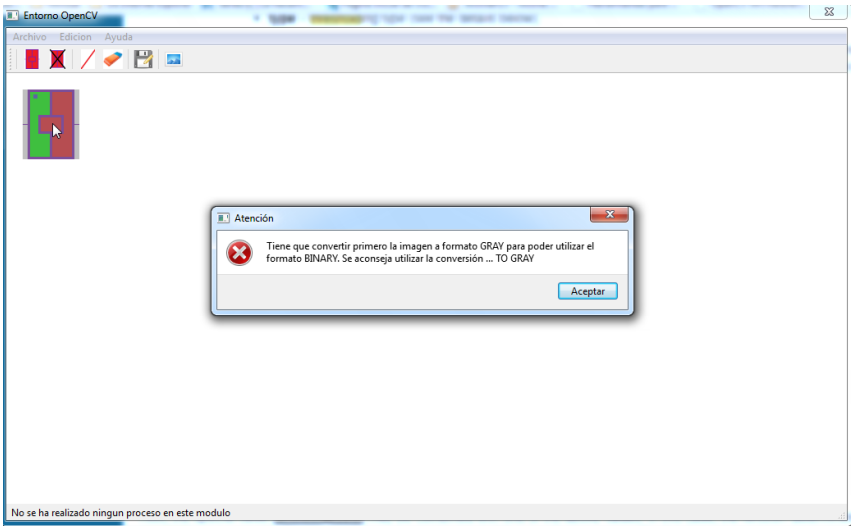


Figura 55. *Funciones OpenCV: caso único en conversión binaria.*

El widget personalizado de esta función se puede ver en la figura 56:

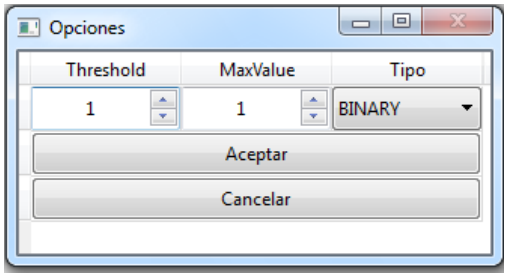


Figura 56. *Funciones OpenCV: conversión binaria.*

3.6.3. Conversión de grises

Esta función es muy simple ya que para convertir a grises o viceversa solo es necesario que el usuario seleccione el tipo de conversión. La función que se ha utilizado de la librería OpenCV para realizar este tratamiento es:

- void **cvtColor**(InputArray **src**, OutputArray **dst**, int **code**, int **dstCn**=0)

El único parámetro que el usuario puede elegir es "code", que es el tipo de conversión que el usuario puede escoger. Los valores que el usuario puede dar a la variable "code" son los siguientes:

- RGB TO GRAY
- RGBA TO GRAY
- BGR TO GRAY
- BGRA TO GRAY
- GRAY TO RGB or BGR
- GRAY TO RGBA or BGRA

Si el usuario convierte una imagen a formato GRAY y luego quiere utilizar los formatos de conversión GRAY TO RGB or BGR y GRAY TO RGBA or BGRA se pierde información de la imagen, es decir, se realiza la conversión de una imagen de un canal a otra de tres canales, por lo que la imagen de un canal se copiará en cada canal de la imagen de tres canales, es decir, la imagen original no se restaurará. El resultado de esto es aumentar la imagen de un canal a tres. En la figura siguiente se puede observar el widget personalizado para esta función:

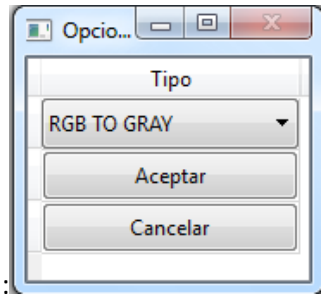


Figura 57. *Funciones OpenCV: conversión de grises.*

3.6.4. Conversión de color

Esta función sirve para dar diferentes formatos de color a la imagen que el usuario desee. Los modelos o espacios de colores que se han implementado en esta función son:

- HSV (Hue Saturation Value)
- HSL (Hue Saturation Lightness)

La elección de estos espacios de colores se debe a que se ha diseñado el entorno de manera que el usuario pueda utilizar dos variaciones (HSV, HSL) del espacio de color HSI (Hue Saturation Intensity). "*Este espacio se basa en el modo que tenemos de percibir los humanos. Los componentes de este espacio, se muestran favorables de cara a realizar segmentaciones de la imagen en atención al tono o tinte del color*". [1]

Un espacio de color es un modelo matemático abstracto que describe la forma en la que los colores pueden representarse. Los espacios de colores HSV y HSL son muy comunes. Esto se debe a que presentan la ventaja de que discriminan los tonos de los objetos (valores de rojos y verdes). Esto sirve para que el usuario pueda identificarlos y separarlos dentro de la imagen.

Las representaciones de los modelos se pueden ver en la siguiente figura:

Modelo de color HSV

Modelo de color HSL

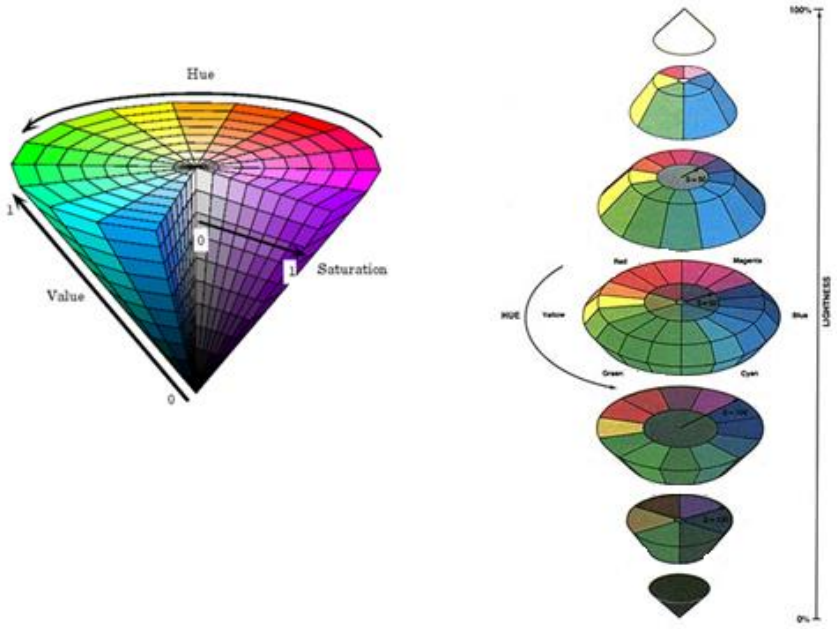


Figura 58. *Funciones OpenCV: Espacios de color HSV y HSL. [15][16]*

Si se observa la figura 58 se puede ver que hay diferencia entre los espacios de color HSV y HSL en su representación cilíndrica, pero la diferencia más importante es que en el espacio HSV se debe establecer la saturación a "0" (0% de saturación) para obtener el color blanco, mientras que en el espacio HSL en L = 1 (100%) se obtiene blanco independientemente del valor de saturación.

En el modelo HSV el rango de S se extiende desde 0 (coincidiendo con el eje de la pirámide) hasta 1, coincidiendo con el final del área hexagonal de la pirámide. La pirámide tiene altura unidad. El vértice corresponde al negro con coordenadas S = 0 y V = 0. El blanco corresponde a S = 0 y V = 1. Los valores coincidentes con el eje de la pirámide son los grises. Cuando S = 0 el valor de H no es importante y se dice que está indefinido. Cuando S > 0, el valor de H empieza a tener importancia.

En el modelo HSL se define sobre una doble pirámide hexagonal. H es el ángulo alrededor del eje vertical, situándose el rojo a 0º. Los colores tienen el mismo orden que en el espacio HSV, estando su color complementario a 180º de él. La saturación se mide radialmente variando desde 0 a 1. La luminosidad es 0 para el negro (en el vértice inferior de la pirámide) y 1 para el blanco (en el vértice superior de la pirámide). En este modelo los grises tienen todos el valor S = 0 y los tonos más saturados se encuentran a S = 1 y L = 0.5.

Para implementar estos espacios de colores se ha utilizado la misma función que en el subcapítulo anterior (conversión de grises):

- void **cvtColor**(InputArray **src**, OutputArray **dst**, int **code**, int **dstCn**=0)

Aunque la función es la misma, hay una diferencia. La única diferencia es que los valores que la variable "code" tiene son diferentes. Estos valores son:

- RGB TO RGBA
- RGB TO BGR
- RGB TO BGRA
- RGB TO HSL
- RGB TO HSL_FULL
- RGB TO HSV
- RGB TO HSV_FULL
- RGBA TO BGRA y viceversa
- RGBA TO mRGBA
- RGBA TO RGB or BGRA TO BGR
- RGBA TO BGR or BGRA TO RGB
- BGR TO HSL
- BGR TO HSL_FULL
- BGR TO HSV
- BGR TO HSV_FULL
- HSV TO BGR
- HSV TO BGR_FULL

- HSV TO RGB
- HSV TO RGB_FULL
- HSL TO BGR
- HSL TO BGR_FULL
- HSL TO RGB
- HSL TO RGB_FULL

El widget personalizado de esta función es muy simple y se puede ver en la figura 59:

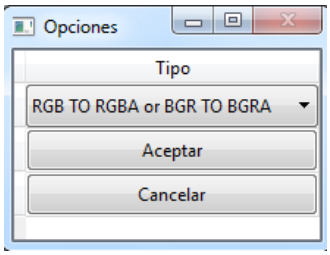


Figura 59. *Funciones OpenCV: conversión de color.*

3.6.5. Dibujar en imagen

La función dibujar en imagen se divide en las siguientes partes:

- Dibujar círculo.
- Dibujar cuadrado o rectángulo.
- Dibujar elipse.
- Dibujar línea.

Cada una de estas funciones se dibujan en la imagen según como quiera el usuario, pero cumpliendo con las condiciones que el entorno exige al usuario en cada una de ellas. Estas condiciones, son las variables que el entorno necesita para dibujar y la posición en la imagen que la indica el usuario con el ratón.

Para que el usuario pueda indicar las posiciones en la imagen se utilizan eventos del ratón que se dan en las mismas. Estos eventos son completamente diferentes para cada una de las imágenes donde el usuario desea dibujar. Para asociar a la imagen eventos, se necesita la función siguiente:

- void **setMouseCallback**(const string& **winname**, MouseCallback **onMouse**, void* **userdata**=0)

Esta función es de las librerías OpenCV y permite registrar la posición del puntero del ratón cuando se pulsa sobre la imagen. Saber esta posición es muy importante, ya que el entorno toma esta posición como el centro o vértice de los dibujos anteriores.

Para dibujar un círculo y una elipse, se necesita solo un punto, pero para dibujar un rectángulo o cuadrado y una línea se necesitan dos puntos. Para la selección de los puntos se utilizan las funciones siguientes:

- static void mouse_centro(int event, int x, int y, int flags, void* param)
- static void mouse_line(int event, int x, int y, int flags, void* param)

Este proceso es exactamente igual que la captura de video, con la diferencia de que en este caso el entorno necesita los datos de los eventos del ratón. Una vez entendido como se asocian los eventos del ratón sobre una imagen.

3.6.5.1. Círculo

Para dibujar el círculo el entorno debe saber la posición de su centro. Para ello el usuario pulsa sobre la imagen en el lugar donde quiere que se sitúe el centro del círculo, pero antes de este paso el entorno tiene que disponer de todos los datos para dibujarlo en la imagen.

Cuando el usuario ha elegido la posición del centro, el entorno utiliza la función siguiente función de la librería OpenCV:

- void **circle**(Mat& **img**, Point **center**, int **radius**, const Scalar& **color**, int **thickness**=1, int **lineType**=8, int **shift**=0)

Esta función permite dibujar círculos sobre imágenes. Los valores que el usuario va a dar al entorno son los que coinciden con los títulos de las columnas del widget personalizado para esta función (ver figura 60).

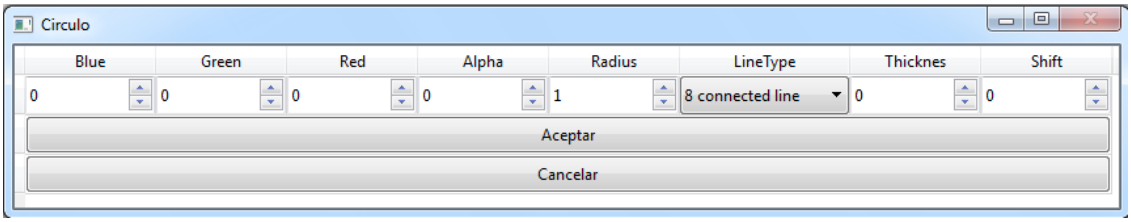


Figura 60. *Funciones OpenCV: widget de la función dibujar círculo.*

Blue, Green, Red, y Alpha son los valores para que se forme el color deseado del círculo. Estos valores se introducen para la variable "color" de tipo Scalar. El rango de la variable radius varía entre [1,2000], la variable thickness varía entre [-1,255] y el resto de variables varían entre [0,255].

En la figura 61 se muestra un ejemplo en donde el valor del radio es 50:

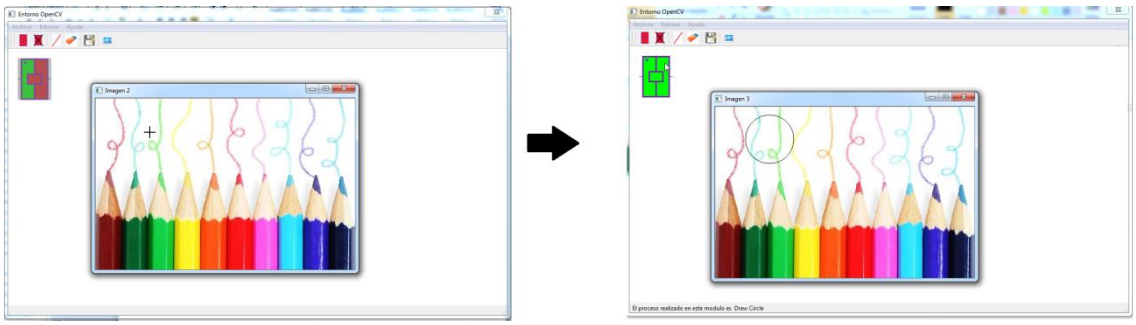


Figura 61. *Funciones OpenCV: dibujar círculo.*

3.6.5.2. Cuadrado o rectángulo

La metodología de esta función es la misma que la de la función anterior, es decir, se necesita los eventos del ratón. La única diferencia con la función anterior es que se necesitan dos pulsaciones en vez de una.

Esto se debe a que la primera pulsación con el ratón en la imagen va a ser una de las esquinas del cuadrado o el rectángulo y la segunda, la esquina contraria. De esta manera, el usuario puede elegir tanto altura como anchura y dependiendo de estos dos parámetros, se formará un rectángulo o un cuadrado. Para entender como el usuario tiene que pulsar sobre la pantalla se puede observar la figura 62:

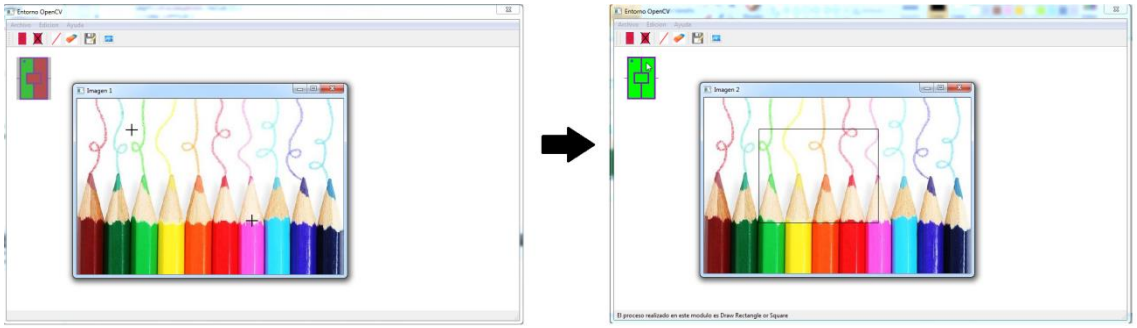


Figura 62. *Funciones OpenCV: dibujar cuadrado o rectángulo.*

La función utilizada de la librería OpenCV es:

- void **rectangle** (Mat& **img**, Point **pt1**, Point **pt2**, const Scalar& **color**, int **thickness**=1, int **lineType**=8, int **shift**=0)

Como en la función anterior, también posee su widget personalizado para que el usuario pueda utilizar los diferentes parámetros (ver figura 63).

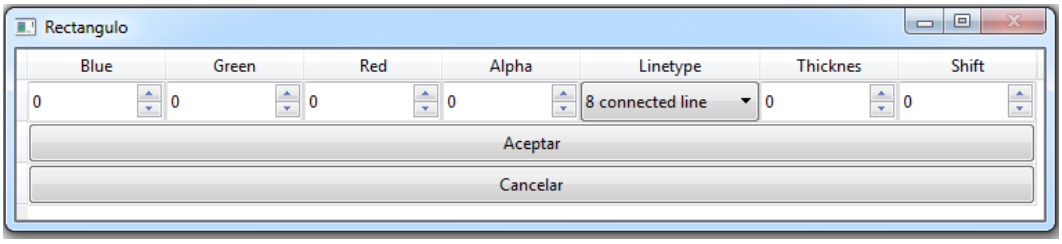


Figura 63. *Funciones OpenCV: widget personalizado de la función dibujar cuadrado o rectángulo.*

El rango de la variable thickness varía entre [-1,255] y el resto de variables varían entre [0,255].

3.6.5.3. Linea

Esta función es exactamente igual que la función anterior, la función dibujar un rectángulo o cuadrado. Lo único que cambia es la función que se utiliza de la librería OpenCV. Dicha función es:

- void **line**(Mat& **img**, Point **pt1**, Point **pt2**, const Scalar& **color**, int **thickness**=1, int **lineType**=8, int **shift**=0)

Debido a que su implementación es igual que la de la función anterior, su widget personalizado, es el mismo (ver figura 64).

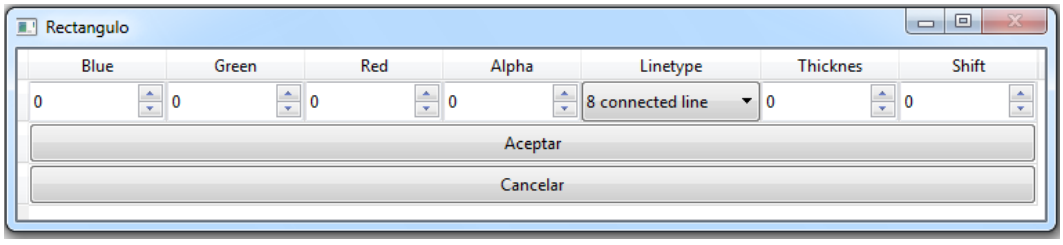


Figura 64. *Funciones OpenCV: widget personalizado de la función dibujar línea.*

A diferencia del rectangulo la variable thickness no puede tener un valor negativo por lo que el nuevo rango para esta variable es entre [0,255] al igual que el resto de las variables restantes. Un ejemplo de uso se puede ver en la figura siguiente:

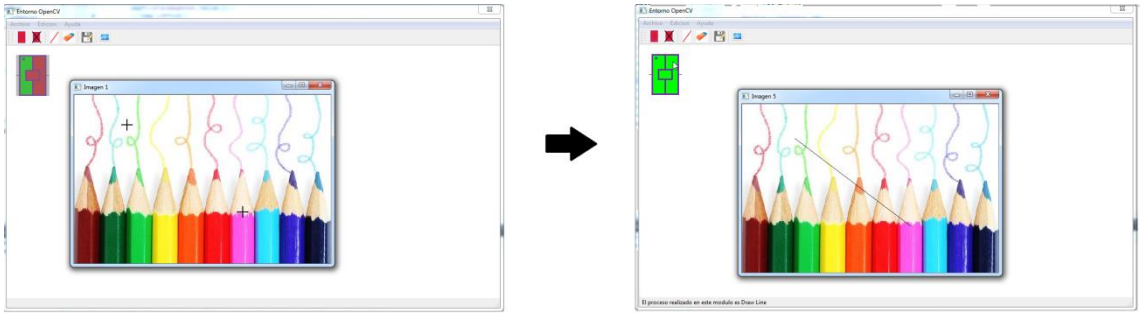


Figura 65. *Funciones OpenCV: función dibujar línea.*

3.6.5.4. Elipse

La elipse es la función que más parámetro tiene, por lo que para entender esta función es aconsejable revisar el API de las librerías OpenCV. La función que realiza el dibujo de una elipse sobre una imagen es:

- void **ellipse** (Mat& **img**, Point **center**, Size **axes**, double **angle**, double **startAngle**, double **endAngle**, const Scalar& **color**, int **thickness**=1, int **lineType**=8, int **shift**=0)

Al igual que en la función dibujar círculo, para la elipse el usuario solo necesita pulsar con el ratón donde quiera que se sitúe el centro de la misma. Si se observa la función que pertenece a la librería OpenCV, se puede ver que hay tres variables que son ángulos de la elipse. Las variables son:

- doublé angle.
- doublé startAngle.
- doublé endAngle.

El rango de estas variables varía entre [-360,360]. La variable angle es el ángulo de rotación de la elipse, es decir, si el ángulo es 180 grados se rota la elipse dicho ángulo.

La variable startAngle marca el ángulo desde que el usuario quiere que se empiece a dibujar la elipse, es decir, si el ángulo es 90 grados, la función empezará a dibujar la elipse desde ese mismo ángulo hasta el ángulo final que se marca con la última variable que es endAngle.

Para introducir los datos necesarios para que el entorno dibuje la elipse en la imagen, esta función también tiene su propio widget personalizado (ver figura 66).

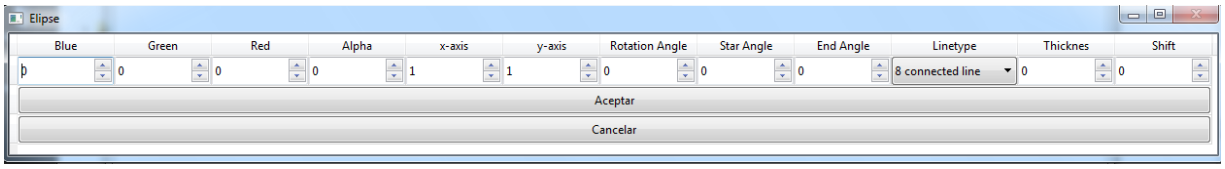


Figura 66. *Funciones OpenCV: widget personalizado función dibujar elipse.*

Para entender mejor esta función, se presenta un ejemplo (figura 67) en el que la variable "angle" es 180, la variable "startAngle" es 90 y la variable "endAngle" es 360 y la variable "x" es 50 y la variable "y" es 30:

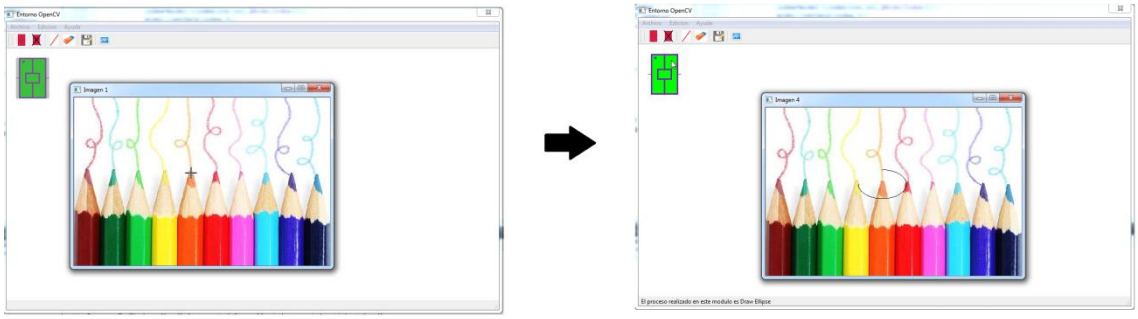


Figura 67. *Funciones OpenCV: función dibujar elipse.*

3.6.6. Escribir texto

Esta función tiene el propósito de escribir sobre una imagen lo que el usuario quiera. Para ello, se utiliza la siguiente función de la librería OpenCV:

- void **putText**(Mat& **img**, const string& **text**, Point **org**, int **fontFace**, double **fontScale**, Scalar **color**, int **thickness**=1, int**lineType**=8, bool **bottomLeftOrigin**=false)

Para esta función, el usuario tiene que posicionar con el ratón el centro del cuadro de texto que se va a escribir. Todos los datos necesarios para la implementación de esta función se proporcionan por el usuario a través del widget personalizado de esta función (ver figura 68).

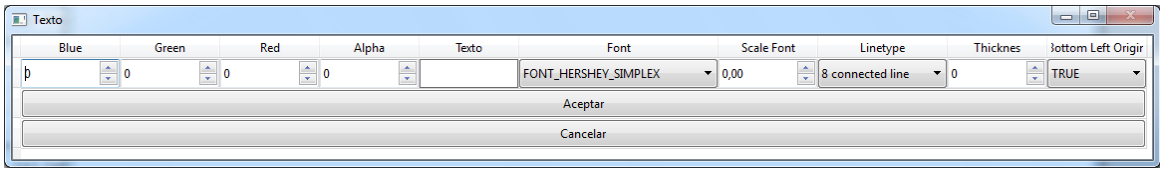


Figura 68. *Funciones OpenCV: widget personalizado de la función escribir texto.*

El rango de las variables varían entre [0,255]. La variable fontScale es especial ya que aunque varía entre dicho rango nunca será 0 ya que se produciría un error. La razón de este valor es que si el usuario quiere que reducir el escalado del texto en vez de ampliarlo, éste debe multiplicar por valores comprendidos entre (0,1), es decir, si el usuario quiere que el escalado sea la mita, debe poner una valor de 0,5 en la variable fontScale. Para comprender mejor como funciona se puede observar la figura 69.

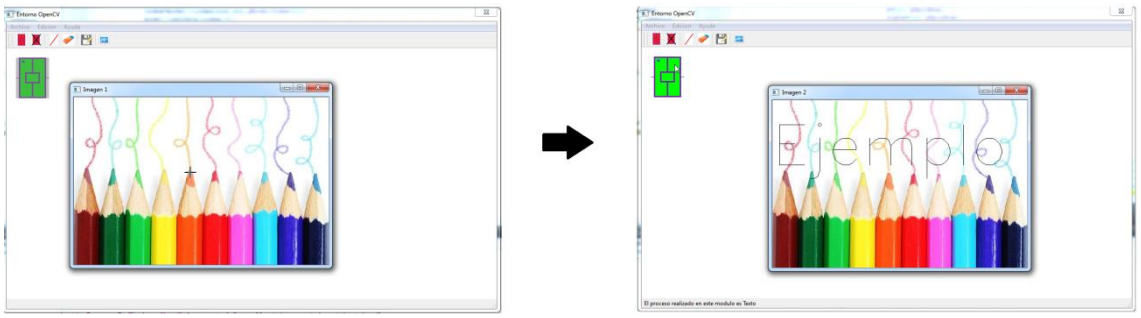


Figura 69. *Funciones OpenCV: función escribir texto.*

3.6.7. Bordes

En la función Bordes el usuario puede utilizar dos funciones diferentes que tienen el mismo propósito. El propósito de las dos funciones es delimitar los objetos en una imagen. "*Para ello la técnica que se utiliza es detectar los puntos donde se produce una variación de intensidad. Para ello se emplean métodos basados en los operadores derivada*" [1]. En esta herramienta utilizaremos las funciones Laplacian y Canny de OpenCV.

En la función Laplacian se utiliza un operador de segundo orden, es decir, se calcula la segunda derivada sobre la imagen muestreada y representa la derivada de ésta respecto a todas las direcciones. Esta derivada se basa en el cálculo de diferencias entre píxeles vecinos, es decir, es una aproximación diferencial. Este operador se utiliza poco ya que tiene mucha sensibilidad al ruido, pero sirve como base a otros operadores como la función Canny que obtiene los bordes a través de *"tres condiciones"* [1]:

1. *Error. Se deben detectar todos y sólo los bordes.*
2. *Localización. La distancia entre el píxel señalado como borde y el borde real debe de ser tan pequeña como se pueda.*
3. *Respuesta. No debe identificar varios píxeles como bordes cuando sólo exista uno.*

La función Canny intenta que el ruido se pueda distinguir de una forma más óptima siguiendo las tres condiciones anteriores, por ello, utiliza la primera derivada de una gaussiana y crea una imagen borrosa. El borde de una imagen puede apuntar en diferentes direcciones, por lo que el algoritmo de Canny utiliza cuatro filtros para detectar las direcciones horizontales, verticales y diagonales en los bordes de la imagen borrosa y con los algoritmos matemáticos dibuja los bordes en la dirección correcta.

Las dos funciones utilizadas de la librería OpenCV son:

- void **Canny**(InputArray **image**, OutputArray **edges**, double **threshold1**, double **threshold2**, int **apertureSize**=3, bool**L2gradient**=false)
- void **Laplacian**(InputArray **src**, OutputArray **dst**, int **ddepth**, int **ksize**=1, double **scale**=1, double **delta**=0, int **borderType**=BORDER_DEFAULT)

Cuando el usuario selecciona esta función, lo siguiente que se encuentra es un menú de selección de función, donde puede seleccionar o la función Canny o la función Laplacian, cada una de estas dos funciones tiene su widget personalizado. Los parámetros que el usuario puede modificar son:

- threshold2 y apertureSize para la función Canny.
- Ddepth, ksize y scale para la función Laplacian.

El resto de las variables tienen valores igual a cero, threshold1, delta, borderType. Esto es debido a que el alumno lo ha diseñado de esta manera para que las funciones sean más genéricas.

En la función Canny el alumno a diseñado una barra de seguimiento estándar en la ventana de la imagen que sirve para que el usuario pueda deslizar dicha barra y así poder ver los bordes que se van borrando, pero antes de que el usuario pueda utilizar dicha herramienta, debe introducir los datos en el widget personalizado de la función Canny:

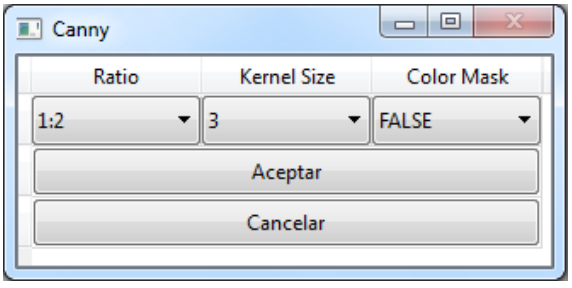


Figura 70. ***Funciones OpenCV: widget personalizado de la función Canny.***

Como se puede observar en la figura 70, se ha añadido dos posibilidades, que son el ratio y la máscara de color. El ratio es entre threshol1 y threshold2, es decir, si por ejemplo el usuario elige 1:2 de ratio, quiere decir que la variable threshold2 tendrá el doble de valor. En cuanto a la mascará de color es una opción en la que los bordes tienen el color de la imagen original. Para entender esta función se muestra un ejemplo en la figura 71.

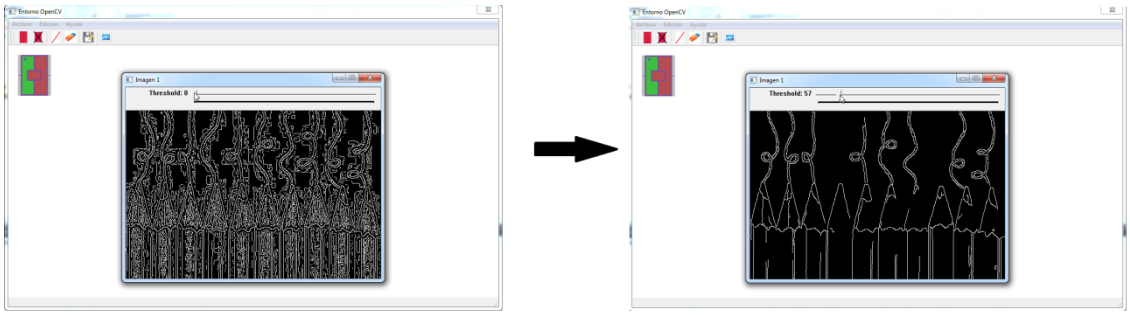


Figura 71. *Funciones OpenCV: trackBar de la función Canny.*

En la figura 71 se puede observar, que en función de la posición de la trackBar (barra de seguimiento estándar) de la ventana hay bordes que van desapareciendo, esto se debe a que no se encuentran en el umbral establecido por la trackBar.

Una vez que el usuario ha decidido el umbral con la trackBar, solo tiene que cerrar la ventana pulsado en el botón cerrar de la propia ventana y la imagen con el umbral establecido se guardará en la salida del módulo. Para el ejemplo anterior, se muestra el uso de la máscara de color en la figura 72.

Figura 72. *Funciones OpenCV: máscara de color de la función Canny.*

En la función Laplacian tenemos también la máscara de color pero no el ratio ya que esta función trae su propia variable de escalado (scale). Funciona exactamente igual que en la función Canny y para ello el entorno necesita que el usuario introduzca los datos en su widget personalizado (ver figura 73).

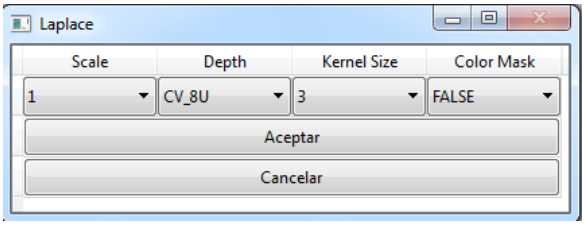


Figura 73. *Funciones OpenCV: widget personalizado de la función Laplacian.*

Un ejemplo de uso de esta función se puede ver en la figura siguiente:

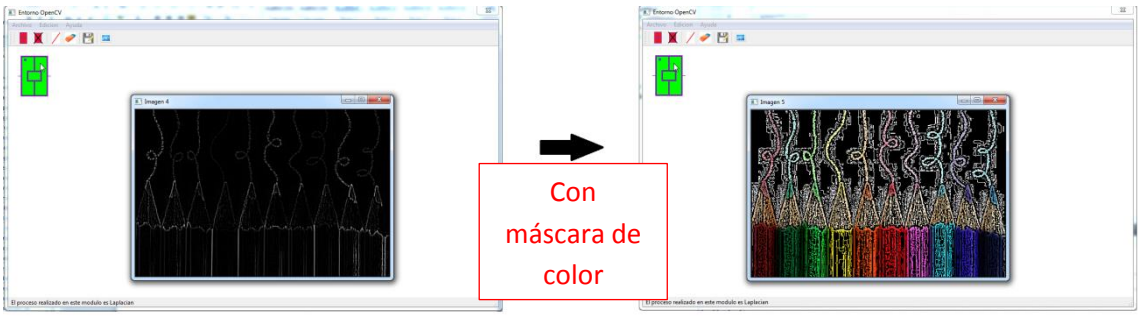


Figura 74. *Funciones OpenCV: función Laplacian.*

3.6.8. Cortar imagen

Esta función es muy sencilla ya que solo se necesita que el usuario elija la zona que quiere de la imagen de entrada del módulo. Para ello, el usuario tiene que pulsar dos veces con el ratón en la imagen. Igual que en la función de dibujar rectángulo o cuadrado, solo que en este caso no dibuja sino que la función copia la parte de la imagen que se queda dentro del rectángulo formado con los dos puntos. Los dos puntos son igual que en la función de dibujar rectángulo o cuadrado, es decir, el primer punto es una esquina y el segundo es la esquina contraria. A diferencia de las otras funciones, ésta no tiene widget personalizado ya que lo único que se necesita, es la interacción del usuario con el entorno seleccionando los dos puntos dentro de la imagen. En la figura 75 se puede ver un ejemplo de uso:

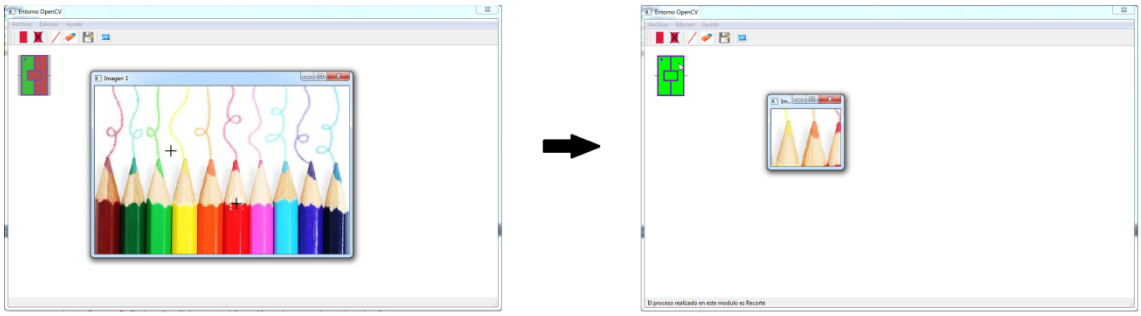


Figura 75. *Funciones OpenCV: función cortar imagen.*

3.6.9. Histograma

La función histograma es la única de las funciones que no realiza ningún tratamiento a la imagen, es decir, solo reporta información sobre ella. El histograma de una imagen representa la frecuencia relativa de los niveles de gris de la misma. Las técnicas de modificación del histograma de una imagen son útiles para aumentar el contraste de imágenes o detectar regiones de interés para el usuario.

La función histograma puede detectar los canales de la imagen de entrada del módulo. Esto es muy útil ya que el usuario, solo tiene que elegir el canal sobre el cuál quiere realizar el histograma. Es decir, si el usuario solo puede realizar el histograma sobre un canal independientemente de que la imagen tenga uno o tres. El histograma no está diseñado para soportar imágenes de cuatro canales, esto se debe a que el alumno ha considerado esto como una mejora futura.

La representación del mismo es una imagen que está formada por 255 columnas y 125 filas que en función del valor que tenga el píxel puede tener una altura de dicho valor o una altura igual a 0 si el valor del píxel es negativo. Los valores de dicho píxel son de tipo uchar (carácter sin signo). Las columnas son pintadas por la función cv::line que se utiliza en el capítulo anterior, por lo que como máximo puede haber 255 columnas que forman cada línea y cada una de ellas puede alcanzar una altura de 125.

Los canales de una imagen son diferenciados mediante el color de las columnas del histograma, es decir, para las imágenes que tengan tres canales, se usan el color rojo, verde y azul (formato RGB), mientras que para las imágenes que dispongan de un solo canal, el color seleccionado es gris (formato GRAY). [17]

La función histograma tiene su propio widget personalizado, pero solo aparece para las imágenes de tres canales, ya que en ese caso, el usuario debe elegir uno de los tres canales posibles. En las imágenes de un canal no es necesario widget ya que el entorno lo detecta y crea el histograma directamente.

El widget personalizado para las imágenes de tres canales se puede ver en la siguiente figura:

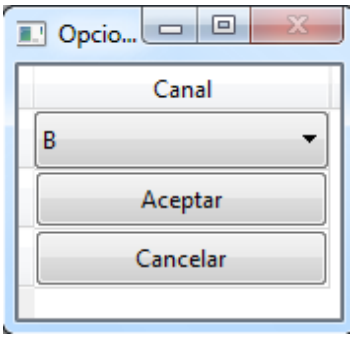


Figura 76. *Funciones OpenCV: widget personalizado función histograma.*

Si el usuario despliega la pestaña se podría ver los tres canales de la imagen de entrada, en este caso concreto son R, G y B. Cada canal tiene su propio histograma y es diferente a los otros, un ejemplo de ello se puede ver en las figuras 77 y 78.

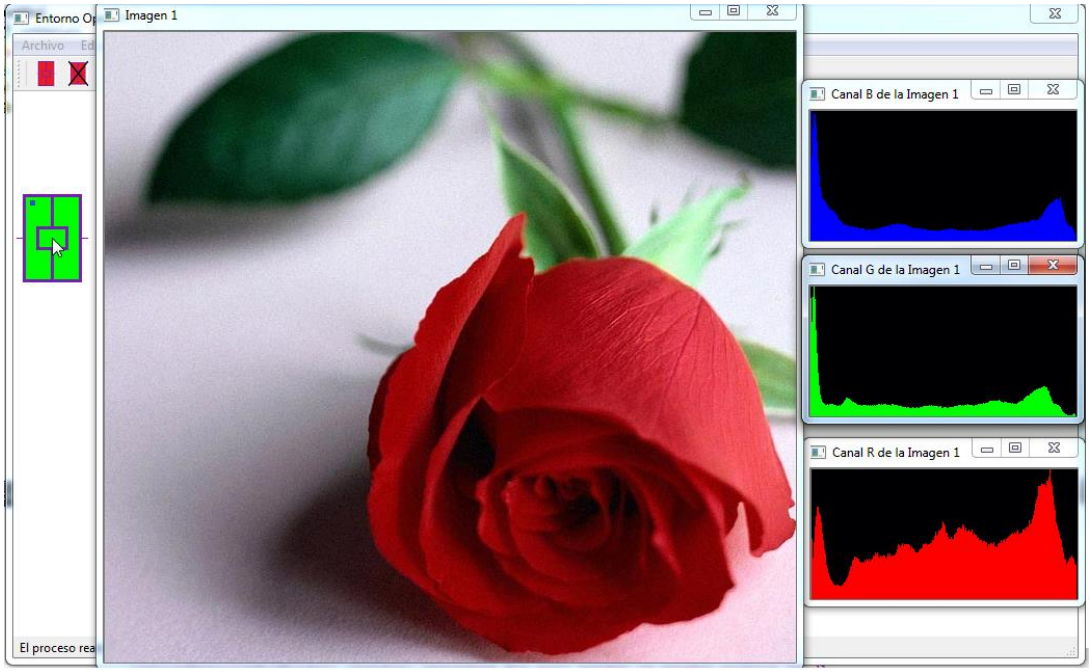


Figura 77. *Funciones OpenCV: función histograma para imagen de 3 canales.*

En la figura 77 se puede ver el histograma de cada componente del espacio de color de la imagen, en este caso, es BGR. En la siguiente figura se ve el histograma para el espacio de color GRAY de la misma imagen.

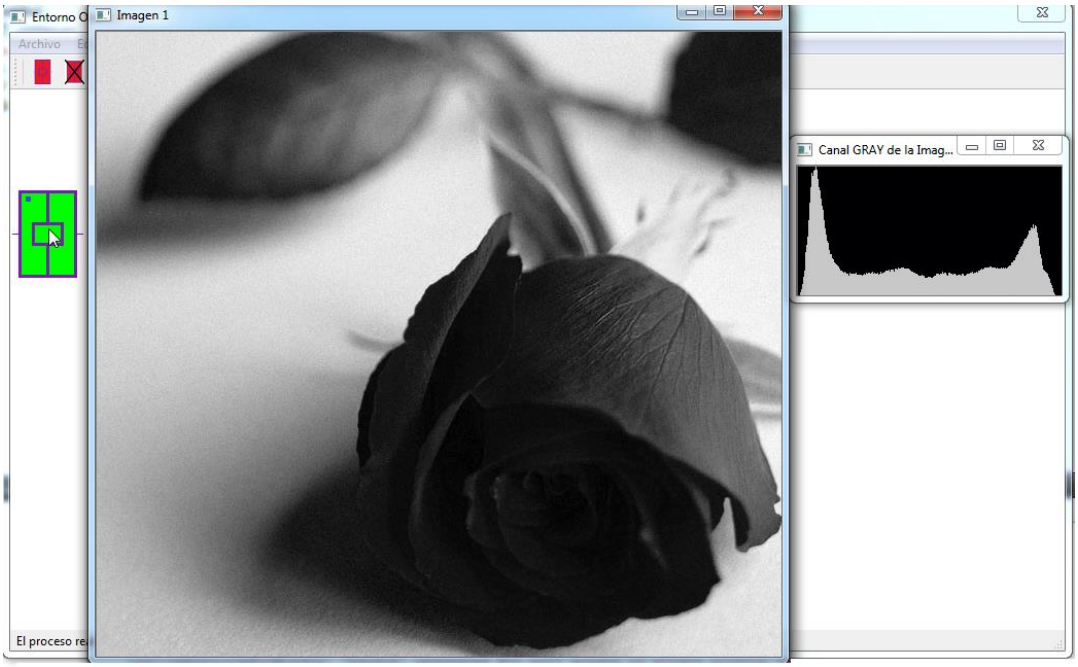


Figura 78. *Funciones OpenCV: función histograma para imagen de 1 canal.*

3.6.10. Selección de región del histograma

Esta función sirve para seleccionar una región de pixeles de un canal de una imagen. Para ello antes de realizar esta función, el usuario debe haber visualizado el histograma del canal deseado. Como se comentaba en el capítulo anterior, el histograma está formado por 255 columnas, y estas columnas son la información que le interesa al usuario ya que elegirá columnas que le interesan seleccionando el rango que comprenda las mismas.

Esta función al igual que el histograma detecta los canales de la imagen. Esto se debe a que el usuario solo puede elegir una región de interés de un solo canal. El valor mínimo del rango es 0 y el máximo 255.

Esta función se suele utilizar con conversiones de colores como HSV o HLS ya que dichas conversiones, sirven para detectar mejor zonas de colores como por ejemplo el color rojo.

Para elegir el canal de la imagen, la función utiliza su propio widget personalizado que varía en función de si la imagen tiene tres canales o solo uno. En la figura 79 se puede observar el widget de casa caso:

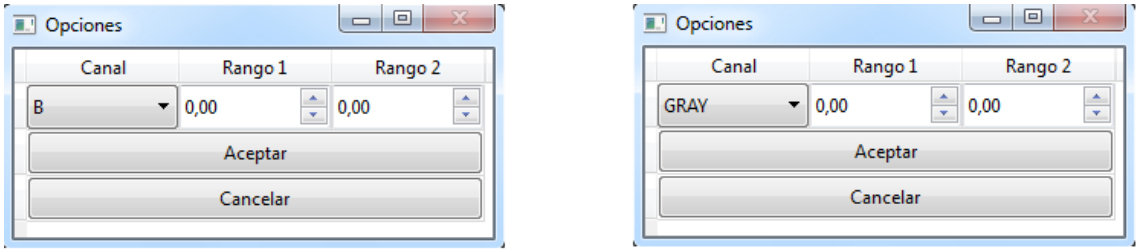


Imagen de 3 canales

Imagen de 1 canal

Figura 79. ***Funciones OpenCV: widgets personalizados de la función selección de rango.***

Como se puede observar en figura 79, la imagen de un canal nos aparece la opción GRAY ya marcada que es la única opción que se podrá marcar.

En los dos casos, Rango 1 es el rango inicial y el Rango 2 es el rango final.

En el capítulo 4 se realiza un ejemplo donde se puede entender bien cómo funciona esta herramienta y cuáles serían sus posibles usos.

3.6.11. Operaciones booleanas

Esta función sirve para complementar a la función del subcapítulo anterior (Selección de rango del histograma). Su propósito es realizar las operaciones AND y OR entre las imágenes que seleccione el usuario. El entorno solo soporta que las operaciones se hagan por canales, es decir, que la operación AND y OR se realiza con un canal de una imagen, por lo que si tenemos una imagen de tres canales, el widget personalizado le pide al usuario que seleccione el canal y si la imagen es de un canal, el widget personalizado ya selecciona dicho canal. En la figura siguiente se muestran los dos widgets.

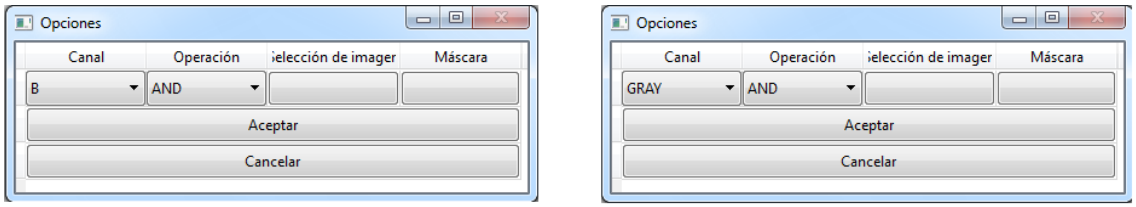


Figura 80. ***Funciones OpenCV: widgets personalizados de la función operaciones booleanas.***

Si se observa la figura 80 se puede ver que hay dos opciones:

- Selección de imagen.
- Máscara.

Para que el usuario realice la operación booleana necesita otra imagen a parte de la imagen de entrada que ya tiene el módulo, por lo que con la opción selección de imagen el usuario puede seleccionarla. La opción máscara es para que cuando se realice la operación booleana, el usuario pueda poner una máscara a la imagen resultante.

Cuando el usuario elige una imagen en estos dos casos, el widget le avisa mediante un mensaje escrito en el propio widget. Si la carga de la imagen se ha realizado correctamente, el mensaje es OK sino sale ERROR. Véase en la figura siguiente:

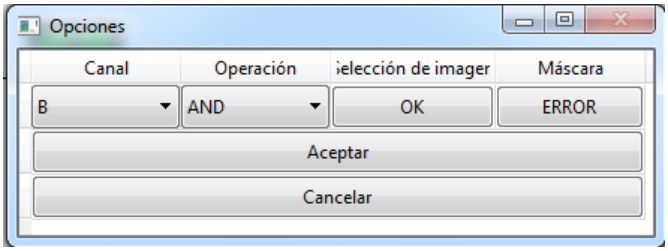


Figura 81. ***Funciones OpenCV: aviso del widget personalizado de la función operaciones booleanas.***

En el capítulo 4 se realiza u ejemplo donde se puede comprender y ver toda la versatilidad de esta función.

ENTORNO DE PROGRAMACIÓN GRÁFICO OPENCV

Capítulo 4. Ejemplo

4. Ejemplo

En esta capitulo se va a realizar un ejemplo con las funciones más complicadas de entender para que su compresión sea mucho mejor. En el ejemplo se utilizaran las funciones conversión de colores, histograma, selección de región del histograma y operaciones booleanas. En el ejemplo se va a suponer que un usuario quiere realizar una conversión de una imagen a formato HSV para luego poder separar los pétalos de la rosa del resto de la imagen. Dicha imagen se muestra en la figura 82:

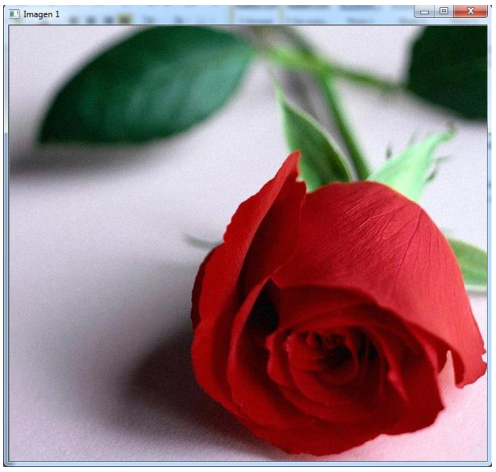


Figura 82. *Ejemplo: Imagen 1.*

La imagen que carga tiene tres canales y está en formato BGR. El usuario tiene que convertir la imagen del formato BGR al formato HSV para detectar el color rojo, es decir, los pétalos de la rosa. Esto se debe a que la intención final es separar los pétalos de la rosa del resto de la imagen.

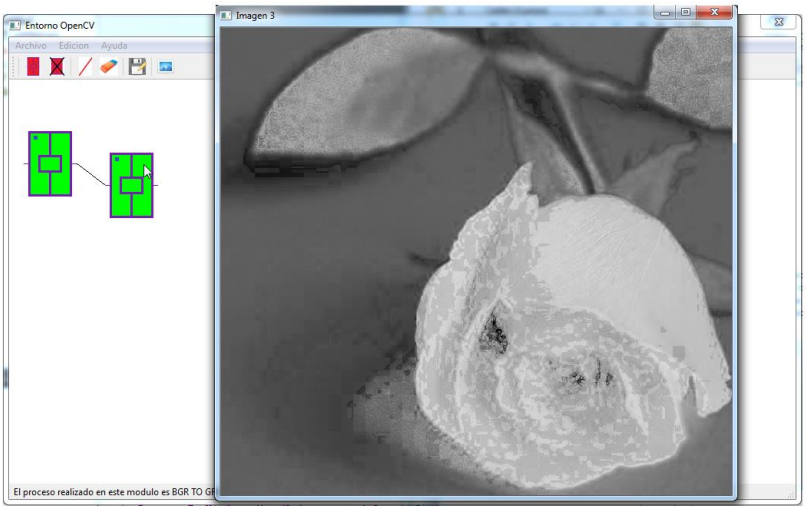


Figura 83. *Ejemplo: Imagen 3.*

En la imagen 3 (figura 83) se puede ver la conversión a GRAY de la imagen 2 (imagen en formato HSV). En ella se puede apreciar la discriminación de los pétalos, esta información se puede ver gracias al canal H (Hue) de la imagen 2.

El paso siguiente es ver el histograma de la imagen 2. En el formato HSV, el canal que tiene la información de los pétalos de la rosa es el H (Matiz) ya que en los canales S (saturación) y V (valor) no se encuentra la información de los tonos rojos. El histograma del canal H de la imagen 2 se puede ver en la siguiente figura:

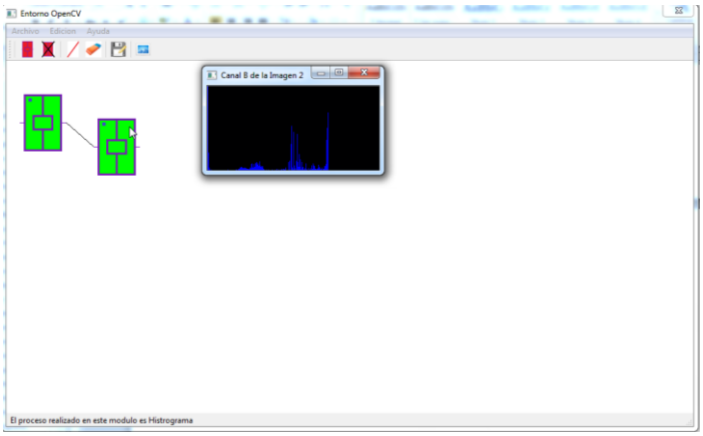


Figura 84. *Ejemplo: Histograma del canal H de la Imagen 2.*

Si se observa la imagen, en el título del histograma pone "Canal B de la Imagen 2". Esto se debe a que el canal B es el equivalente al canal H, es decir, el B es equivalente al H, el G es equivalente al S y el R es equivalente al V.

El siguiente paso es buscar en el histograma las regiones de interés. El tono se mide como el ángulo alrededor del eje S (ver figura 58). El rojo se sitúa a 0º, el verde a los 120º y el azul a los 240º. Los colores complementarios son aquellos que se encuentren a 180º del señalado. Teniendo en cuenta esta indicación se sabe que la información de los tonos rojos se va a encontrar al principio y al final del histograma que es donde vuelve a empezar los tonos rojos. Las regiones de interés que se van a seleccionar son las mostradas en la figura siguiente:

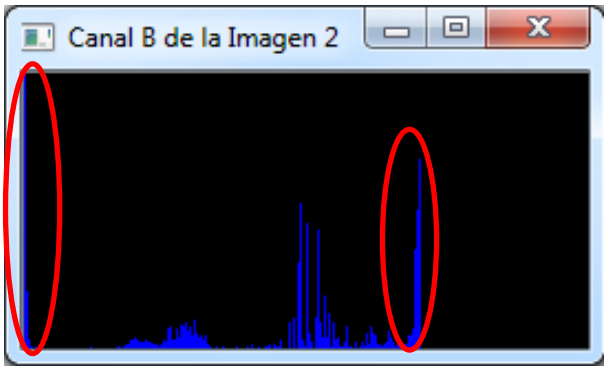


Figura 85. *Ejemplo: Zonas de interés de la Imagen 2.*

Al ver este histograma se puede ver que el usuario deberá usar la función selección de región del histograma y cuando seleccione los dos rangos, deberá unir las dos imágenes creadas de los rangos con la operación booleana OR para que no se pierda información.

Para seleccionar las zonas de interés, sabemos que el histograma tiene 255 columnas, por lo que si nos finamos en la zona del principio de la figura 85, podemos ver que el rango de esa información se encuentra desde la columna 0 hasta la columna 5 más o menos. Se pueden usar decimales si se quiere, es decir, en vez de 5 se puede poner 5,17. Este rango se llamará zona 1.

En la otra zona se puede ver que el rango de las columnas se encuentra desde la columna 172 a 190. Todas estas suposiciones se realizan con la estimación oportuna del usuario. Este rango se llamará zona 2.

Solo que utilizar la función de selección de rango del histograma, por lo que una vez usada, se pueden ver los resultados en la figura 86.

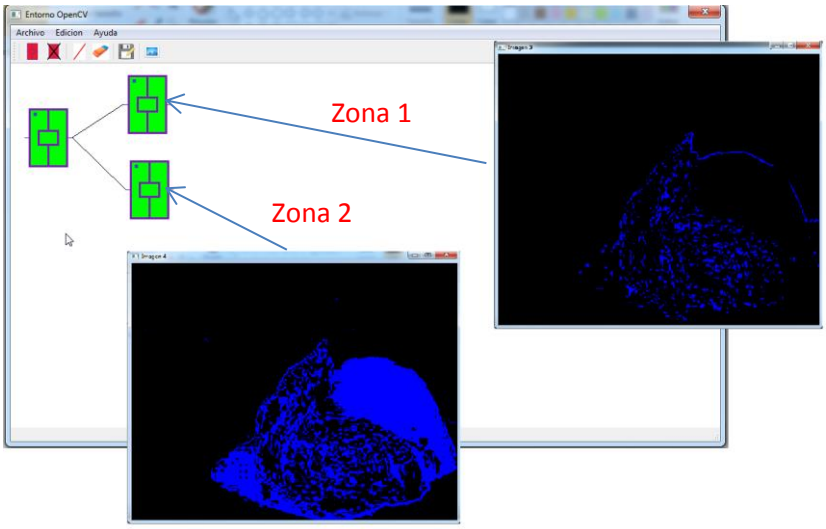


Figura 86. *Ejemplo: Zona 1 y 2.*

El último paso que le que le quedaría al usuario por realizar es usar la función de operaciones booleana ya que necesita unir estas dos imágenes y para ello la operación que se va a seleccionar es la OR. Para ello una de las dos imágenes se tiene que guardar.

Esto se debe a que la opción operación booleana se realiza a la imagen de entrada del módulo y todos los módulos solo pueden tener una entrada, por lo que se tiene que utilizar un módulo que tenga de entrada una de las dos imágenes (imagen 3 o imagen 4) y la otra se seleccionará en el widget personalizado, por ello deberá guardar con anterioridad con la herramienta guardar imagen. En este caso guardaremos la imagen 3 pondremos de entrada al módulo la imagen 4. El resultado de esto se puede ver en la siguiente figura:

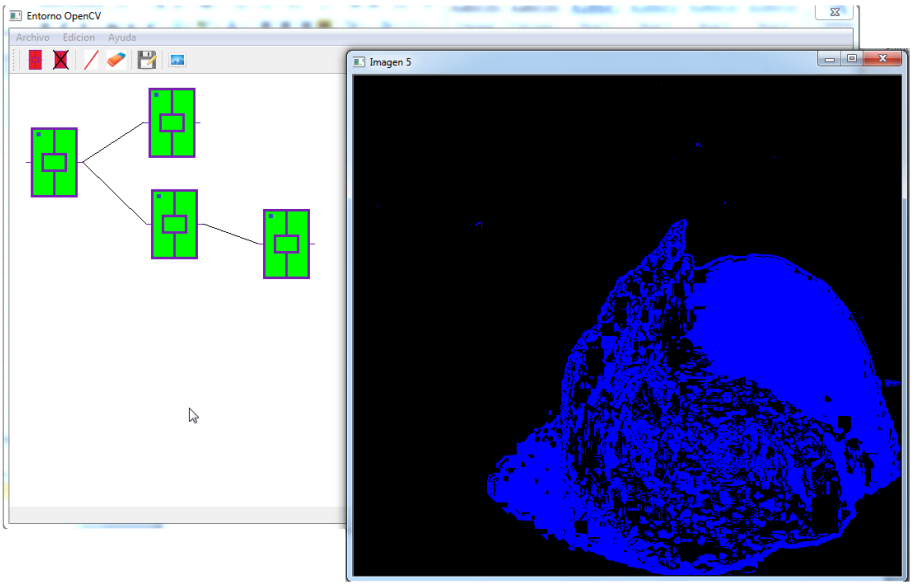


Figura 87. *Ejemplo: operación booleana OR.*

En la figura 87 se puede ver como solo se tiene el pétalo y se ha juntado la información de las dos imágenes en una, la imagen 5. Para el mismo proceso pero con la máscara quedaría de la siguiente forma:

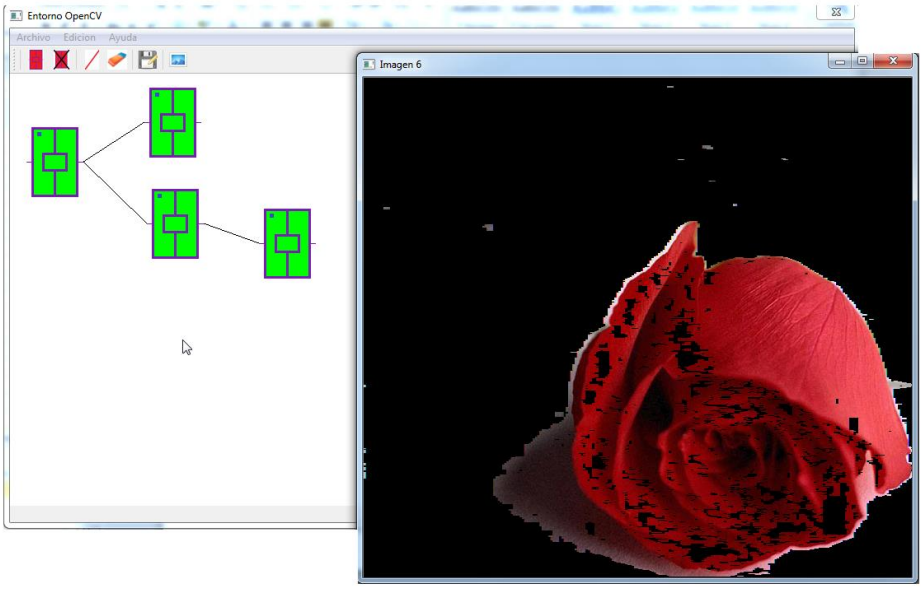


Figura 88. *Ejemplo: operación booleana OR con máscara.*

En la figura 88 se puede ver que no sale los pétalos de la rosa completamente, pero eso se debe a la precisión en la función selección de rango del histograma y a que dependiendo de la imagen la conversión HSV puede detectar mejor o peor las zonas rojas.

ENTORNO DE PROGRAMACIÓN GRÁFICO OPENCV

Capítulo 5.
Conclusiones y trabajo futuro

5. Conclusiones y trabajo futuro

La realización de este proyecto de fin de grado ha servido para aprender sobre el uso de las librerías OpenCV y sus diferentes tipos de aplicaciones que se pueden llegar a conseguir.

Este tipo de aplicaciones pueden ser muy diversas, desde aplicaciones complejas de seguridad hasta aplicaciones básicas, como es el caso de este proyecto. En el caso del proyecto realizado, se puede decir que puede llegar a ser una herramienta muy útil para los programadores de las librerías OpenCV. Este entorno, realiza tratamientos muy básicos a las imágenes, pero la idea es dejar establecida una base robusta para su desarrollo futuro.

En el futuro, se puede pensar sobre todo en la implementación de muchas más funciones, pero antes de la implementación de funciones nuevas, se puede pensar en mejorar las existentes. La manera de mejorar las funciones existentes, es añadiendo más control al usuario, es decir, introduciendo más opciones en las funciones de manera que se puedan realizar tratamientos más concretos. Un ejemplo de ello, es añadir al módulo la opción de que pueda tener múltiples entradas, y que se pueda seleccionar cada una de ellas en función del gusto del usuario. Otro ejemplo es la ampliación de la función histograma de manera que pueda soportar imágenes de cuatro canales.

Gracias al trabajo realizado bajo el entorno Qt Creator, el alumno ha podido conocer la herramienta Qt Designer y aprender cómo utilizar las herramientas que proporciona. Un ejemplo de estas herramientas son las señales y slots, que permiten al programador ahorrar código y tiempo. Sobre todo, el alumno ha aprendido el diseño e implementación de las forms de Windows que ha sido la base para conseguir realizar este proyecto. Gracias a ello, el alumno puede tener una visión global de la programación que hay detrás de un trabajo de este tipo.

Finalizando con la conclusión, se puede decir que el alumno se ha familiarizado más con este tipo de programación, proporcionándole una educación más práctica en la recta final de su carrera.

ENTORNO DE PROGRAMACIÓN GRÁFICO OPENCV

Capítulo 6. Presupuesto

6. Presupuesto

6.1. Coste de material

A continuación se exponen los gastos relativos a materiales y equipos empleados en el desarrollo del proyecto.

Código	Unidad	Descripción	Cantidad	Coste unidad	Coste total
103	PC	Ordenador con Windows 7	1	699,00 €	699,00€
124	OpenCV	Librería OpenCV	1	0	0
138	Webcam	Webcam Hercules 720p	1	29,60 €	29,60 €
			TOTAL PARTIDA		728,6€

Tabla 1. *Coste de material.*

6.2. Coste de personal

En la siguiente tabla se recoge el gasto correspondiente del personal necesario para realizar el proyecto.

Código	Unidad	Descripción	Cantidad	Coste unidad	Coste total
510	Ingeniero industrial electrónico	Ingeniero industrial para la investigación y desarrollo del proyecto: Entorno de programación gráfico en Open CV	3 meses	1600,00€/mes	4800,00€
			TOTAL PARTIDA		4800€

Tabla 2. *Coste de personal.*

6.3. Resumen del presupuesto

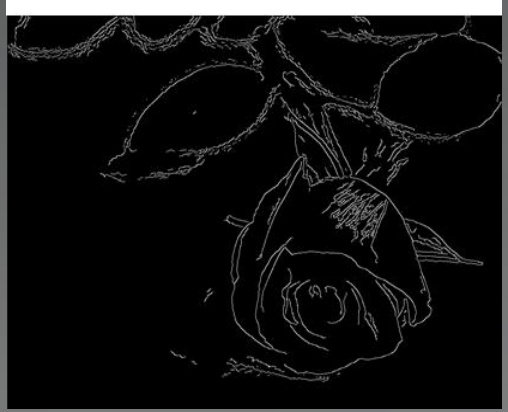
Finalmente se concluye con un cálculo aproximado del coste total necesario para desarrollar el proyecto.

Código	Unidad	Cantidad	Coste
100	Coste material y equipos	1	728,60€
500	Coste de personal	1	4800,00€
		TOTAL PARTIDA	5528,6€

Tabla 3. *Resumen del presupuesto.*

ENTORNO DE PROGRAMACIÓN GRÁFICO OPENCV

Capítulo 7. Bibliografía



7. Bibliografía

[1] A. de la Escalera, Visión por computador. Fundamentos y métodos., 2001.

[2] A. Kaebler & G. Bradski, Learning Open CV: Computer vision in C++with the OpenCV library., 2013

[3] <<opencourseware>> [En línea]. Avaliable: http://ocw.uc3m.es/ingenieria-de-sistemas-y-automatica/sistemas-de-percepcion [Último acceso Agosto 2013]

[4] <<GUI>> [En línea]. Avaliable: http://doc-snapshot.qt-project.org/qtcreator-extending/qtcreator-api.html [Último acceso Agosto 2013]

[5] <<UI>> [En línea]. Avaliable: http://doc-snapshot.qt-project.org/qtcreator-extending/qtcreator-api.html [Último acceso Agosto 2013]

[6] <<QWidget>> [En línea]. Avaliable: http://doc-snapshot.qt-project.org/qtcreator-extending/qtcreator-api.html [Último acceso Agosto 2013]

[7] <<QMainWindows>> [En línea]. Avaliable: http://doc-snapshot.qt-project.org/qtcreator-extending/qtcreator-api.html [Último acceso Agosto 2013]

[8] <<API OpenCV 2.4.4>> [En línea]. Avaliable: http://docs.opencv.org/2.4.4/modules/refman.html [Último acceso Agosto 2013]

[9] <<Visión artificial >> [En línea]. Avaliable: http://www.revistasbolivianas.org.bo/scielo.php?pid=S1997-40442008000200046&script=sci_arttext [Último acceso Septiembre 2013]

[10] D. Mery and F. Pedreschi. Segmentation of colour food images using a robust algorithm. Journal of Food Engineering, 2004. (accepted April 2004).

[11] <<Titere>> [En línea]. Avaliable: http://ciclope.fi.upm.es/tools/titere/index_es [Último acceso Agosto 2013]

[12] <<LabVIEW>> [En línea]. Avaliable:http://www.ni.com/gettingstarted/labviewbasics/environment.htm [Último acceso Agosto 2013]

[12] <<MatLab>> [En línea]. Avaliable: http://www.mathworks.es/products/matlab [Último acceso Agosto 2013]

[13] <<OpenCV>> [En línea]. Avaliable: http://opencv.willowgarage.com/wiki/ [Último acceso Septiembre 2013]

[14] <<Threshold>> [En línea]. Avaliable: http://docs.opencv.org/doc/tutorials/imgproc/threshold/threshold.html [Último acceso Septiembre 2013]

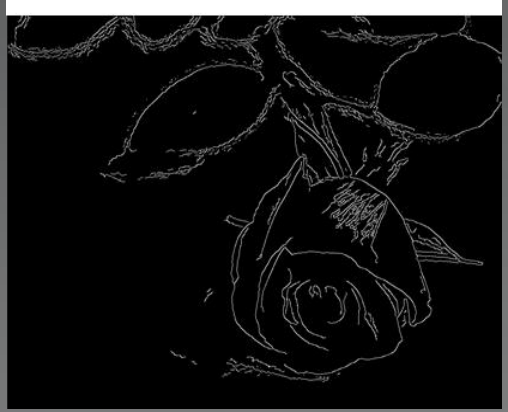
[15] <<HSV color space>> [En línea]. Avaliable: http://www.mathworks.es/es/help/images/converting-color-data-between-color-spaces.html [Último acceso Septiembre 2013]

[16] Gar A. Bergstedt (April 1983). US patent 4694286, "Apparatus and method for modifying displayed color images". Filed 1983-04-08. Issued 1987-09-15. Assigned to Tektronix, Inc.

[17] <<Histogram in OpenCV>> [En línea]. Avaliable: http://opencv- code.com/tutorials/drawing- histograms-in-opencv/ [Último acceso Septiembre 2013].

ENTORNO DE PROGRAMACIÓN GRÁFICO OPENCV

Anexo I.



8. Anexo I: anexión de las librerías OpenCV en Qt Creator

Para poder utilizar las librerías OpenCV en Qt Creator primero han de ser compiladas con CMake. Según el sistema operativo que se tenga (Windows, Linux) la compilación de las librerías se hace de una forma u otra. El propósito de este tutorial es sólo enseñar cómo se anexionan las librerías OpenCV en Qt Creator una vez que están compiladas.

En el Qt Creator se pueden hacer varios tipos de proyectos diferentes. En todos los proyectos que se crean hay un archivo que tiene la extensión .pro.

Este archivo es el director del proyecto, es decir, contiene información relacionada con los archivos que se incluyen en el proyecto, pasos de generación personalizada y los ajustes para el funcionamiento de las aplicaciones. En la figura 89 se puede ver el archivo del entorno gráfico OpenCV (untitled.pro).

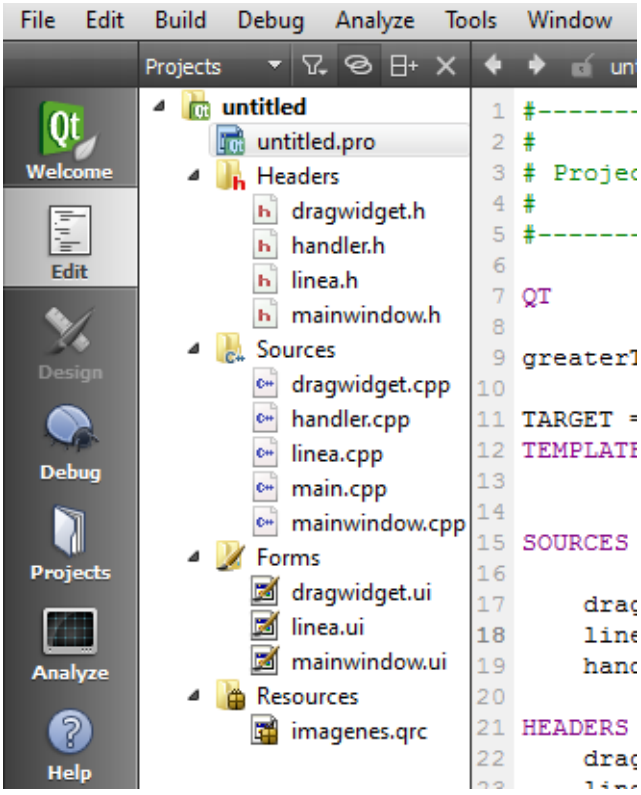


Figura 89. *Archivo .pro del entorno gráfico OpenCV.*

Una vez localizado el archivo de proyecto se escribe en el mismo la dirección de los archivos binarios de las librerías OpenCV. Los archivos binarios son los que se han creado una vez que se ha utilizado la aplicación CMake. Se escribe lo siguiente en el archivo .pro:

```
INCLUDEPATH += C:\\opencv244\\opencv-build\\install\\include
INCLUDEPATH += C:\\opencv244\\opencv-build\\install\\bin
LIBS += -LC:\\opencv244\\opencv-build\\install\\lib \
   -lopencv_core244.dll \
   -lopencv_highgui244.dll \
   -lopencv_imgproc244.dll \
   -lopencv_features2d244.dll \
   -lopencv_calib3d244.dll
```

Como se puede ver la versión de las librerías en este caso es la 2.4.4. pero para cualquier otra versión de las librerías sólo hay que cambiar 244 por el número de la versión. Por ejemplo, para la versión OpenCV 2.3.1. quedaría de la siguiente forma:

```
INCLUDEPATH += C:\\opencv231\\opencv-build\\install\\include
INCLUDEPATH += C:\\opencv231\\opencv-build\\install\\bin
LIBS += -LC:\\opencv231\\opencv-build\\install\\lib \
   -lopencv_core231.dll \
   -lopencv_highgui231.dll \
   -lopencv_imgproc231.dll \
   -lopencv_features2d231.dll \
   -lopencv_calib3d231.dll
```

Las líneas INCLUDEPATH marcan los directorios donde se encuentran los archivos binarios. En cada ordenador esta ruta varía en función del usuario. El resultado es el siguiente:

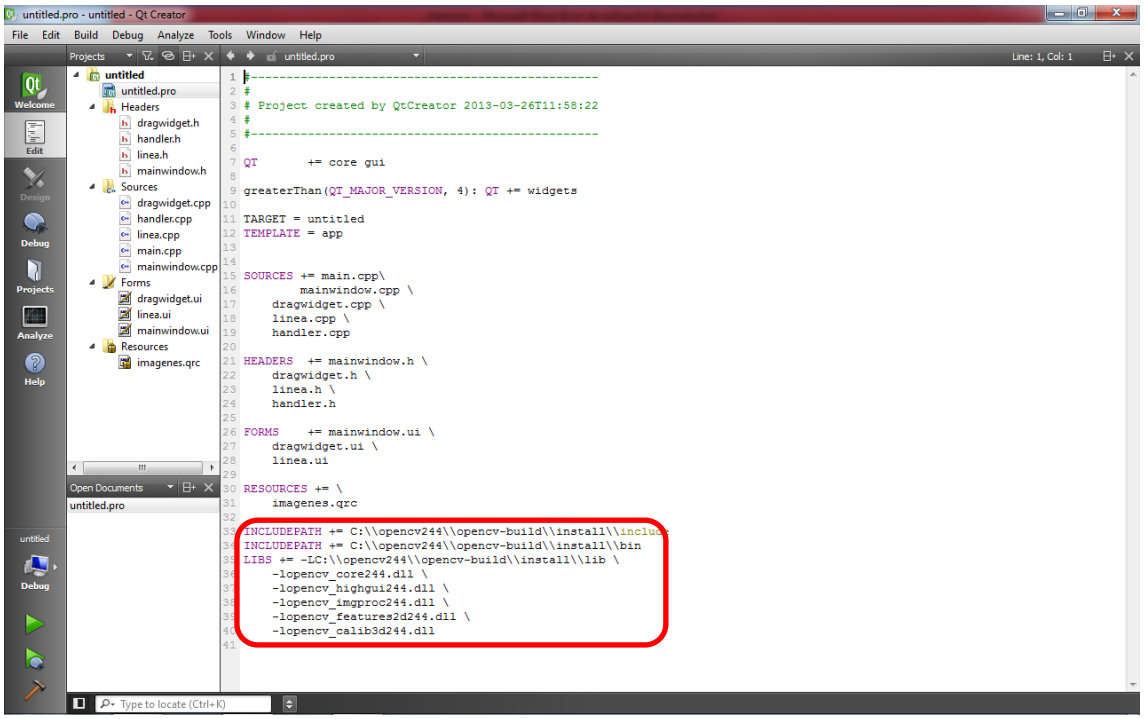


Figura 90. ***OpenCV en el archivo .pro.***

Una vez realizado este paso ya sólo queda incluir en los archivos Header (archivos con extensión .h) del proyecto las siguientes líneas:

```
#include<opencv2/core/core.hpp>
#include<opencv2/highgui/highgui.hpp>
#include<opencv2/opencv.hpp>
```

Estas tres líneas se tienen que incluir en todos los Headers del proyecto. Un ejemplo de ello se puede ver en la figura siguiente:

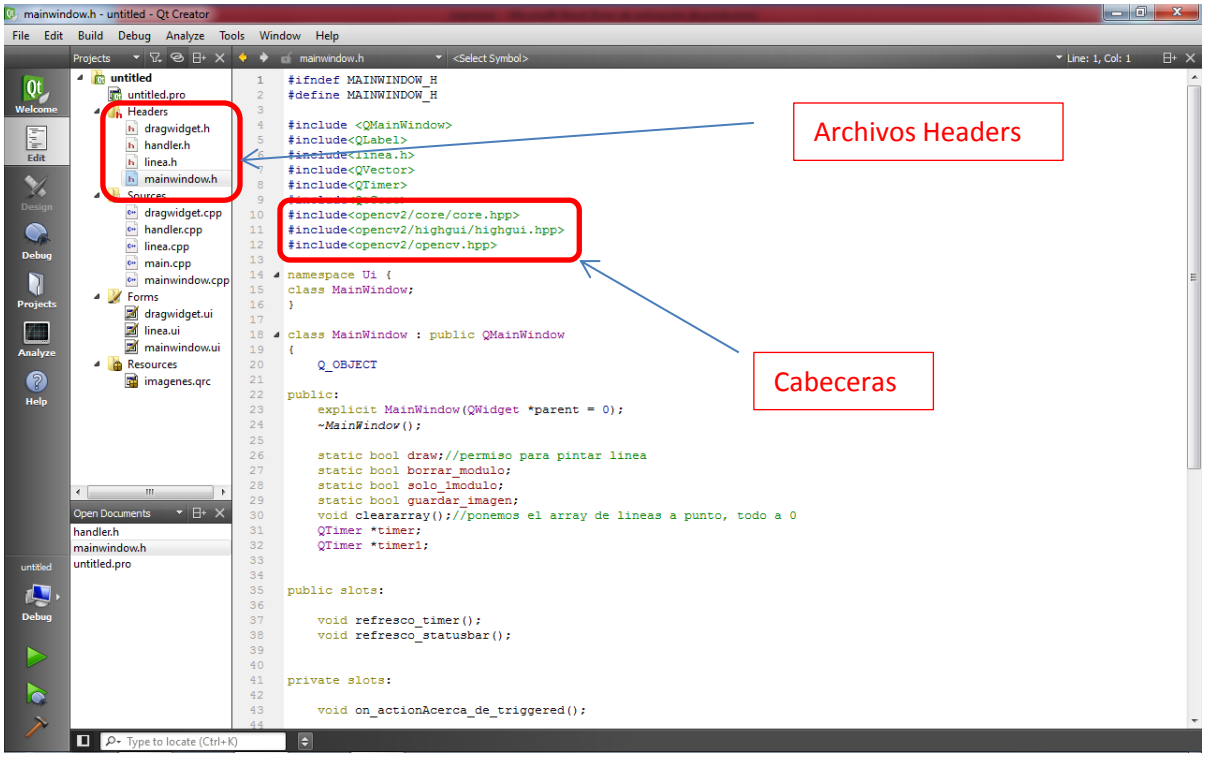


Figura 91. ***Cabeceras de las librerías OpenCV.***

Una vez realizado este último paso ya se pueden usar las librerías OpenCV en Qt Creator.