Proyecto Fin de Máster
Máster Universitario en Ingeniería Electrónica, Robótica y Automática

Entorno de programación modular e interactivo para visión automática

Autor: Jesús Carrión Rísquez
Tutor: Manuel Ruiz Arahal

Dpto. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2020

Proyecto Fin de Máster
Máster Universitario en Ingeniería Electrónica, Robótica y Automática

Entorno de programación modular e interactivo para visión automática

Autor:
Jesús Carrión Rísquez

Tutor:
Dr. Manuel Ruiz Arahal
Profesor titular

Dpto. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2020

Trabajo Fin de Máster: Entorno de programación modular e interactivo para visión automática

Autor: Jesús Carrión Rísquez

Tutor: Manuel Ruiz Arahal

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

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2020

El Secretario del Tribunal

A mi familia

A mis compañeros y profesores

A todos los que me han apoyado en este camino

Agradecimientos

En primer lugar, agradecer a mi tutor Manuel Ruiz Arahal, el haberme permitido desarrollar esta idea como mi Trabajo Fin de Máster. Desde el primer momento me ayudó a orientarlo y me dio indicaciones para poder llevarlo a buen puerto.

Y, por último y muy especialmente,a mi familia, por su eterna confianza y sustento, por hacer que todas las metas que afronto en mi vida sean más fácilesde alcanzar.

Finalmente, dar las gracias a todos aquellos que me apoyaron en este camino.

.

Jesús Carrión Rísquez

Sevilla, 2020

Resumen

En el presente proyecto se ha desarrollado una aplicación de escritorio que permite un rápido análisis y desarrollo de algoritmos centrados en el campo de la visión por ordenador. El proyecto busca poder sustituir la programación a través de código por la programación a través del empleo de bloques.

Para alcanzar tal fin, la aplicación crea una capa de abstracción alrededor de la librería de OpenCV, permitiéndonos emplear, combinar y probar distintas configuraciones de varias de las técnicas de tratamiento de imágenes que dispone en la librería. La aplicación pone a nuestra disposición un conjunto de bloques, los cuales encapsulan diferentes funcionalidades tales como: reproducción de video, operaciones de *thresholding*, operaciones morfológicas etc. Estos bloques tienen la capacidad de poder conectarse unos con otros. De esta forma se puede ir desarrollando diferentes algoritmos de manera visual.

Gracias al empleo de esta aplicación se puede reducir enormemente el tiempo de desarrollo de algoritmos, incluso permite a las personas menos expertas en este campo, un rápido desarrollo.

Abstract

In the present project, a desktop application has been developed that allows a quick analysis and development of algorithms focused on the field of computer vision. The project seeks to replace code-based programming with block-based programming.

To achieve this goal, the application creates an abstraction layer around the OpenCV library, allowing us to use, combine and test different configurations of several of the image processing technques available in the library. The application provides us with a set of blocks, which encapsulate different functionalities such as: video reproduction, thresholding operations, morphological operations, etc. These blocks have the ability to connect to each other. In this way, different algorithms can be developed visually.

Thanks to the use of this application the time of development of algorithms can be reduced enormously, even allowing the less expert people in this field, a fast development.

... -translation by deepl-

Índice

ÍNDICE DE FIGURAS

Notación

CPU*	Central Processing Unit
SQL.	Structured Query Language
API.	Application Programming Interfaz
GUI	Graphical User Interface
IDE.	Integrated Development Enviorement
GPL	General Public License
BSD	Berkeley Software Distribution
LGPL	Lesser General Public License
JSON	Java Script Object Notation
CUDA	Compute Unified Device Architecture
Add-on	Añadido
XML	eXtensible Markup Language

1 INTRODUCCIÓN

En la Ciencia se suele partir de una/as ideas iniciales que buscan modelar algún fenómeno. Durante todo este proceso se acaba llegando a la creación de una teoría, la cual se suele expresar en un conjunto de fórmulas o procedimientos. En última instancia siempre se busca poder verificar o comprobar esta teoría empleando para ello cálculos numéricos.

La obtención manual de estos resultados numéricos solía ser engorrosa, difícil y lenta. Por ello se acabaron creando los lenguajes de programación gracias a los cuales facilitan todo este proceso.

1.1 Generaciones de Lenguajes de Programación

A lo largo de la historia, los lenguajes de programación han tenido multitud de aplicaciones. La más evidente ha sido el facilitar la obtención de cálculos numéricos. Se han desarrollado una gran multitud de lenguajes de programación para alcanzar tal fin. Podemos hablar de la existencia de Generaciones de Lenguajes de Programación entre las cuales tenemos:

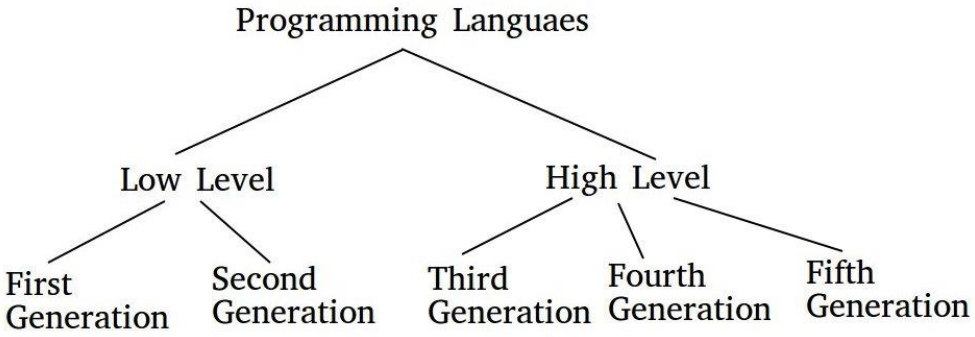


Figura 1-1. Clasificación Lenguajes de Programación

1.1.1 Primera Generación

Esta generación llega hasta principios de los 50s. Los programadores escribían su código empleando el lenguaje máquina, es decir, usando simplemente ceros y unos. Esto provocaba que se necesitase un conocimiento preciso de la memoria y el juego de instrucciones asociados a esa máquina en específico. Además, por aquel entonces no existían compiladores, ensambladores ni "Debuggers".

La principal Ventaja de esta generación era que el código podía correr muy rápido y de una forma muy eficiente, debido principalmente porque era ejecutado directamente por la CPU. Sin embargo, la principal desventaja que presentaba esta generación era la gran dificultad de detectar y corregir fallos en el código, además resultaba muy difícil, por no decir imposible, que una persona fuese capaz de interpretar el código. [1] [2] [3]

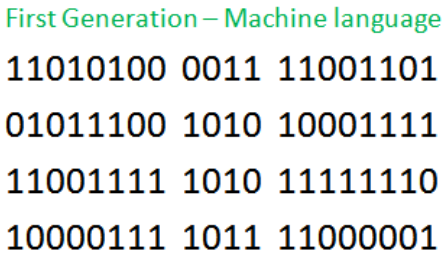


Figura 1-2. Ejemplo Código Máquina.

1.1.2 Segunda Generación

Esta generación surge a mediados de los 50s y se la conoce como la generación de los lenguajes ensambladores. Aunque el código sigue siendo muy dependiente de la máquina empleada, se incorporan símbolos, los cuales se emplean para crear líneas de instrucciones que se corresponden directamente a un comando en el procesador.

El código creado puede ser leído de manera más fácil por los programadores. Sin embargo, se necesita de un sistema que convierta este código a código máquina. Los encargados de realizar ese proceso son los llamados ensambladores. [2] [3]

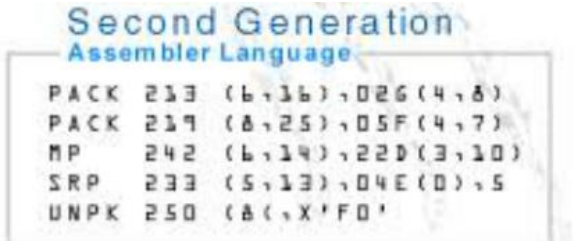


Figura 1-3. Ejemplo de Código Ensamblador

1.1.3 Tercera Generación

Esta generación surge a finales de los 50s. Se la considera la generación que inició el concepto de lenguajes de alto nivel. Se caracteriza por volverse mucho más independiente de la máquina empleada. Una vez creado el código, puede llevarse a otra máquina para ejecutar dicho código, previa re-compilación en la mayoría de los casos. Además, resulta ser mucho más amigable de cara a los programadores gracias a que introduce características como: estructuras de datos y control, bucles, condicionales, clases etc. Esto se traduce en que una línea de código en esta generación, puede llegar a traducirse en varias líneas de código en las anteriores generaciones, disminuyendo así el tiempo de desarrollo.

Los lenguajes de esta generación son más abstractos que en las anteriores y se necesita emplear compiladores o intérpretes que conviertan el nuevo código a código máquina.

La mayoría de los lenguajes empleados hoy en día pertenecen a esta generación. Entre los lenguajes más famosos tenemos a: C/C++, COBOL, Fortran, Java, C#, Python, etc. [2] [3]

```
  x += playerSpeed
  player.setx(x)

def fire_bullet():
  #Declare bulletState
  global bulletState
  if bulletState == "ready":
    os.system("afplay laser.wav&")
    bulletState = "fire"
    x = player.xcor()
    y = player.ycor()
    bullet.setposition(x, y + 10)
    bullet.showturtle()

def isCollision(t1, t2):
  distance = math.sqrt(math.pow(t1.xcor()-t2.xcor(),2)+math.pow(t1.ycor()-t2.ycor(),2))
  if distance < 15:
    return True
  else:
    return False

#Create Keyboard Bindings
turtle.listen()
turtle.onkey(move_left, "Left")
turtle.onkey(move_right, "Right")
turtle.onkey(fire_bullet, "space")

#Main Game Loop
while True:
  for enemy in enemies:
    #Move the enemy
    x = enemy.xcor()
    x += enemySpeed
    enemy.setx(x)

    #Move the enemy back and down
    if enemy.xcor() > 280:
```

Figura 1- 4. Ejemplo Código Tercera Generación.

1.1.4 Cuarta Generación

Los lenguajes de esta generación se caracterizan en que simplemente hay que indicarles lo que se quiere conseguir, pero no se le indica cómo se debe hacer. Es el propio lenguaje el que se encarga de realizar esa acción. Suelen estar más enfocados en resolver un problema, reduciendo así el coste y esfuerzo en el desarrollo de software.

Los principales campos en los cuales tenemos este tipo de lenguajes son: peticiones a bases de datos, generadores de reportes, manipulación de datos, creadores de Interfaces gráficas, optimización matemática y desarrolladores web.

Ejemplos de estos lenguajes de programación serian: SQL, ProLog, VisualBasic, QTCreator etc.

Cabe aclarar que la línea que separa a lenguajes de la 3 y la 4 Generación no esta muy bien definida. [2] [3]

1.1.5 Quinta Generación

Esta generación resulta ser incluso más abstracta que la anterior. Se pueden encontrar varias definiciones tales como:

"Any programming language based on problem-solving using constraints given to the program, rather than using an algorithm written by a programmer." [4]

"Is programming that basically uses a visual or graphical development interface to create source language and also known as (Artificial Intelligence). It was basically amid and focused on developing a device in which it could handle and respond to natural language and capable of self learning and self organization." [3]

En el caso de la Cuarta y la Quinta Generación, la línea que las separas es casi inexistente. Por lo que no hay un criterio generalizado para catalogarlas.

1.2 Evolución Natural de la Programación

Al final observamos que la forma de programar va evolucionando poco a poco a lo largo del tiempo, adoptando cada vez soluciones que se centran más en el diseño y desarrollo de la propia idea.

Esto lo podemos ver claramente con los lenguajes de la cuarta y quinta generación. Veamos algunos ejemplos para aclarar esta idea.

1.2.1 SQL

Un primer ejemplo, lo podemos ver en el manejo de las peticiones a las bases de datos. La forma en la cual se realizan estas peticiones es imponiendo un conjunto de restricciones en los resultados que se desea obtener. En ningún momento se le indica la manera en la cual debe buscar la información en la base de datos. Se tiene un ejemplo en la Figura 1-5.

```
SELECT "Pedidos"."IDPedido" AS "NumPedido",
"Pedidos"."Destinatario", "Pedidos"."FechaEnvío",
"Transporte"."TipoTransporte",
"Pedidos"."CiudadDestinatario" AS "Destino" FROM
"Pedidos", "Transporte" WHERE
"Pedidos"."TipoTransporte" =
"Transporte"."IDTransporte" AND
"Pedidos"."CiudadDestinatario" = 'madrid' AND
( "Pedidos"."TipoTransporte" = 1 OR
"Pedidos"."TipoTransporte" = 2 ) ORDER BY
"Pedidos"."Destinatario" ASC
```

Figura 1-5. Ejemplo de petición a una Base de Datos

1.2.2 Scracth

Es un proyecto creado por el Grupo Lifelong Kindergarten del MIT Lab. El objetivo del proyecto es ayudar a los pequeños a que aprendan los fundamentos de la programación de una forma interactiva y atractiva para ellos. Con este software se pueden crear sus propios juegos y animaciones, entre muchas otras cosas. El programa se ofrece de forma gratuita.

Figura 1-6. Ejemplo de Programa Scracth

1.2.3 Labview

Es un software de ingeniería de sistemas requerido para pruebas, medidas y control con acceso rápido a hardware e información de datos. Ofrece un enfoque de programación gráfica que ayuda al desarrollo de algoritmo de análisis de datos y visualización de los mismos. La utilización del programa requiere de la adquisición de una licencia. [5]

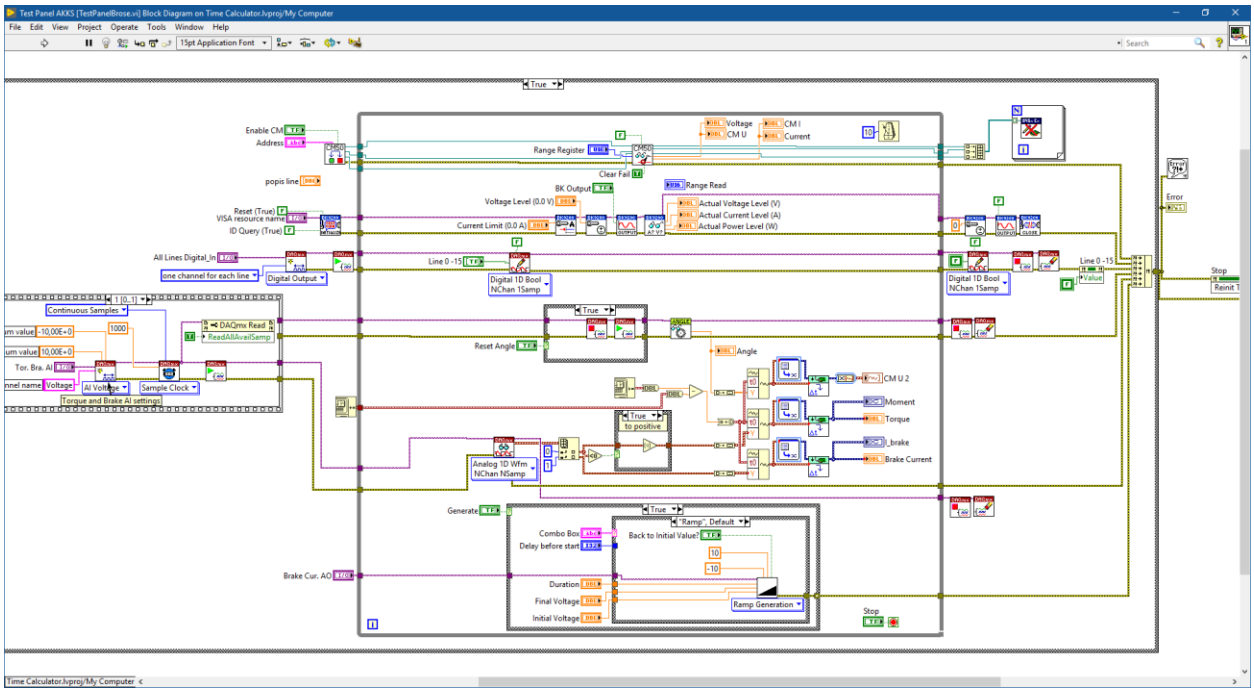


Figura 1-7. Ejemplo del Programa LabView

1.2.4 Simulink

Este software sirve para realizar el diseño y simulación de un sistema, de ahí su nombre "Simulink". El software hace uso de una metodología de programación gráfica, mediante el uso de bloques, para realizar el diseño del sistema. Las principales características de este software son: Modelación y Simulación de Sistemas, Realización de pruebas de forma anticipada y con frecuencia, además de poder generar código automáticamente.

Dispone de numerosos paquetes de software para una gran cantidad de ámbitos tales como: Comunicaciones Inalámbricas, Sistemas de Control, Procesamiento de Señales, Robótica etc. El software forma parte de Matlab y requiere de la compra de una licencia para su utilización. [6]

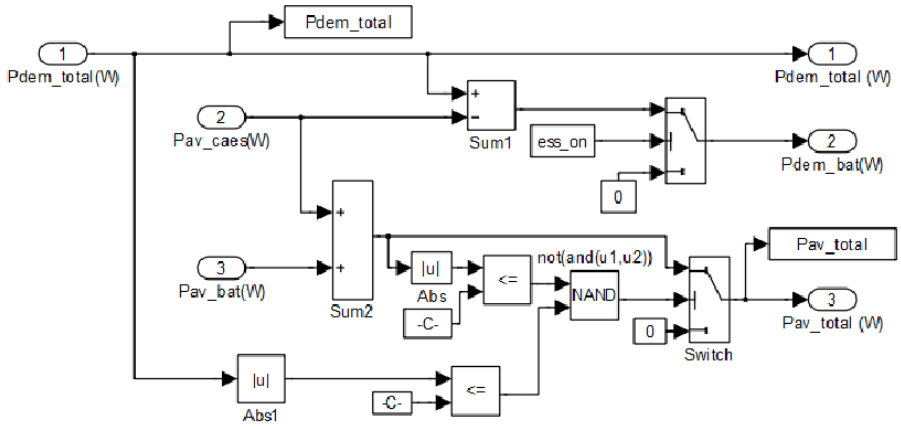


Figura 1-8 Ejemplo de Simulink

1.2.5 BluePrints – Unreal Engine 4

Unreal Engine es un motor gráfico empleado para el desarrollo de videojuegos. No es solo un motor gráfico, sino que es un entorno completo que pone a tu disposición todas las herramientas que necesitarías para poder realizar un videojuego completo.

En el desarrollo de videojuegos, concretamente la parte de programación, se puede realizar por medio de dos metodologías: creación de código manualmente o empleado el sistema de programación visual, denominado Blueprints. Es un sistema desarrollado por los creadores del propio motor gráfico, el cual nos da la posibilidad de poder desarrollar (programar) toda la lógica o partes empleando este método.

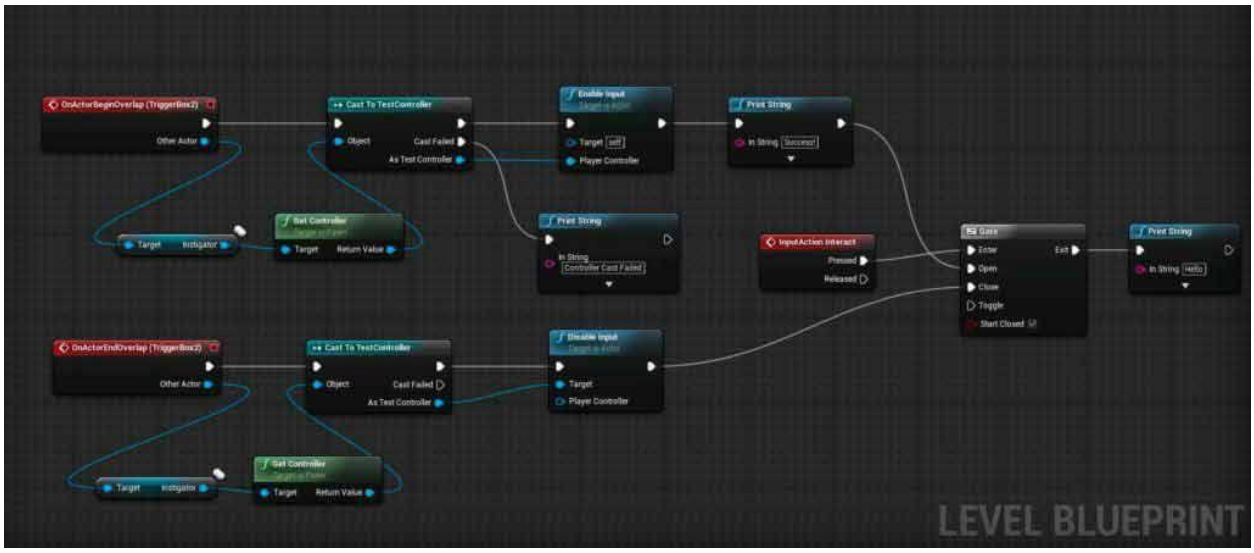


Figura 1-9. Ejemplo de uso del Sistema Blueprint.

1.3 Objetivos, Motivación y Alcance del Proyecto

A medida que los diseños se vuelven más y más complejos, se incrementa el tiempo para desarrollar una solución. Al final, para agilizar el proceso se van desarrollando nuevos lenguajes y software con el objetivo de facilitar y disminuir el tiempo de desarrollo.

Supongamos que somos unos expertos en el campo de la visión por ordenador, o cualquier otro campo, y queremos desarrollar nuevos algoritmos. Se quiera o no vamos a tener que meternos en el terreno de la programación para desarrollar nuestras ideas. Además, lo más seguro es que no seamos un experto en programación ni conozcamos en profundidad el funcionamiento de la librería/as para llevar a cabo nuestra idea.

Llegados a este punto, seguro que más de uno se ha preguntado porque no se ha creado una forma que nos permita saltarnos ese paso intermedio de programación y dedicarnos exclusivamente al diseño.

Aquí es donde entra en juego el proyecto que nos ocupa. Este proyecto tiene como objetivo crear una capa superior de abstracción para el empleo de las diferentes herramientas (librerías), facilitando de esta manera la utilización de las mismas. Esto al final se traduciría en un menor tiempo de desarrollo.

Para llevar a cabo el presente proyecto se ha partido de las diferentes ideas comentadas en el punto 1.2 y se ha desarrollado un nuevo programa que se queda con lo mejor de todos ellos. Se podría ver a esta aplicación como una mezcla entre Labview, Simulink y BluePrints, creada específicamente con el objetivo de facilitar y agilizar el diseño.

Por lo tanto, este trabajo tiene como objetivo la creación de una aplicación piloto que demuestre la viabilidad de esta forma alternativa de programación, además de mostrar el potencial de crecimiento de la misma.

Para mostrar las ventajas de esta forma alternativa de programación, es necesario elegir un campo de aplicación. Para el presente proyecto se ha escogido el campo de la visión por ordenador, debido a que precisamente en este campo es donde mejor se pueden apreciar las bondades de esta nueva metodología, ya que la mayoría de

herramientas disponibles son a nivel de programación. El problema con las herramientas actuales es que a medida que nos metemos en profundidad en su utilización se empiezan a volver más complicadas de aprender a usarlas. Al final, todo esto se traduce en tiempos de desarrollo elevados.

2 METODOLOGÍA

2.1 Software Empleado

Para la realización del siguiente proyecto, se ha utilizado las siguientes herramientas de Software.

- Librería de Qt y Entorno de Desarrollo QT Creator.
- Librería de Visión por ordenador. OpenCV.
- Librería Gestión Nodos. NodeEditor.
- Control de Versiones. Git y GitHub.
- Lenguaje de Programación. C++

Debido a que el siguiente proyecto presenta cierto nivel de complejidad, se ha considerado buena idea emplear un entorno de desarrollo para llevarlo a cabo. Aunque siempre es una buena práctica acostumbrarse a desarrollar empleando un entorno de desarrollo, debido a que estos entornos están pensados para facilitar el desarrollo de software.

Concretamente, los dos entornos de programación, más empleados son Visual Studio, Qt Creator y Eclipse (con su extensión para C++). También existen otros entornos como CodeBlock… pero se consideran, por lo menos bajo mi criterio, entornos menos profesionales que los mencionados anteriormente.

2.2 QT

Es una API gratuita y de Código abierto para el diseño de interfaces de usuario, conocidas en ingles por sus siglas GUI, así como aplicaciones multiplataforma tanto a nivel software como de hardware. Se puede emplear en entornos: Window, Linux, macOS, Android y Sistemas embebidos.

Actualmente "The Qt Company" se encarga de desarrollarla. Está disponible bajo licencia comercial como licencia de Código abierto (GPL 2.0, GPL 3.0, LGPL 3.0).

Esta escrita completamente usando el lenguaje de programación C++. Soporta la mayoría de los compiladores, tales como GCC, CLlang, Visual Studio Compiler y MinGW. Además, presenta módulos que nos permiten utilizar de una forma fácil el uso de otras librerías tales como: SQL, Json, XML, Threads etc.

Para hacernos una idea del potencial que presenta esta librería vamos a hacer una pequeña recopilación de distintos softwares que hacen uso de ella.

En el campo de Desktop UIs, podemos encontrarnos con proyectos como, KDE Plasma, LXQt, Lumina, Trinity Desktop, etc. Todos estos ejemplos son sistemas operativos, como ubuntu, en los cuales para su creación o parte de ella se ha empleado la librería de Qt.

En el campo de Embedded and Mobile UIs, podemos encontrarnos con algunos proyectos interesante, algunos de los cuales sin continuacion de servicio como Blackberry 10. Sin embargo, hay muchos otros proyectos que siguen con servicio, tales como Sailfish OS, Ubuntu Touch, Plasma Mobile, Tesla Model S(in-car UI), webOS, etc.

En el campo de las aplicaciones, nos podemos encontrar con algunos programas populares que hacen uso de

esta librería, tales como Maya, 3ds Max, EAGLE, Krita, Mathematica, Telegram, Virtual Box, VLC media Player, etc.

Finalmente, algunas empresas que hacen uso de la librería de Qt, entre las cuales podemos encontrarnos a: AMD, Blizzard Entertainment, BMW, Tesla, LG, Samsung, Siemens, HP, DreamWorks, LucasFilms, European Space Agency, etc. [7] [8]

2.2.1 Modulos QT

La librería de Qt se encuentra dividida en un conjunto de módulos, encargados de gestionar diferentes herramientas. Aunque existen un elevado número de módulos en QT, se pueden clasificar en dos grandes grupos: Essential and Addons. *Essential modules*, son los de uso más general y los que son utilizados en casi todas las aplicaciones. Add-On modules, son todos aquellos que se emplean para un propósito específico. En la Figura 10 se muestra una lista de parte de estos módulos.

Module	Description
Qt Core	The only required Qt module, containing classes used by other modules, including the meta-object system, concurrency and threading, containers, event system, plugins and I/O facilities.
Qt GUI	The central GUI module. In Qt 5 this module now depends on OpenGL, but no longer contains any widget classes.
Qt Widgets	Contains classes for classic widget based GUI applications and the QSceneGraph classes. Was split off from **QtGui** in Qt 5.
Qt QML	Module for QML and JavaScript languages.
Qt Quick	The module for GUI application written using QML2.
Qt Quick Controls	Widget like controls for **Qt Quick** intended mainly for desktop applications.
Qt Quick Layouts	Layouts for arranging items in **Qt Quick**.
Qt Network	Network abstraction layer. Complete with TCP, UDP, HTTP, SSL and since Qt 5.3 SPDY support.
Qt Multimedia	Classes for audio, video, radio and camera functionality.
Qt Multimedia Widgets	The widgets from **Qt Multimedia**.
Qt SQL	Contains classes for database integration using SQL.
Qt WebEngine	A new set of Qt Widget and QML webview APIs based on Chromium.
Qt Test	Classes for unit testing Qt applications and libraries.

Figura 10. Essentials Módulos de QT

Para este proyecto, se han empleado los módulos de Qt Core, Qt GUI, Qt Widgets y Qt Multimedia principalmente.

The following table lists the Qt add-ons:

Module	Development Platforms	Target Platforms	Description
Active Qt	Windows	Windows	Classes for applications which use ActiveX and COM
Qt 3D	All	All	Functionality for near-realtime simulation systems with support for 2D and 3D rendering.
Qt Android Extras	All	Android	Provides platform-specific APIs for Android.
Qt Bluetooth	All	Android, iOS, Linux, macOS, and UWP (*)	Provides access to Bluetooth hardware.
Qt Concurrent	All	All (*)	Classes for writing multi-threaded programs without using low-level threading primitives.
Qt D-Bus	All	All (*)	Classes for inter-process communication over the D-Bus protocol.
Qt Gamepad	All	Android, iOS, macOS, tvOS (including the tvOS remote), Linux, Windows, and QNX	Enables Qt applications to support the use of gamepad hardware.
Qt Graphical Effects	All	All	Graphical effects for use with Qt Quick 2.

Figura 11. Add-ons Módulos de Qt.

La lista de módulos en la sección de add-ons es mucho mayor, pero por brevedad se han mostrado solo algunos de ellos.

Al crear una aplicación haciendo uso de la librería de Qt, tenemos que entender como funcionan sus principales mecanismos, los cualesen esencia son tres: QWidgets, Layout System y Signal & Slots System.

2.2.2 QT Widget Architecture

El framework de Qt, en concreto los QWidget, está montado haciendo uso del paradigma de la programación orientada a objeto.

2.2.2.1 Programación Orientada a Objeto

Al hablar sobre la programación orientada a objetos, no se debe confundirla con un nuevo lenguaje de programación, ya que no es cierto. Tampoco se debe confundir el empleo de un lenguaje de programación orientado a objeto, como podría ser Python, C#, etc, con usar realmente la programación orientada a objeto. Se puede usar los lenguajes orientados a objetos sin hacer uso del paradigma de la programación orientada a objeto.

Se tiene que entender que la programación orientada a objeto es un nuevo paradigma de programación, es decir, una nueva metodología de creación de código. La mayoría de personas saben programar haciendo uso del paradigma de programación estructurada, donde se suele solucionar un problema descomponiéndolo en un conjunto de pasos más simples que son finalmente programados.

En cambio, en la programación orientada a objetos no se descompone el problema en pasos. En su lugar se descompone el problema en un conjunto de objetos, de ahí su nombre. La interacción de estos objetos son los que se encargan de solucionar el problema.

A principios del desarrollo de software, muchos programadores acababan escribiendo código que, en esencia, realizaban las mismas tareas. Una forma de evitar este esfuerzo poco fructuoso fue la creación de módulos que permitiesen a los programadores usar código ya existente, evitando así reinventar la rueda. Una de las primeras soluciones adoptadas fue la creación de funciones. Aunque fueran de ayuda, tenían el problema de que se centraban en dar pequeñas funcionalidades, dejando los datos en un segundo plano.

Haciendo uso del paradigma de la programación estructurada y el uso de funciones se creaba nuevo software. Pero a medida que la complejidad del mismo crecía se disparaban: los tiempos de desarrollo, problemas con su diseño, aumento del coste de producción, retrasos… En definitiva, este paradigma mostraba un problema de escalabilidad, impidiendo que el software escalase de forma correcta.

Debido a estos problemas surgió la necesidad de la creación de un nuevo paradigma que permitiera la escalabilidad del software de una forma más natural. Es aquí donde surge el paradigma de la programación orientada a objetos para apaliar esta necesidad.

En la programación orientada a objetos, se suele descomponer el problema en clases. Estas tienen un conjunto de datos y funcionalidades asociadas a la propia clase. Clase y objeto son conceptos distintos pero que guardan relación. Se debe ver la clase como la plantilla o el plano, mientras que el objeto es la creación o instanciación de esa plantilla. Es decir, se usa a la clase para crear al objeto. A este proceso se le denomina “Instanciación”. Cada objeto será una entidad con sus propios datos dentro del software. Se podrán crear tantos objetos de una clase específica como se desee. [10] [11]

Los fundamentos clave de la programación orientada a objeto son: Encapsulación, Herencia y Polimorfismo.

- **Encapsulación**

Toda clase tiene un conjunto de variables internas, llamados atributos y un conjunto de funciones internas, llamados métodos.

En general, en el diseño de una clase, no se quiere que el usuario final pueda acceder a todos los atributos y métodos de la misma. Más bien, interesa proporcionarle un conjunto limitado de acciones suficientes para el correcto desempeño del trabajo.

Para ello, la mayoría de los lenguajes que soportan la programación orientada a objetos, proporcionan un conjunto de palabras reservadas para llevar a cabo esta limitación. Estas palabras suelen ser: *public*, *protected* y *private*.

Todo aquello que sea marcado como *public* será visible para todos. Todo aquello que sea marcado como *private*,

solo será visible para la misma clase. Todo aquello que sea marcado como *protected*, será visible para los hijos de la clase, pero invisible para el usuario final.

Gracias al uso de estas palabras reservadas, se consigue implementar el concepto de encapsulación, ya que el usuario final sólo podrá interaccionar con la clase de la forma en la cual se haya ideado en su diseño [10] [11]

- **Herencia**

La herencia es el mecanismo por el cual se puede reutilizar el código ya existente para crear nuevo código. Supongamos que se quiere crear una clase que tenga unas funcionalidades específicas y resulta que ya existe una clase que tiene la mayoría de las funcionalidades deseadas. En primera instancia, se podría pensar que no se puede usar y se tendría que implementar desde cero. Se puede crear la nueva clase usando como base a la clase ya existente. Esto se denomina Herencia. Gracias a este mecanismo, solo se tendrá que programar las nuevas funcionalidades. El resto de las funcionalidades las heredara de su clase base. [10] [11]

- **Polimorfismo**

"En programación orientada a objetos se denomina polimorfismo a la capacidad que tienen los objetos de una clase de responder al mismo mensaje o evento en función de los parámetros utilizados durante su invocación. Un objeto polimórfico es una entidad que puede contener valores de diferentes tipos durante la ejecución del programa." [9]

Básicamente, el polimorfismo busca que un objeto se comporte como si fuese un objeto de una clase derivada, aunque este se haya guardado en su forma base. [10] [11]

Ahora que se ha explicado brevemente en que consiste la programación orientada a objetos, se puede explicar el funcionamiento básico de la arquitectura de los QWidgets.

Los QWidgets son la base para poder crear aplicaciones, ya que son los cimientos de todo el sistema. Básicamente, QWidget es la clase base de la cual heredan el resto de componentes.

La Figura 12 ayuda a hacerse una idea de como esta montada la arquitectura de este sistema.

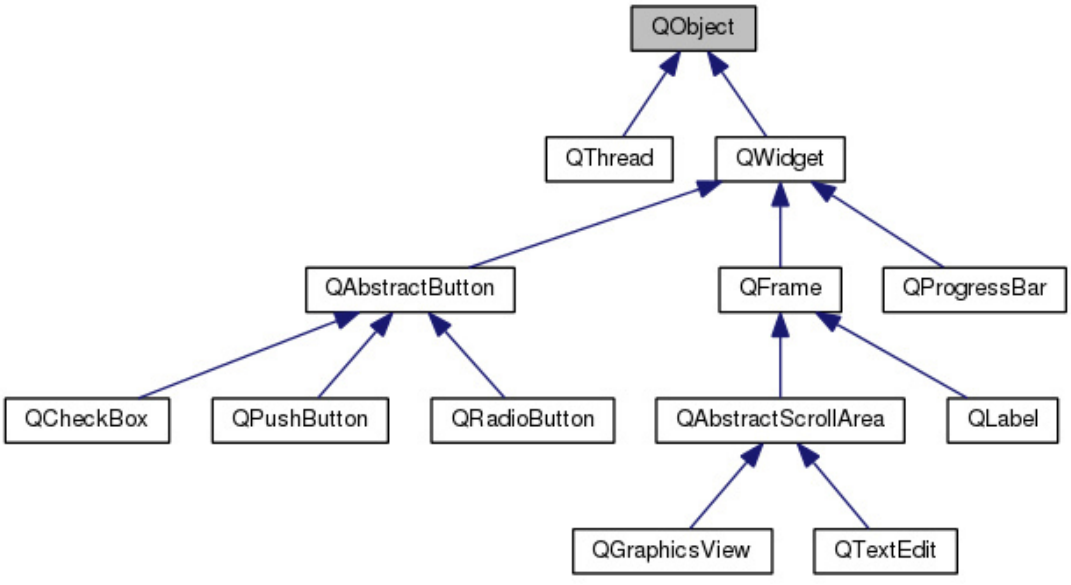


Figura 12. Arquitectura de Clases de los QWidgets.

Se ve como de la clase QObject hereda la clase QWidget y a su vez, de QWidget heredan otras clases. Estas clases que heredan de QWidget son especializaciones más refinadas de QWidgets. Se puede apreciar que todos los componentes presente en la interfaz son básicamente objetos independientes, y estos objetos heredan a su vez de una clase superior.

Usando este mecanismo no haría falta implementar todo el código desde cero, sino que cada vez que se desease crear un nuevo widget, simplemente se tendría que escoger un widget que presente las características buscada y crear una clase derivada de él.

Como se ha podido imaginar, la arquitectura de los QWidgets hace uso del paradigma de la programación orientada a objetos. Además, empleando el mecanismo de la herencia facilita la escalabilidad del software.

2.2.3 QT SIGNAL & SLOT System

Se ha visto el mecanismo que emplea Qt para crear los diferentes componentes, pero no sabemos el mecanismo que hay detrás para que todo funcione correctamente. Es decir, sabemos que una ventana gráfica está formada de más widget los cuales saben autogestionarse e interactuar los unos con los otros, creando lo que se llama programación reactiva.

La programación reactiva, también conocida como programación por eventos, es una forma de programar en la cual el flujo de la información debe reaccionar de forma correcta ante un evento que se produzca en el sistema de forma asíncrona. Por ejemplo, cuando pulsamos un botón en la interfaz, esta reacciona ejecutando alguna acción.

En la Figura 13se puede ver un ejemplo sencillo de una interfaz de usuario.

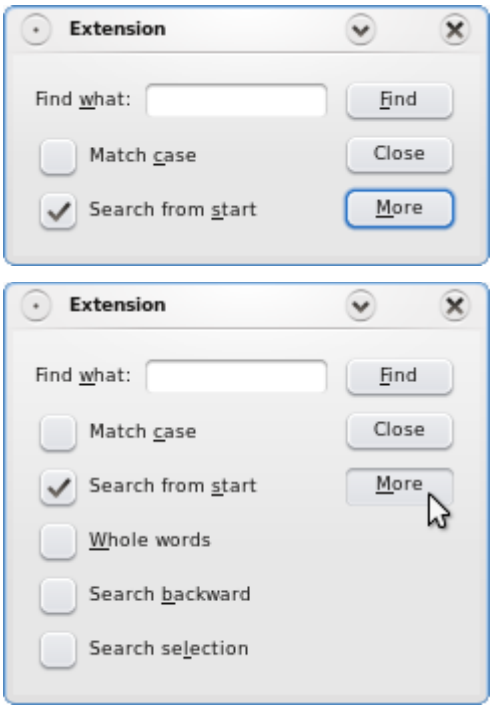


Figura 13. Ejemplo de ventana Dinámica.

En la Figura 13 se observa que al pulsar el botón "More", la ventana reacciona mostrando más acciones disponibles. Ahora nos interesaría saber el mecanismo que emplea Qt para llevar a cabo este proceso de comunicación o interacción entre objetos (Qwidgets). El mecanismo detrás de esta interacción se denomina "SIGNAL & SLOTS".

Este mecanismo, es una alternativa a las "*callbacks*" que realizan otros programas. Una **"SIGNAL"** se emitida cada vez que un evento específico tiene lugar. Un "**SLOT"** se ejecuta como la respuesta de esa señal. Un "**SLOT**" una función miembro de nuestra clase.

Gracias a este mecanismo, se puede conectar cualquier objeto con otro cualquiera, haciendo que cuando en uno de ellos se produzca un evento, el otro objeto conectado reaccione ejecutando alguna acción que se le haya programado previamente.

El sistema permite conectar a una misma "**SIGNAL"**, uno o varios "**SLOTS"**, en cuyo caso los "**SLOTS"** se irán ejecutando en el mismo orden el que fueron conectados. También, se puede conectar una "**SIGNAL"** a otra "**SIGNAL".**

Las ventajas de este mecanismo es que resulta ser "*type safe*". Además de presentar un acoplamiento libre. Esto significa que la clase que emite la señal no sabe, ni le importa, que clase/s reciben la señal que ha emitido. Para

poder emplear este mecanismo, es necesario que las clases que se creen hereden de la clase QObject o de alguna otra clase que a su vez herede de QObject.

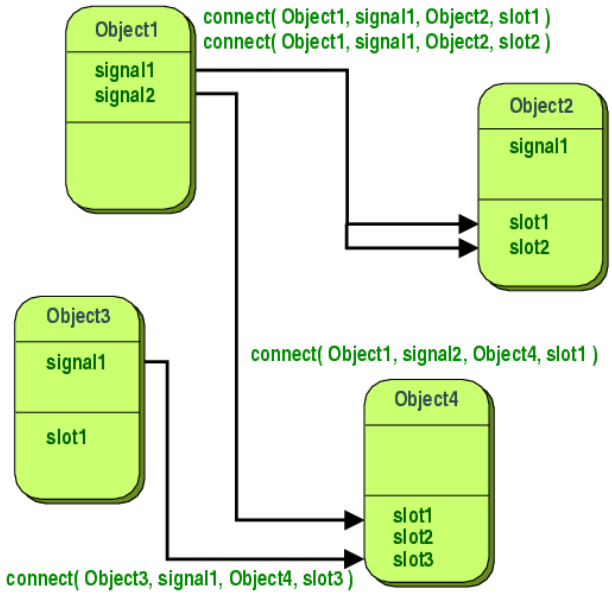


Figura 14. Ejemplificación del funcionamiento del sistema SIGNAL & SLOT.

```
#include <QObject>

class Counter : public QObject
{
    Q_OBJECT

public:
    Counter() { m_value = 0; }

    int value() const { return m_value; }

public slots:
    void setValue(int value);

signals:
    void valueChanged(int newValue);

private:
    int m_value;
};
```

```
void Counter::setValue(int value)
{
    if (value != m_value) {
        m_value = value;
        emit valueChanged(value);
    }
}
```

```
Counter a, b;
QObject::connect(&a, &Counter::valueChanged,
                 &b, &Counter::setValue);
```

Figura 15. Ejemplo de uso del sistema de SIGNAL Y SLOT

En la Figura 15 se un ejemplo sencillo de código empleando el mecanismo de Signal and Slots.

2.2.4 Ejemplo de Inicializacion de un Nuevo Proyecto

Cuando se está interesado en crear un proyecto usando QT, se usa el propio entorno de desarrollo que nos proporciona la librería para tal fin, Qt Creator. En este entorno de desarrollo se pueden ir añadiendo archivos, editarlos, compilar, ejecutar el proyecto completo, hacer debugging… Se puede decir que es un entorno bastante parecido a Visual Studio, aunque no tan completo como este último. Una de las ventajas que presenta este IDE es que no posee un compilador propio, como sería el caso de Visual Studio, sino que se le debe configurar el compilador a usar: GCC, Microsoft Visual Studio Compiler etc.

Cuando se inicia Qt Creator se verá lo siguiente:

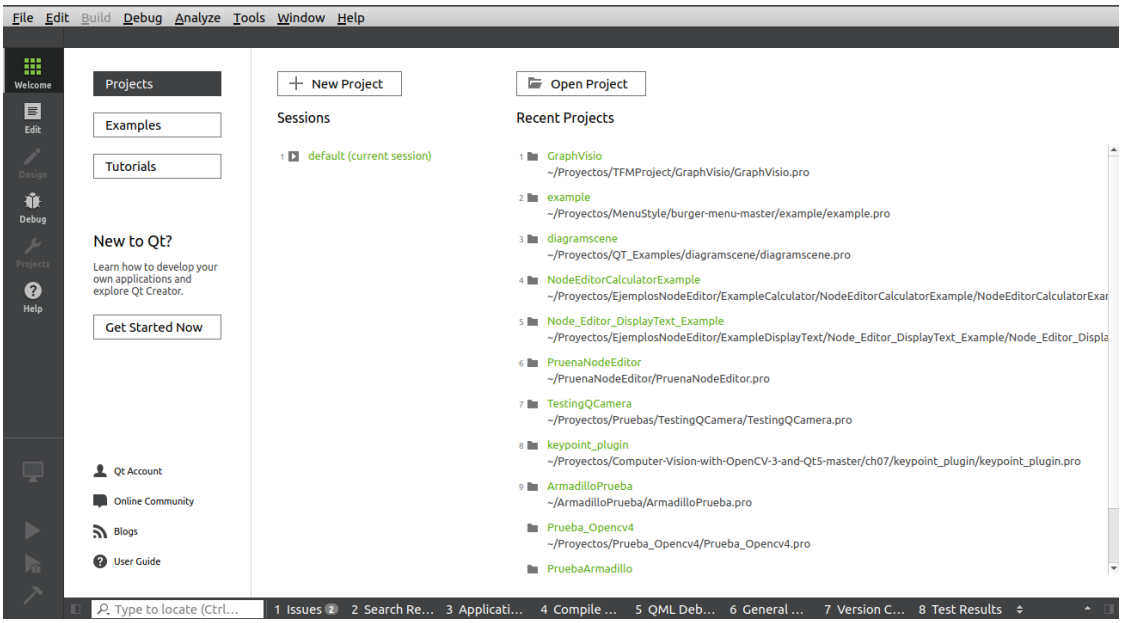


Figura 16. Ventana Principal de Qt Creator.

En esta ventana se puede abrir proyectos recientes, abrir un nuevo proyecto, abrir proyectos de ejemplos o ver tutoriales para aprender a manejar la librería.

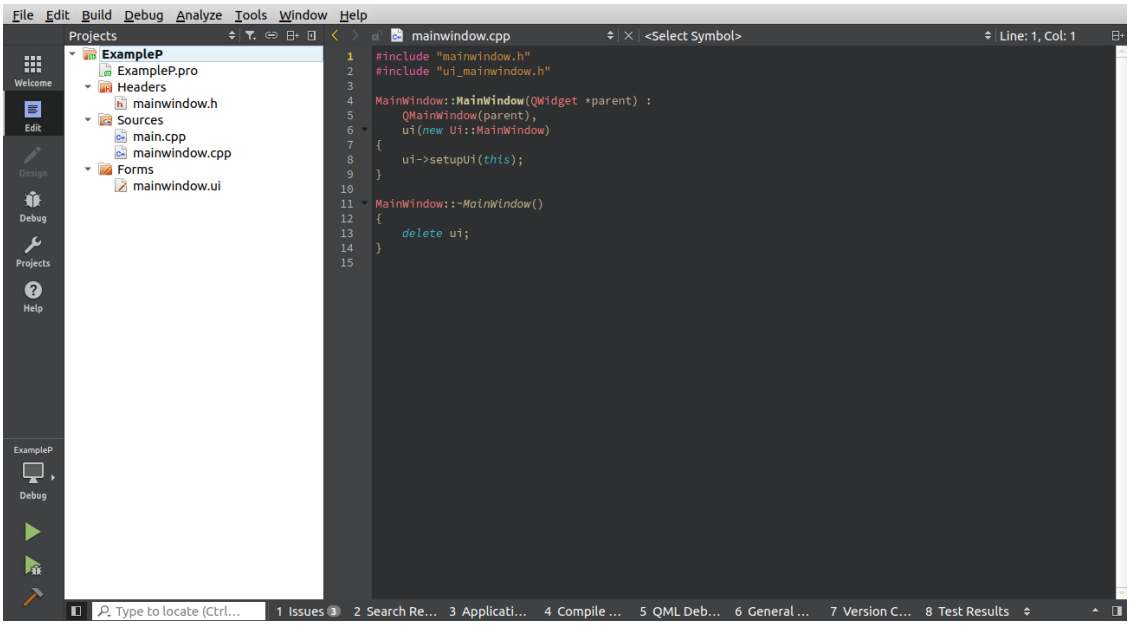


Figura 17. Workspace de un proyecto inicial en Qt

A la derecha, se tiene la ventana de edición o creación de código, Figura 17. A la izquierda, se tiene el directorio de archivos que componen el proyecto. La carpeta Raíz es el nombre del propio proyecto. El primer archivo que se ve, y se verá en cualquier proyecto en Qt, es un archivo de extensión ".pro".

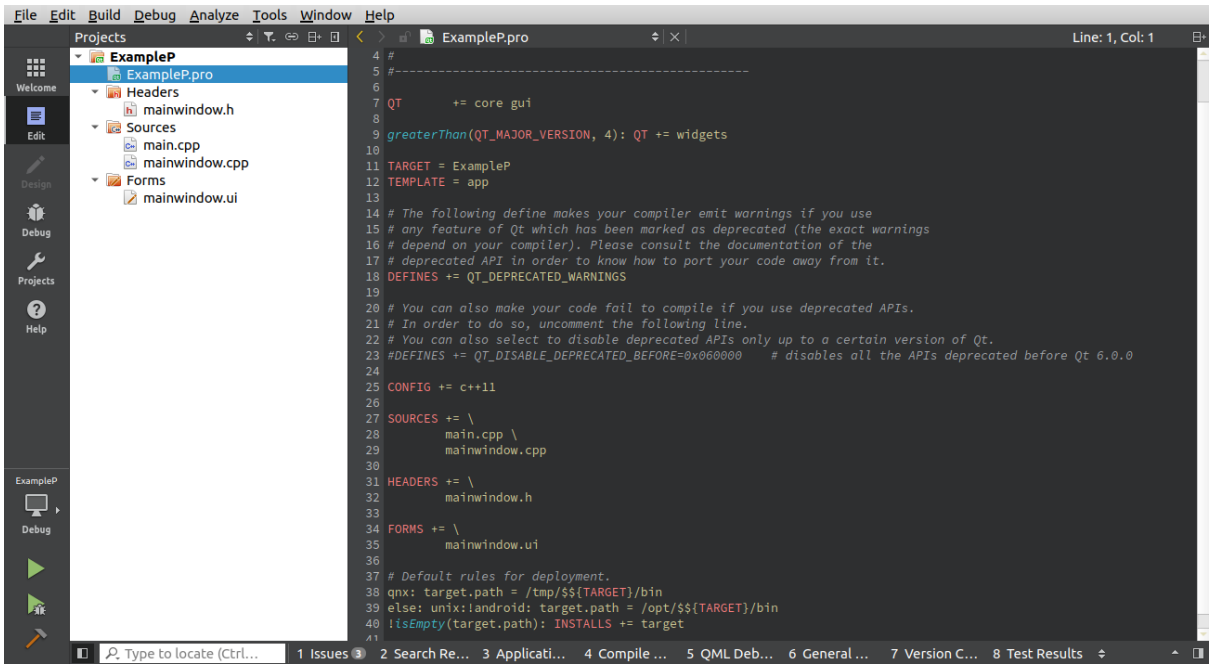


Figura 18. Configuración básica del archivo .pro.

En este archivo se añadirán los modules que se necesitan emplear, la versión de C++ que se desea emplear, los distintos archivos que componen el proyecto (.h, .cpp, .ui), además de la configuración de librerías externas que se necesiten emplear en el proyecto. En resumidas cuentas, este es el archivo de configuración del proyecto.

Dentro de la carpeta raíz se encuentran varias carpetas. La primera es “Headers”, aquí irán a parar todos los archivos “.h” que se vayan creando. La siguiente carpeta será “Sources”, estará formada por el archivo main que toda aplicación en C++ posee, además de todos los archivos “.cpp” se vayan creando en el proyecto.

Como última carpeta se tendrá a “Forms”. Se ve que los archivos que se guardan en esta carpeta tienen la extensión “.ui”. Estos son los archivos que emplea Qt para crear el formulario con los diferentes componentes gráficos (QWIdgets). En realidad, estos archivos son un documento “XML” encargados de guardar la información gráfica de la ventana que se está diseñando. Cuando se compila el proyecto, Qt se encarga de convertir este archivo a uno equivalente en C++ que contendrá todo el código necesario.

Al hacer doble clic en los archivos de extensión “ui” se entra en el editor de formularios que nos proporciona Qt Creator.

El editor está diseñado para que sea muy fácil e intuitivo su manejo. Se tiene un área central donde se encuentra nuestro formulario. En esta área se le pueden ir añadiendo nuevos componentes. Todos los componentes que tenemos a nuestra disposición se encuentran en la zona de la izquierda. Simplemente se tendrá que arrastrar y soltar cualquiera de estos componentes para añadirlo a nuestro formulario. En la zona de la derecha tenemos dos bloques. El bloque de arriba muestra la jerarquía de los diferentes componentes que se van añadiendo al formulario. Al seleccionar cualquiera de ellos, la ventana de abajo a la derecha se nos desplegará con diferentes propiedades que se le pueden configurar a dicho componente.

Cabe mencionar que cualquiera de las propiedades puede ser modificada por código en cualquier momento. Además, si se desea, se podrá crear todo el formulario a mano programándolo directamente con código. Qt ofrece esta herramienta para facilitar el desarrollo.

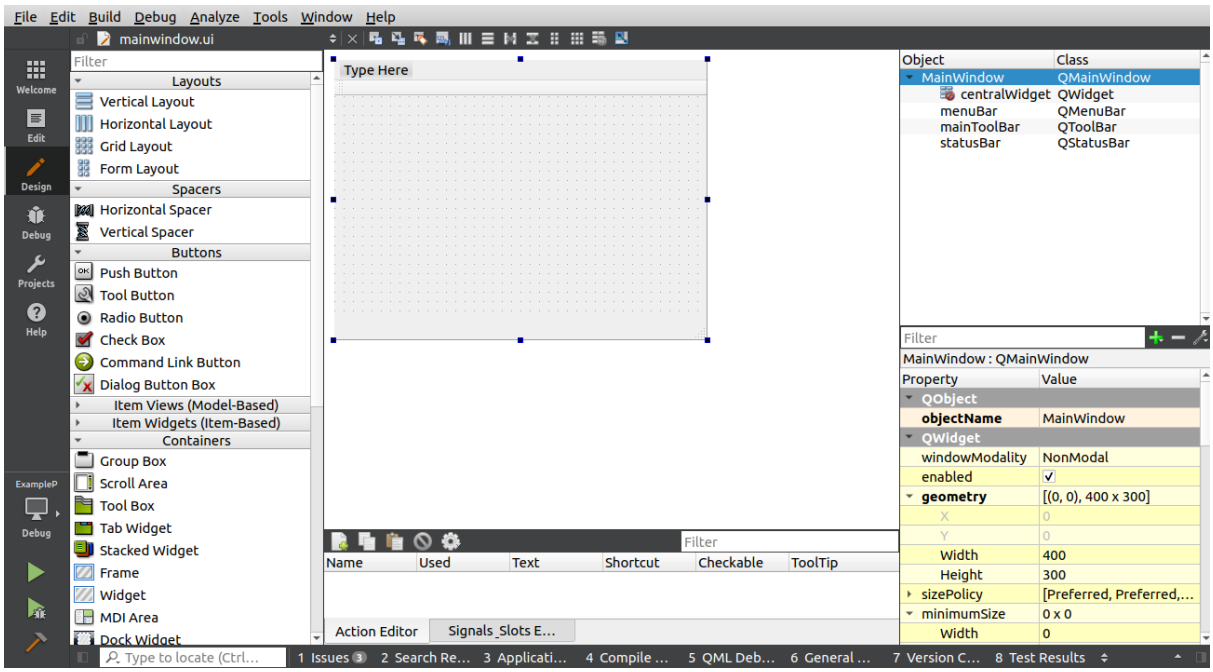


Figura 19. Interfaz gráfica para la creación de Widgets.

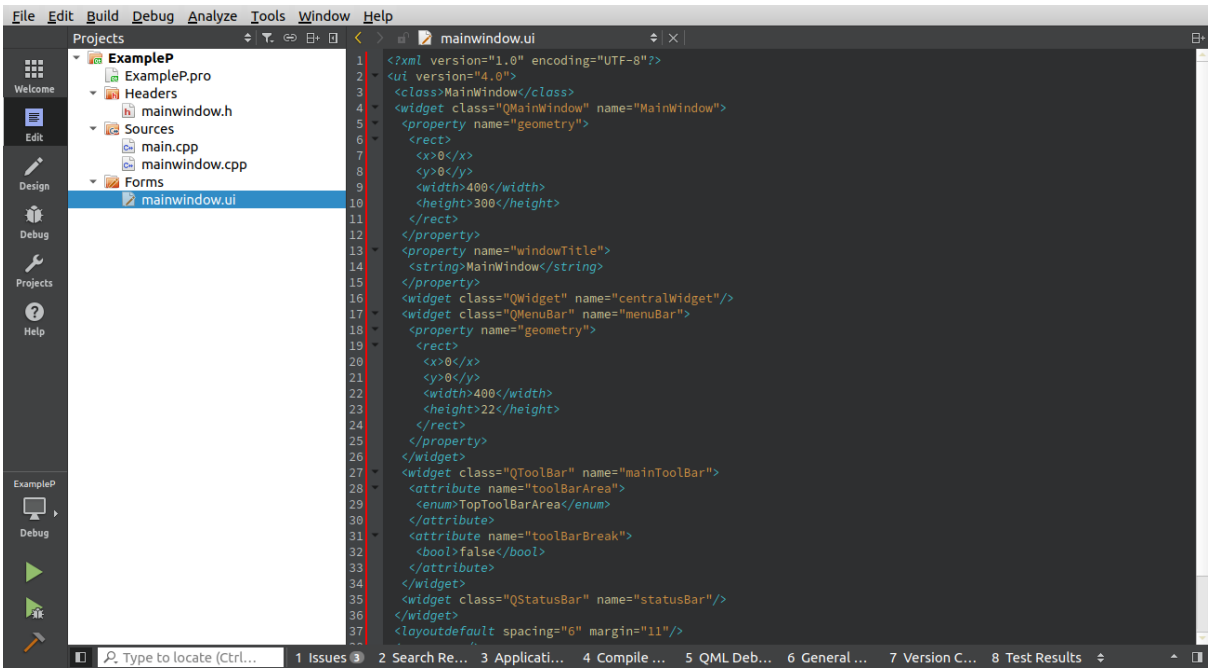


Figura 20. Interior del archivo .ui.

Debido a que el siguiente proyecto ha sido realizado bajo el uso del lenguaje de programación C++, además que para su desarrollo se emplean diferentes algoritmos de visión por ordenador, se ha considerado que la mejor librería para llevar a cabo este fin es la de OpenCV.

2.3 OpenCV

OpenCV (Open Source Computer Vision Library) es una librería de Código libre diseñada para el campo de la visión por computador y "*Machine Learning*". Se creo con el objetivo de proporcionar una infraestructura común para el Desarrollo de aplicaciones de visión por ordenador, acelerando de esta forma el uso de "*Machine Perception*" en productos comerciales. Gracias a que posee una licencia BSD, facilita la utilización para usos comerciales y la modificación del código.

La librería posee más de 2500 algoritmos optimizados, entre los cuales se incluyen un conjunto de algoritmos clásicos, estado del arte sobre la visón por ordenador al igual que posee también algoritmos para el campo de "*machine learning*". Tiene una comunidad de más de 47000 usuarios y se estima que excede los 18 millones de descargas. Es usada extensamente, tanto por compañías, grupos de investigación y organizaciones gubernamentales.

Está escrita de forma nativa en C++, aunque posee interfaces que dan soporte para otros lenguajes como: Python, Java y Matlab. Ademas de poder operar en varios sistemas operativos: Windows, Linux, Android y macOS. La librería está tendiendo hacia aplicaciones de visión por ordenador en tiempo real, gracias a un conjunto de infraestructuras, las cuales están siendo activamente desarrolladas en la actualidad. Estas dan soporte para integrarse con CUDA y OpenCL. [12]

OpenCV se inició en uno de los laboratorios de investigación de Intel. Uno de sus autores, Gary Bradski, estaba visitando universidades y noto que algunos de los grandes grupos de investigación tales como, MIT Media Lab habían desarrollado una infraestructura que era pasada de unos estudiantes a otros. De esta forma, no se tenía que rehacer todo, sino que se creaba nuevo código sobre el ya existente. Debido a esto, OpenCV fue concebido como una forma de hacer que la visión por ordenador tuviera una infraestructura disponible universalmente. [13]

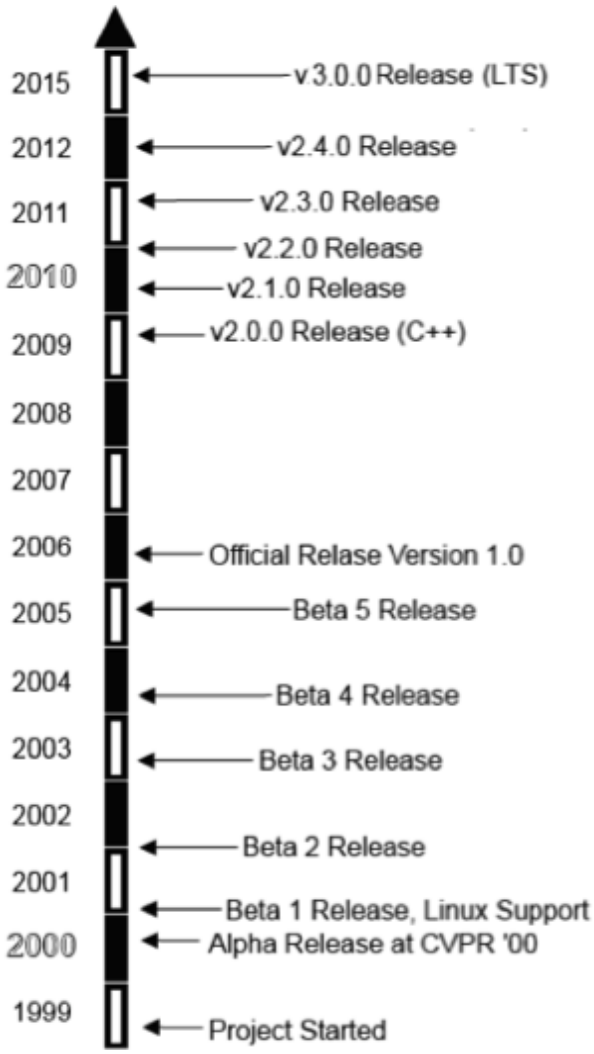


Figura 21. Evolución Cronológica de OpenCV

Los algoritmos de la librería de OpenCV se pueden usar para detectar y reconocer caras, identificar objetos, clasificar acciones humanas en vídeo, seguimientos de cámara, seguimientos de objetos, extraer información 3D de Objetos, producir nubes de puntos en 3D a partir de imágenes de cámaras en estereo, encontrar imágenes similares de una base de datos de imágenes, remover ojos rojos de imágenes tomadas con flash, seguir el movimiento de los ojos, reconocer el escenario y establecer marcadores para superponerlos con realidad

aumentada, etc. [13] [12] [14]

2.4 Control de Versiones. Git & GitHub

En proyectos donde la cantidad de archivos empieza a ser considerable o proyectos en los cuales estén involucrados un grupo de desarrolladores, se evidencia la necesidad de una herramienta para mantener un control sobre los diferentes archivos que componen el proyecto.

No es buena práctica, ni tampoco útil, hacer muchas copias del proyecto, ya que al final se acabaría con un desorden descomunal de carpetas.

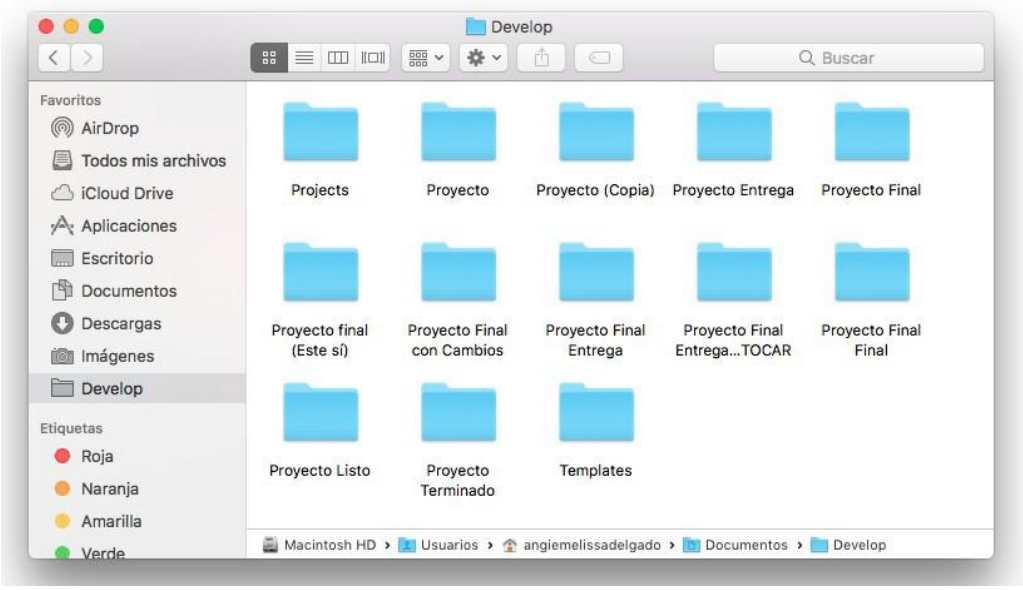


Figura 22. Ejemplificación de los problemas de no usar control de versiones

Por lo tanto, para evitar la mala práctica mostrada en la Figura 22, se hace uso del control de versiones. Suele ser un programa el encargado de realizar esta tarea. Este, va registrando los cambios que se van realizando sobre el proyecto a lo largo del tiempo, con la gran ventaja y utilidad de que en cualquier momento se podrá volver a una versión anterior del mismo proyecto.

Se pueden clasificar a los Sistemas de Control de Versiones en grupos: Centralizados o Distribuidos.

Sistemas Centralizados

En este caso, se tiene una única versión centralizada alojada en un servidor, en donde los diferentes usuarios del proyecto trabajan conjuntamente en ella.

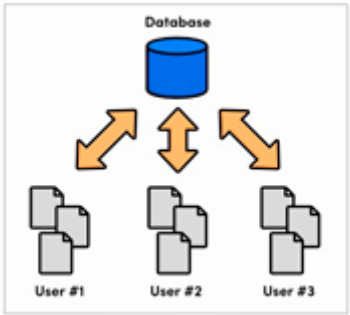


Figura 23. Esquema de un Sistema de control de Versiones Centralizado.

Sin embargo, los conflictos son el principal problema que presenta este esquema. Si dos usuarios tratan de

trabajar en el mismo archivo, el sistema puede bloquear a alguno de ellos, o en el caso que estuviesen trabajando en el mismo archivo, cada vez que uno hiciese una modificación al archivo le generaría un conflicto al otro.

Sistemas Distribuidos

En este tipo de sistemas, no se trabaja con un único repositorio común para todos. En su lugar, cada usuario posee una copia del repositorio raíz. De esta forma, cada uno de los usuarios pueden trabajar de forma segura en los diferentes archivos, ya que todos poseen una copia del mismo y están trabajando de manera local sobre los archivos.

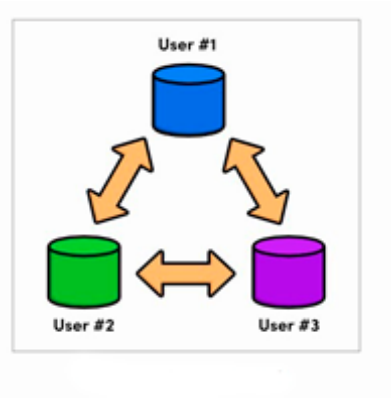


Figura 24. Esquema de un Sistema de Control de Versiones Distribuido.

El sistema de control de Versiones más conocido es Git. Es un software de uso libre que debe descargarse en el ordenador. Es el sistema de control de versiones empleado en el desarrollo del proyecto.

No se debe confundir git con GitHub, ya que git es el control de versiones local, es decir, es el software que gestiona las versiones del proyecto, pero de forma local para nuestro propio ordenador. Si se pierden los datos se pierde todo el proyecto. Como a ningún desarrollador de software le gusta la idea de perder todo el trabajo realizado en su proyecto surge GitHub. Este, es un sistema de control de versiones remoto que nos permite subir nuestro proyecto a la nube, quedando así almacenado de forma segura. De esta forma, aunque le pase algo a nuestro ordenador, siempre podremos volver a descargar todo el proyecto del repositorio remoto y continuar el trabajo por donde lo dejamos.

2.4.1 Principales Comandos

- git add → Añadir archivos.
- git branch Name → Crea una nueva rama
- git -b checkout Rama → Te cambias de Rama y la creas al mismo tiempo.
- git commit → Añade un nuevo “Commit” al Proyecto. Es parecido a un punto de retorno.
- git status → Nos informa sobre el estado del directorio de trabajo, indicandonos los archivos que se encuentran con cambios no registrados.
- git merge NombreRama2Absorber → Integra en la rama actual la rama indicada.
- git push RemoteRepo LocalRep → Actualiza el repositorio remoto con el repositorio local indicado.

Una de las ventajas de emplear un IDE como Qt Creator, es la opción que ofrece de poder integrar el control de versiones git dentro de nuestro proyecto. De esta forma, cuando se quiera actualizar el repositorio con los cambios realizados, se puede emplear la propia interfaz de usuario de Qt Creator para realizar un *Commit*. Al realizar el *Commit*, el sistema indicará los archivos que han sufrido cambios con respecto al *Commit* anterior, pudiendose seleccionar aquellos archivos que se quieran meter en el *Commit*. Esto resulta de gran utilidad, ya que los comandos de git *add* y git *commit,* se pueden realizar a través de la propia interfaz de Qt Creator.

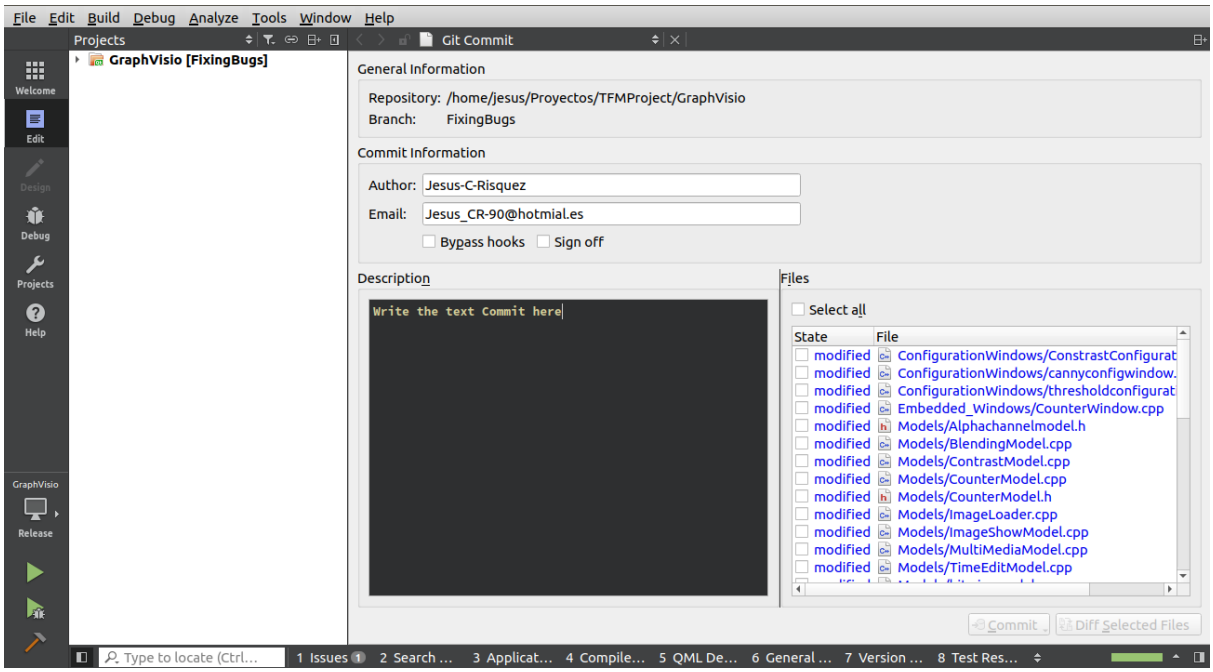


Figura 25. Sistema de control de versiones embebido en Qt Creator

2.5 Node Editor Library

Cuando se quiere crear un proyecto que va más allá de un simple scritp, suele ser una buena práctica buscar si existe alguna librería que proporcione un punto de partida para desarrollarlo. Ya que empezar a desarrollar todo desde cero podría llegar a ser una locura.

Para este proyecto, se ha buscado por internet si existía alguna librería que proporcionase ese punto de partida. Al final se ha dado con un proyecto en GitHub que posee los cimientos para el desarrollo de nuestra idea. Este proyecto nos proporciona las herramientas para poder crear bloques en los cuales se tendrá que definir tanto el conjunto de entradas y salidas, el tipo de entrada de las mismas, la operación que lleva a cabo el bloque etc.

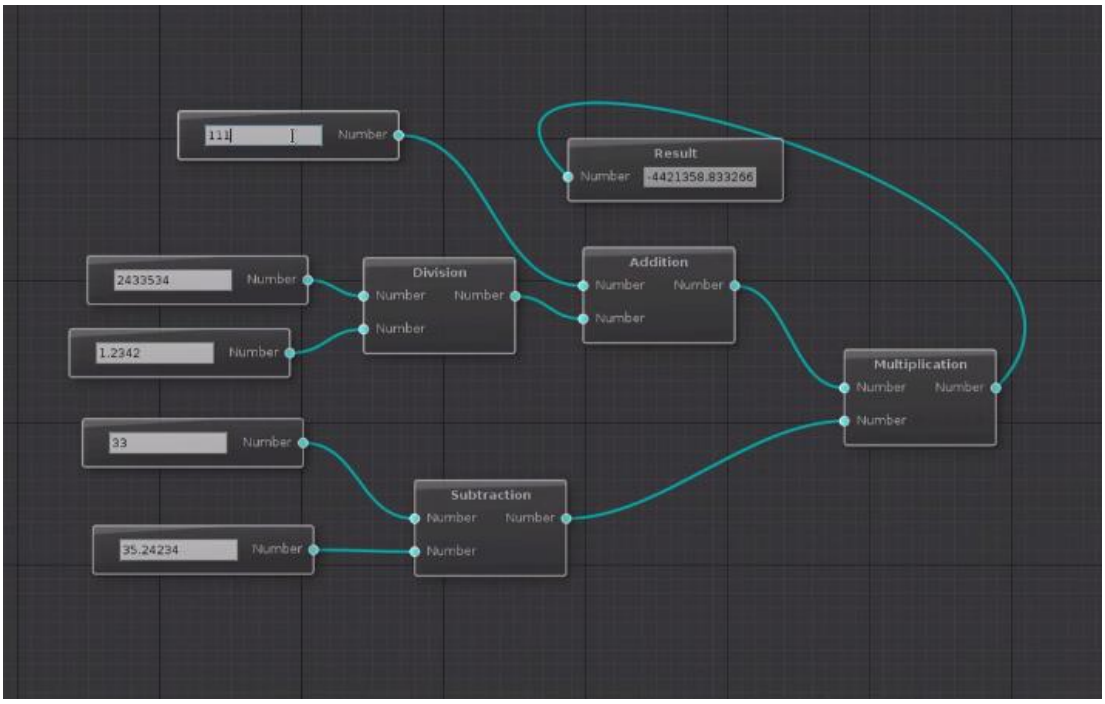


Figura 26. Ejemplo proporcionado por la propia librería.

Se ha elegido este proyecto ya que esta realizado completamente en C++ y además su núcleo está montado usando la librería de Qt. Por lo tanto, podríamos decir que calzaba perfectamente con nuestra idea.

Aprender a manejarlo no ha sido tarea fácil, ya que al no ser un proyecto oficial como podría ser OpenCV, sino que es un proyecto creado y gestionado por un pequeño grupo de desarrolladores, no posee documentación alguna para saber cómo funciona o como se debe emplear. Simplemente, se contaba con unos cuatro o cinco ejemplos que los desarrolladores proporcionaban y el código fuente. Así que con esa poca información nos hemos tenido que apañar.

3 IMPLEMENTACION

En secciones anteriores se comentaba que el presente proyecto buscaba ser una prueba piloto para demostrar la viabilidad de esta nueva metodología de programación, y para ello se tenía que escoger un campo de aplicación. Finalmente, se escogió el campo de la visión por ordenador para el proyecto que nos ocupa. Sin embargo, tanto la librería de OpenCV, como el propio campo de la visión por ordenador dispone de un abanico enorme de posibilidades. Por lo tanto, se debe ser más específico en aquello que se va a implementar, sino se acabaría implementando bloques que encapsulan operaciones elementales de OpenCV sin llegar a un fin específico. Esto podría tener como consecuencia alargar enormemente el desarrollo del presente proyecto, además de que no se podría ver en realidad la viabilidad y potencial a futuro del mismo.

3.1 Ejemplo a Desarrollar

3.1.1 Ejemplo

Como el objetivo a futuro que sigue el proyecto es poder demostrar la viabilidad y potencial de esta nueva forma de desarrollar algoritmos en cualquier campo, no solo en el campo de la visión por ordenador, se ha decidido montar un ejemplo de aplicación práctico.

Se ha optado por escoger un ejemplo clásico de la visión por ordenador. El problema a solucionar escogido es la localización en tiempo real de un conjunto de monedas que presentan gran presencia de ruido y sombras como se ve en la Figura 27.

Figura 27. Imagen con el Problema a Desarrollar.

El video ha sido grabado manualmente empleando la cámara de un móvil, monedas y granos de café para simular el ruido. El sistema debe ser capaz de localizar las monedas en un entorno lejos del ideal, presencia de sombras y mucho ruido, como se ve en la Figura 27

3.1.2 Diagrama de Flujo

La Figura 28 muestra el diagrama de flujo ha implementar para obtener el resultado buscado.

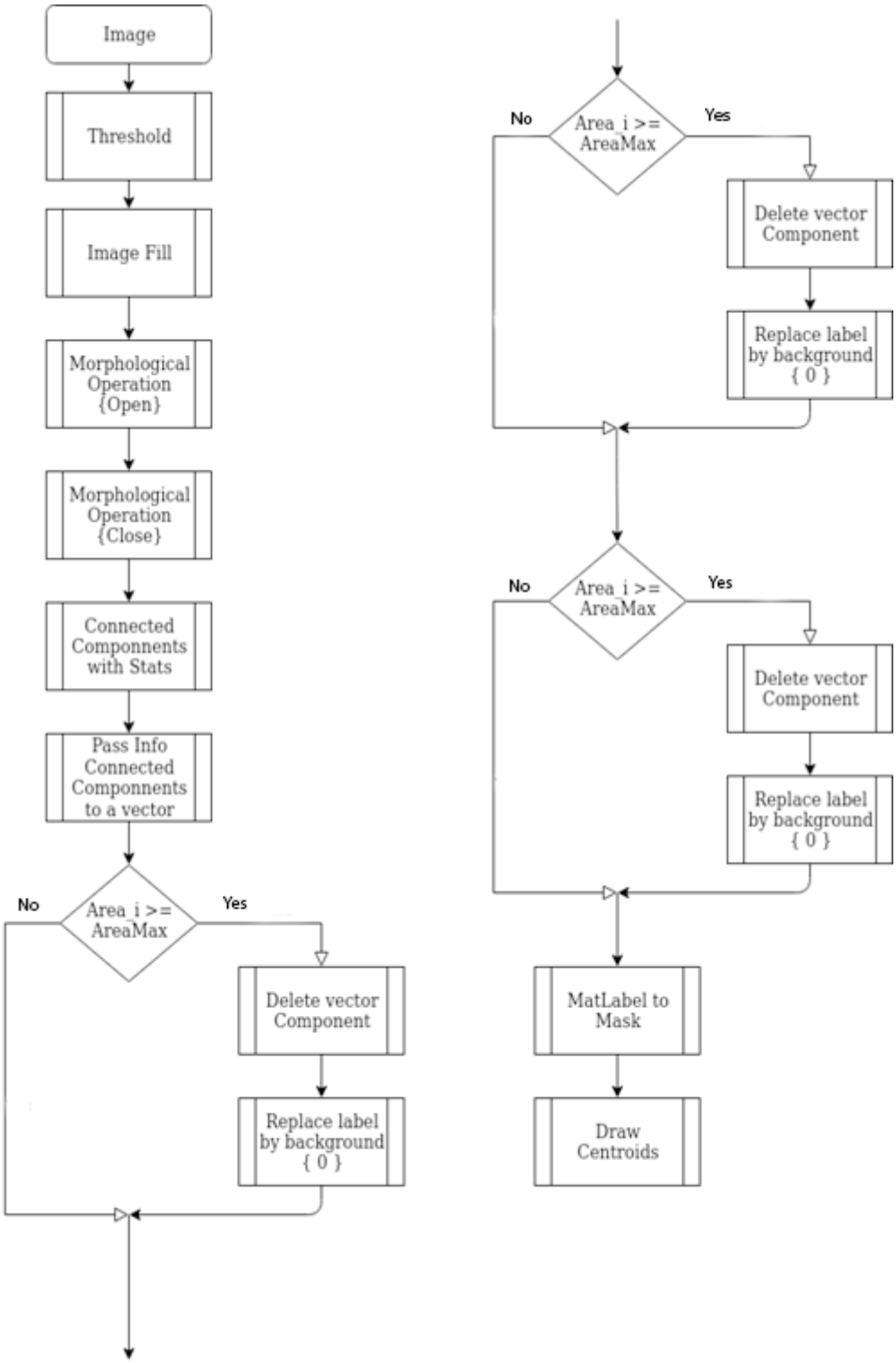


Figura 28. Diagrama de Flujo para la solución propuesta

3.1.3 Bloques Necesarios

En el diagrama de Flujo de la Figura 28 se puede observar que los bloques que se deben implementar son:

- Read Source (Video or Image)
- Threshold Operation
- Image Fill Operation

- Morphological Operation
- Connected Components
- Filter by Area Connected Components
- Convert MatLabel to Mask
- Draw Points in the Centroids Position

3.2 Arquitectura del Sistema

Se empezará por listar las diferentes clases creadas para la realización del proyecto. Así se tendrá una idea global del conjunto de clases que lo conforman. Se procederá agrupando las clases en diferentes grupos.

Grupo Operations

Figura 29. Clases creadas pertenecientes al grupo de Operaciones

Este grupo se compone de las diferentes operaciones que se han ido implementando. Se deben comentar ciertos aspectos de este grupo. Primero, se ha optado por separar la información asociada a la propia operación, de la operación en sí misma. Esto dota al sistema de mayor flexibilidad ya que por una parte tenemos los datos y por otra la lógica.

Segundo, se parte de una clase base y de ella heredan especializaciones de la misma. Se podría decir que la clase base actúa como interfaz común para las especializaciones. En este caso, las clases Bases o Interfaces son: *Info_BaseAlgorithm* y *BaseAlgorithm*.

Grupo Nodes

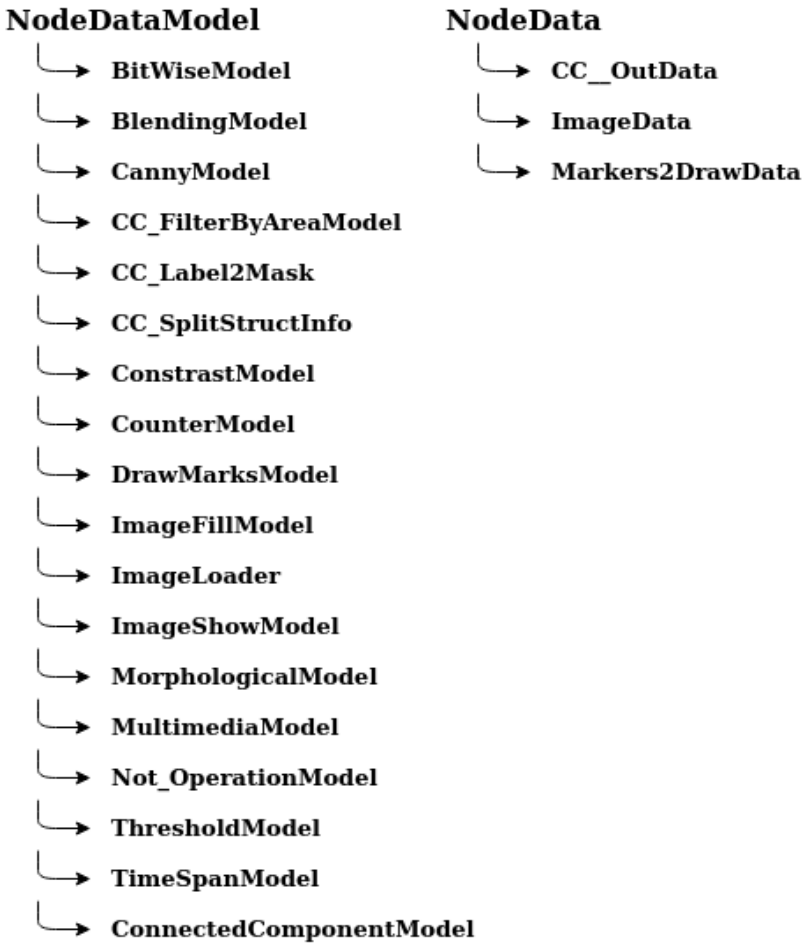


Figura 30. Clases creadas para el grupo Nodos

Todas estas clases heredan de las clases base *NodeDataModel* y *NodeData*. Ambas, seran explicadas en mayor profundidad en las siguientes secciones.

Grupo Multimedia

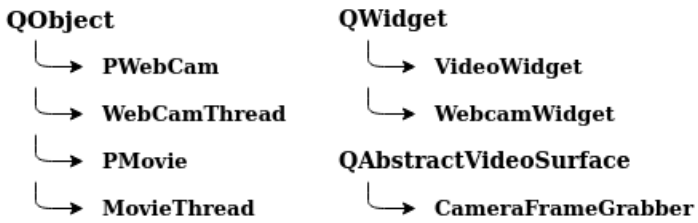


Figura 31. Clases creadas para el Grupo Multimedia

Este conjunto de clases son las encargadas de gestionar la/s WebCam y el Video/s que se cargan en la apliacion.

Grupo de ConfigDialog & Utilities Widget

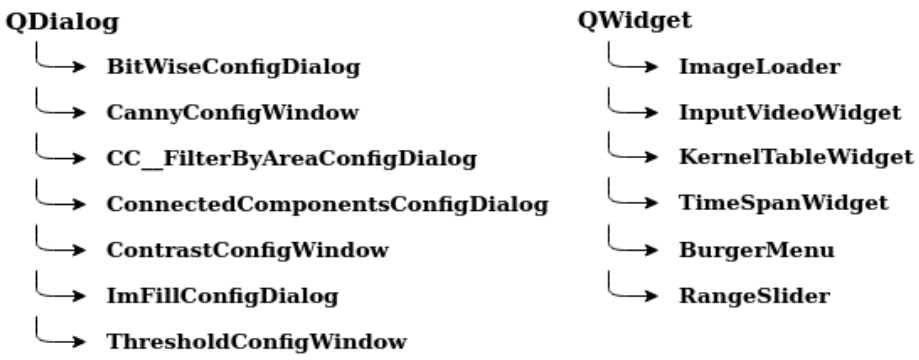


Figura 32. Conjunto de clases auxiliares creadas para el sistema.

Finalmente, el siguiente conjunto de clases son las encargadas de crear las diferentes ventanas de configuración de las operaciones y widgets de utilidad para las mismas.

3.2.1 Arquitectura en Conjunto

El comportamiento del sistema se basa en la existencia de un conjunto de Nodos, que pueden tener a su vez un conjunto de entradas y salidas asociadas. Estos Nodos se encargan de realizar operaciones simples, empleando para ello la información recibida por los puertos de entrada. En última instancia, transmiten el resultado obtenido hacia los puertos de salida. Las Conexiones o Enlaces es la forma que tienen los Nodos de transmitirse la información procesada. Estas Conexiones conectarn unos Nodos con otros.

En principio, se podría pensar que con los Nodos y Conexiones es suficiente. Sin embargo, aún nos hace falta el elemento principal. Este es el encargado de sostener y gestionar toda la arquitectura de Nodos y Conexiones. Dicho elemento es la denominada Escena. Se podría decir que la escena es el contenedor de los diferentes Nodos y Conexiones. La Figura 33 muestra cómo se organiza el sistema.

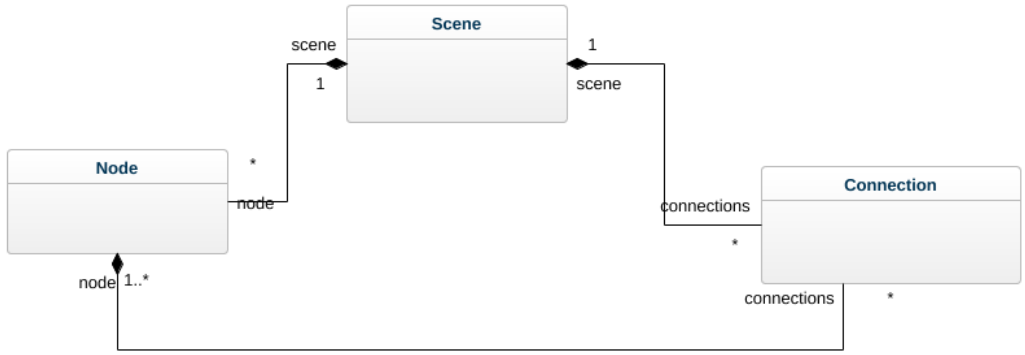


Figura 33. Esquema Simplificado de los Principales Componentes del Sistema.

En la Figura 33 muestra las clases primarias que componen el Sistema: Node, Scene y Connection.

Primero, se atenderá a la relación entre *Node* y *Scene*. La línea indica que hay una relación entre un *Nodo* y una *Escena*. El rombo negro del lado de la *Escena*, indica una relación de composición. Esto quiere decir que los *Nodos* no pueden existir sin la *Escena*. Si se elimina la *Escena*, los *Nodos* son eliminados con ella, puesto que no podrían existir. La multiplicidad (los números en los extremos de las conexiones) es otra característica que se observa en la relación entre *Nodo* y *Escena*. Para el caso de la *Escena* y *Nodo*, indica que una escena puede contener cualquier número de *Nodos*, mientras que un *Nodo* sólo puede pertenecer a una única *Escena*.

La relación entre *Escena* y *Conexión* es igual a la de *Nodo* y *Escena*. Se tiene una relación de composición, no pudiendo existir una conexión sin una *Escena*. Además, una *Escena* puede contener cualquier número de *Conexiones*. Sin embargo, una *Conexión* solo puede existir en una única *Escena*.

Finalmente, se tiene la relación entre *Nodo* y *Conexión*. En esta, se observa una relación de Composición, puesto que no puede existir una *Conexión* si no existe un *Nodo*. Además, un *Nodo* puede contener cualquier número de *Conexiones*, mientras que una *Conexión* puede pertenecer a un *Nodo* o a muchos. Aunque esto último parezca

extraño al principio, si se piensa detenidamente, es el comportamiento más lógico. Puesto que esta relación indica que es posible realizar una misma *Conexión* a más de un *Nodo*.

3.2.2 Arquitectura de los Nodos

La arquitectura de los *Nodos* se puede ver en el siguiente Diagrama de Clases de la Figura 34.

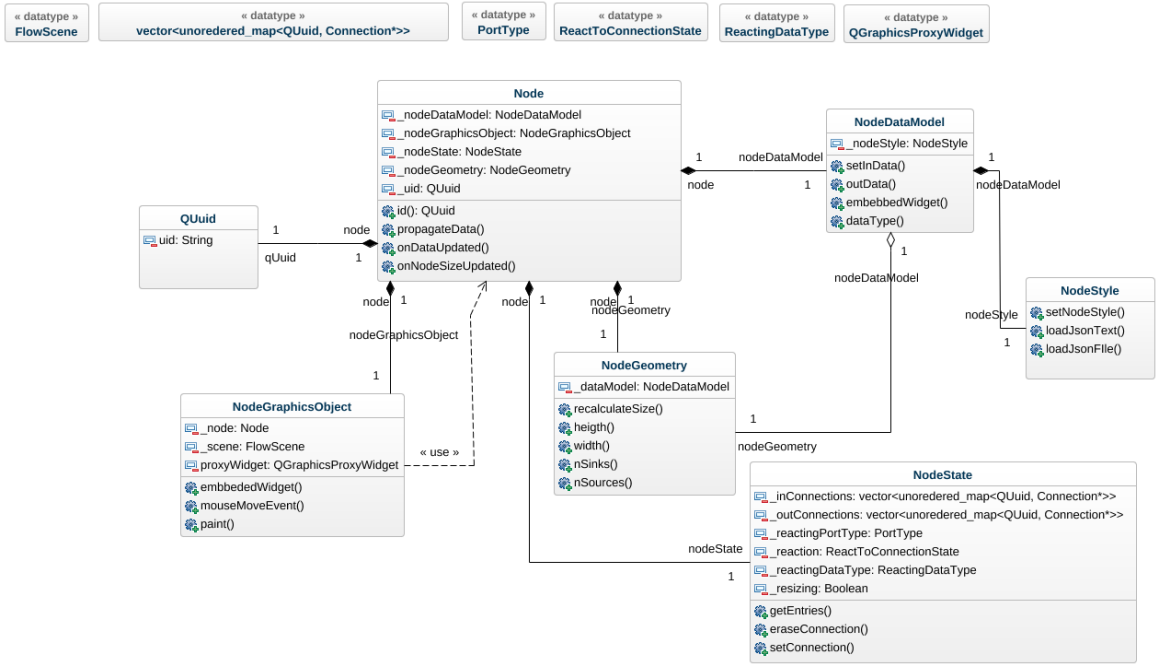


Figura 34. Diagrama de Clases que implementa la arquitectura de Nodos.

Lo primero que se observa es el empleo del paradigma de la programación orientada a objeto. Gracias a este, las diferentes funciones del sistema quedan bien definidas en las distintas clases, permitiendo que el proyecto pueda crecer de forma correcta.

En el diagrama de clases de la Figura 34 se observan las Siete Clases que componen a este *Nodo*.

- **QUuid**. Esta clase es propia de Qt. Se encarga de generar y guardar identificadores Universales únicos. Su nombre proviene de: "*Universal Unique Identifier*". De esta forma cada nodo tiene un identificador único cuando es creado.
- **Node**. Es la Clase principal en la arquitectura. Se puede ver a esta clase como una especie *Manager*, es decir, es la encargada de gestionar a las demás, pero son las otras clases, las encargadas de realizar las diferentes acciones. En el diagrama de clases, se observa como esta tiene como atributos, una instancia de las demás clases. Esto es otro indicio indicando que se trata de una clase Gestora.
- **NodeGraphicsObject**. Esta clase es muy importante, debido a que los únicos objetos que se pueden colocar en una escena son objetos del tipo *QGraphicsObject*. Esta clase hereda de estos objetos. En posteriores secciones, se explicará el funcionamiento de la escena. Pero por ahora, para entender la necesidad de esta clase se debe pensar que la escena es un tipo especial de Widget, el cual permite la inserción e interacción de objetos entre sí. Pero estos objetos no son widgets en sí mismos, si no que heredan de un objeto especial llamado *QGraphicsObject*. Básicamente, la escena es como un "*2D Graphic Engine*". Entonces ¿cómo es que los nodos tienen Widget? La respuesta la da el atributo *QGraphicsProxyWidget*, presente en el diagrama. Esta clase es proporcionada por Qt y sirve para embeber un Widget dentro de *QGraphicsObject*.

- **NodeGeometry**. Esta clase se encarga de gestionar la geometría del nodo. Cuando se realizan diferentes acciones como: darle un nombre al nodo, especificarle el número de puertos de entrada/salida, embeberle un widget dentro del nodo… provocan que las dimensiones del nodo sean aleatorias. Esta clase es la encargada de calcular las dimensiones necesarias para visualizar correctamente el nodo. Además de actualizar sus dimensiones si se redimensionase manualmente el nodo.
- **NodeState**. Esta clase se encarga de gestionar el estado interno del propio nodo. Las conexiones, se encargan de gestionarlas otras clases, las cuales se comentarán en próximas secciones. Sin embargo, esta clase también actúa de nexo entre las clases encargadas de las conexiones y el nodo.
- **NodeDataModel**. Esta es la clase más importante de todas, por el simple hecho de que cualquier bloque que se desee crear tendrá que heredar de esta. En próximas secciones se explicará en profundidad el funcionamiento de esta clase, pero por ahora se tendrá que tener la idea de que esta clase es donde se definirán ciertos aspectos tales como: la cantidad de puertos de entrada/salida, el widget que se desea embeber, la información que es enviada por los puertos de salida, el tratamiento de la información recibida por los puertos de entrada, etc.
- **NodeStyle**. El nombre de la propia clase da la idea de que será la encargada de personalizar el estilo visual del nodo. Además, esta clase puede ser completamente opcional, es decir, el sistema proporciona una por defecto. Pero si se desease dar un aspecto visual al nodo diferente al estándar, se tendría que proporcionar al sistema una versión personalizada de esta clase.

3.2.3 Arquitectura de las Conexiones

El Diagrama de Clases de las Conexiones se tiene en la Figura 35.

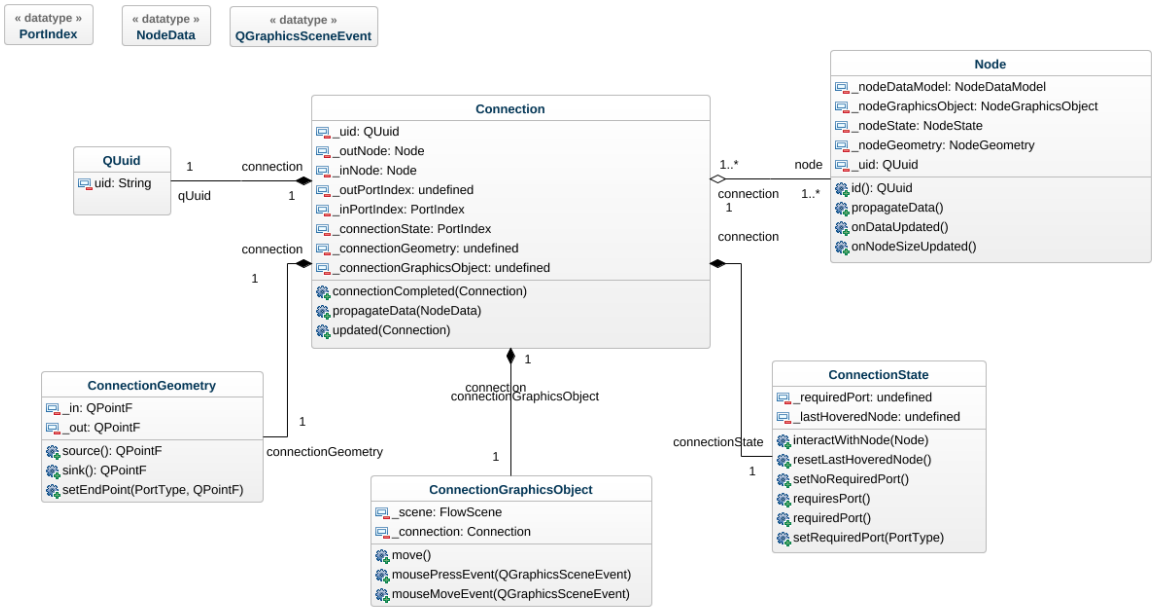


Figura 35. Diagrama de Clases que implementa la arquitectura de las Conexiones.

Se puede ver rápidamente la similitud que presenta con la arquitectura de los Nodos. Esta similitud es completamente normal, puesto que al final ambos son objetos que comparten características comunes.

La arquitectura esta compuesta por seis clases:

- **Connection**. Es la encargada de gestionar todos los aspectos relacionados con la Conexión, delegando diferentes tareas en otras clases. Básicamente, al igual que pasaba con el Nodo, la clase actúa como una clase gestora.
- **Node**. Aquí es donde se ve la relación entre la Conexión y el Nodo. Se puede observar la

relación de agregación entre el Nodo y la Conexión. Esto quiere decir que, si borramos la conexión, el nodo puede seguir existiendo. Mientras que el resto de componentes presenta una relación de composición. Al borrar la Conexión, toda clase que presente una relación de composición será borrada automáticamente. Un último aspecto a comentar es la Multiplicidad. Solo para el caso del Nodo, la multiplicidad es del tipo 1 o * (muchos), debido a que una Conexión conecta dos Nodos.

- **ConnectionState**. Esta clase es la encargada de gestionar el estado interno que presenta en todo momento la conexión.
- **ConnectionGraphicsObject**. Esta clase es la encargada de gestionar todos los aspectos relacionados a la Scene. Además, como se ha comentado en secciones anteriores, es la que permite poder colocar la propia conexión dentro de la escena.
- **ConnectionGeometry**. Esta clase, al igual que en la arquitectura de los Nodos, es la encargada de gestionar todos los aspectos geométricos de las conexiones. Se ve que dos de los atributos de la clase son las posiciones geométrica de ambos extremos de la conexión. Estos parámetros son los empleados para generar la forma (curva) que une ambos extremos del nodo.
- **QUuid**. Al igual que pasaba con los Nodos, las conexiones también tienen asociadas un identificador único universal.

3.2.4 Arquitectura de la Escena

El Diagrama de Clases de la Escena se observa en la Figura 36:

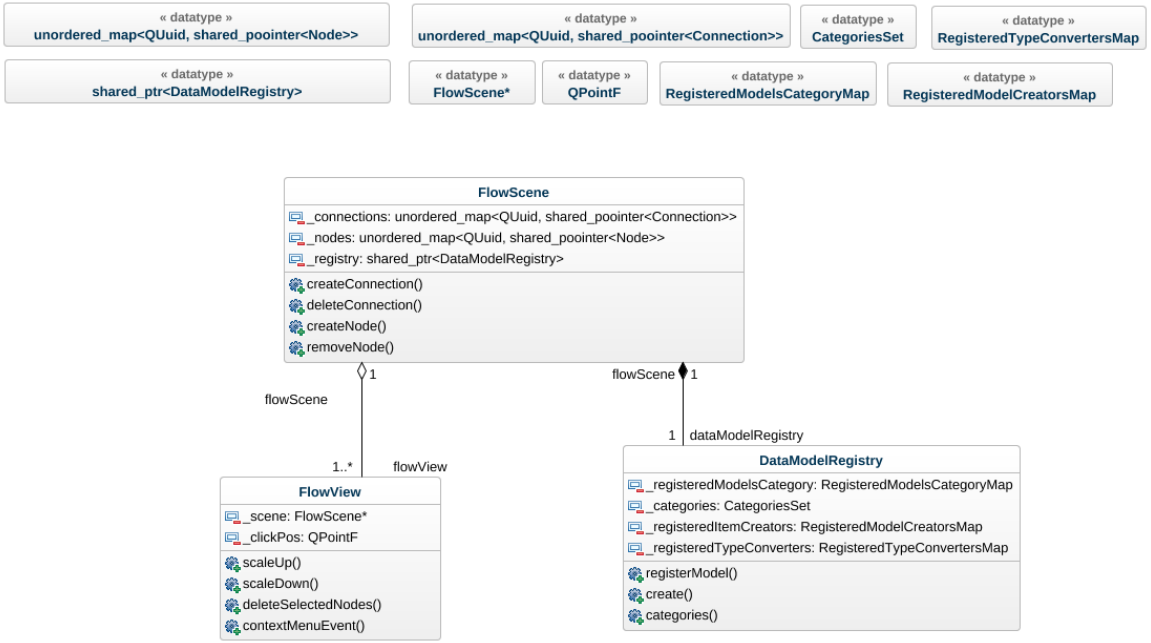


Figura 36. Diagrama de Clases que implementa la Arquitectura de la Escena.

La Escena está compuesta por 3 clases:

- **FlowScene**. Aquí es donde se guardan los distintos Nodos y Conexiones que se van creando. Se podría ver a esta clase como una caja de arena, en ella se guarda todo lo que se va creando. Además, se observa que proporciona métodos para la gestión de estos objetos en la escena, tales como "*createNode*", "*removeNode*", "*createConnection*", etc. Esta clase hereda de *QGraphicsScene*, la cual es el objeto que nos proporciona Qt para crear este tipo de widgets. Tal como se comentó en la arquitectura del Nodo, todos los objetos que se metan en la escena tienen que ser forzosamente del tipo *QGraphisObject*. De ahí que cuando se comentó la arquitectura del Nodo, una de las clases que entraba en juego era *QNodeGraphicsObject*.

Finalmente, comentar que el usuario no interactúa directamente con la escena. Para interactuar con la escena directamente usamos la clase FlowView.

- **FlowView**. Se debe ver a la escena como el área total disponible. Sin embargo, no tendría mucho sentido mostrar todo el área disponible, puesto que sería muy costoso a nivel de rendimiento. En su lugar, se suele crear una región en la cual solo se muestra una parte de la escena. Obviamente, esta vista puede ser desplazada y hacerse zoom sobre ella. Una analogía sería la propia televisión. En un programa emitido en directo, existe todo un plató. Sin embargo, no se ve constantemente todo el plató, sino que se ve una pequeña región del mismo, debido a que la cámara está enfocando a una única región. Pues para el presente caso es similar. La escena sería el plató de televisión, mientras que la vista sería la cámara, la región que se visualiza por la televisión. Al igual que un plató dispone de varias cámaras, aquí podemos crear varias vistas asociadas a la escena.

  Se Interactua con la escena gracias a la vista. Cualquier acción que es realizada en la interfaz, se realiza en realidad sobre la vista, y esta es la encargada de transmitirla hacia la escena. Una de las acciones más importantes es la creación del Menú Contextual. Esta acción es la encargada de crear el menú que nos muestra los diferentes nodos que se pueden emplear. Además de indicarle a la escena que cree el nodo seleccionado.

- **DataModelRegistry.** Cada nodo que se crea, debe ser registrado para que este aparezca en la lista de nodos disponibles. Además, cuando se crea un nuevo nodo, esta clase es quien realmente se encarga de crear al nodo y devolvernos el objeto ya creado.

3.2.5 Arquitectura de las Operaciones

El Diagrama de Clases correspondiente a las Operaciones se muestra en la Figura 37:

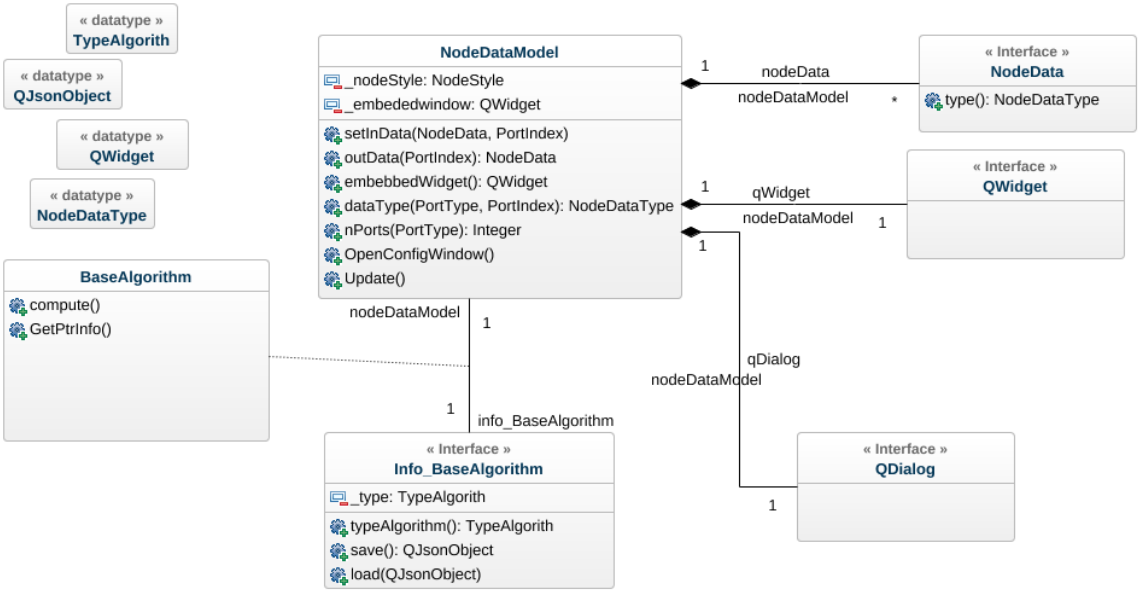


Figura 37. Diagrama de Clases de la Arquitectura de las Operaciones.

En próximas secciones se creará un diagrama de clases más concreto para explicar el funcionamiento de cada bloque implementado. Sin embargo, para el caso que nos ocupa, explicar la arquitectura de las operaciones, se ha usado una forma generalizada de explicar el funcionamiento. De ahí que la mayoría de las clases presenten la palabra Interface.

Se comenzará explicando el funcionamiento de la Clase "NodeDataModel". Esta clase es proporcionada por la librería de los Nodos. Sin embargo, no es posible usar directamente esta, ya que la clase actúa como una interfaz. Es decir, todos los Nodos que se vayan a crear tienen que heredar de esta clase. Además, se tiene que reimplementar los diferentes métodos de la propia clase para que hagan lo que se desee.

En el diseño de la clase heredada, somos libres de elegir los datos que se van a guardar. De ahí que presente una relación con la clase "NodeData". Se pueden guardar los NodeData que se desee. Esto explica la relación a muchos (*).

Los métodos más importantes que tenemos que reimplementar en la clase "NodeData" son:

- **nPorts().** Aquí se indica el número de Puertos de Entrada y Salida que presenta el Nodo.
- **dataType().** En este método se tiene que especificar el tipo de dato asociado a cada puerto.
- **embebbedWidget().** Este método será siempre el mismo en todos los Nodos. Solo debemos asegurarnos que el puntero devuelto sea el Widget que se quiere embeber en el Nodo. Este diseño de Widget es completamente libre. Se podrá embeber cualquier diseño de widget que se desee.
- **setInData().** Ya que el sistema funciona por eventos, no se sabe el momento exacto en que una nueva información llega por los puertos de entrada. Este método se ejecutará cada vez que un nuevo dato llegue por alguno de los puertos de entrada. Simplemente se tendra que especificar al Nodo que es lo que se desea hacer con ese dato que ha llegado.
- **outData().** El disparo de este método debe ser controlado manualmente. La librería de Nodos, proporciona una "*SIGNAL*", especial llamada "*dataUpdate(port)*". Somos los responsables de lanzar esta "*SIGNAL",* indicando en que puerto de Salida se va a transmitir el Dato. Una buena práctica, es implementar la lógica fuera de este método. Lo idóneo, es crear un "*NodoData"* por cada puerto de Salida que se disponga en nuestro diseño del Nodo. Así, cada vez que se invoque a este método lo único que se hará en él, será enviar el dato asociado al "*NodeData"* específico.

Los otros dos métodos, no son proporcionados por defecto en la librería. Son métodos creados para el desarrollo de este proyecto.

- **OpenConfigWindow().** Si se desea que el Nodo creado muestre una ventana de configuración al hacer doble-clic sobre el propio Nodo, entonces se empleará este método para tal fin. Como su nombre indica, este método primero verificará si la ventana de configuración ha sido instanciada. Si no es el caso, el método creará la ventana y luego la mostrará.
- **Update().** Es el método encargado de actualizar el estado interno de la información en el Nodo. En el diseño creado para este proyecto, se ha optado por separar la lógica de los datos. Así que, el método comentado anteriormente, "*setInData()"*, su único cometido es verificar que el dato que le llega por el puerto es válido. Si el dato es válido, lo asigna a su correspondiente atributo, *NodeData*. Acto seguido, llama al método "*Update"*. Este método es el encargado de crear una instancia de la clase "*BaseAlgorithm"* derivada, pasarle todos los datos que presente el Nodo en el estado actual y pedirle a la Clase "*BaseAlgorithm"* que procese los datos. Finalmente, pide los resultados, los guarda en los atributos de salida y ejecuta la "*SIGNAL*" "*dataUpdate(port)*".

La clase "*NodeData"*, se utiliza para definir el tipo de dato que se le va a pasar a los diferentes puertos. Se puede interpretar como una especie de molde que se emplea para enviar cualquier tipo de dato. Dentro de este molde, se podrá poner cualquier tipo de dato que se desee. De ahí, la existencia del método "type()", el cual sirve para identificar el tipo de dato que se ha asociado a este molde.

La clase "*QDialog",* en realidad es opcional. Debido a que se puede dar el caso de que no se requiera agregarle a un Nodo una ventana de Configuración.

Algo similar sucede con la clase "*EmbebbedWidget"*. Puede que en el diseño de un Nodo no se requiera del uso de un widget embebido. Cuando el Nodo es simple se le suele asociar un simple "*QLabel"* con una imagen como widget embebido generalmente.

En las figuras 3-10 y 3-11, se especifica el conjunto de procesos que realizan la mayoría de los Nodos que embeben una operación.

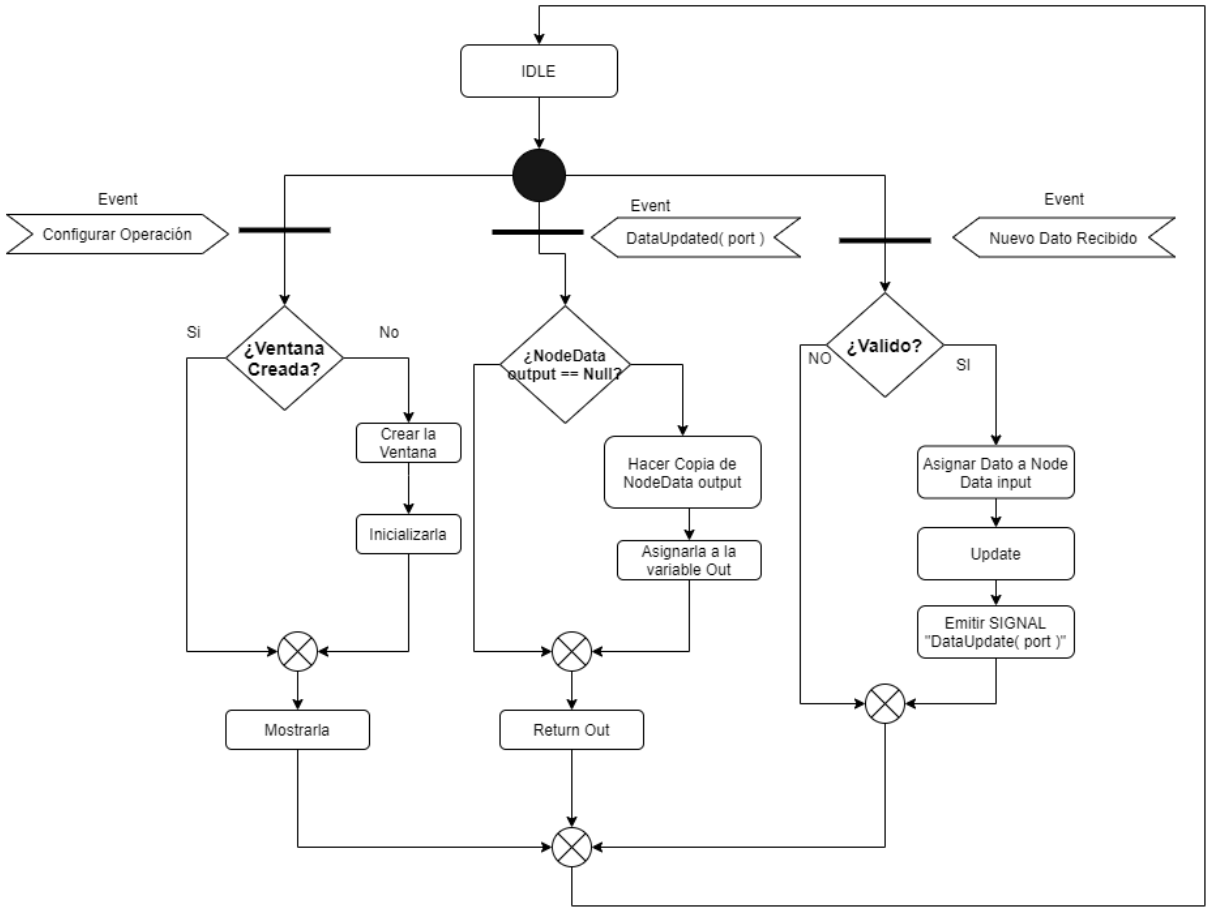


Figura 38. Diagrama de Procesos que muestra la Forma básica de Operación de la Mayoria de los Nodos que embeben una Operación en su Interior.

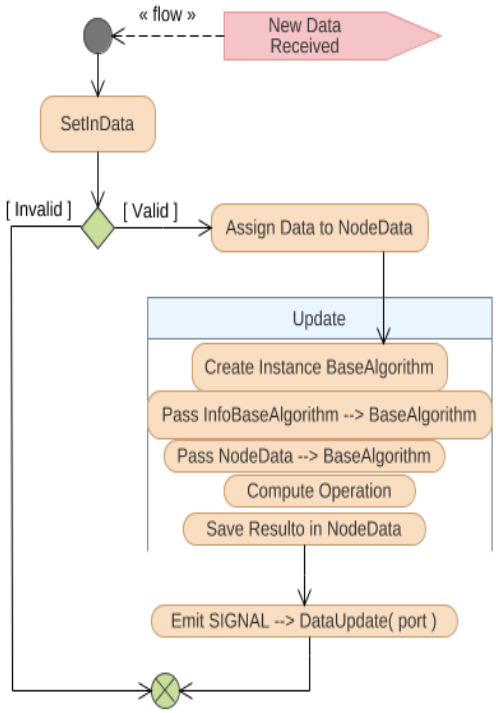


Figura 39. Ejemplificación del Funcionamiento del método Update.

3.3 Bloques Implementados

En los siguientes apartados se va a proceder a explicar en profundidad el funcionamiento interno de los diferentes Nodos. Explicando las diferentes clases que los componen y la forman en la que se conectan entre ellas.

3.3.1 Nodo MultiMedia Loader

Este bloque es considerado un bloque fuente porque se encarga de proporcionar imágenes para el resto de bloques por su único puerto de salida. En las sucesivas figuras, se verán las diferentes combinaciones del bloque:

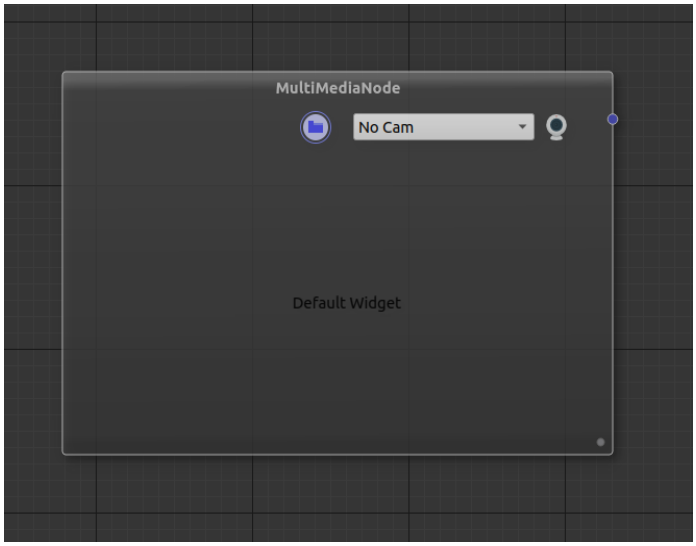


Figura 40. Nodo Multimedia en su estado por defecto.

En la Figura 40 se tiene al bloque "*MultiMedia*" en su estado básico. Arriba a la derecha, tiene el puerto por el cual se conecta a otro Nodo (círculo azul). Este es el puerto por donde se obtienen las Imágenes. Abajo a la derecha, tiene un pequeño punto gris. Este punto se emplea para modificar las dimensiones del bloque.

Debajo del nombre del bloque, se tiene de izquierda a derecha los siguientes componentes: un botón, un *Combobox* y otro botón. El botón con el símbolo de la carpeta azul, sirve para cargar un video. El botón con el símbolo de la webcam, se emplea para permitir que el sistema busque las cámaras web disponibles a las que se puede conectar. La forma de proceder con las cámaras web es la siguiente. Primero, se pulsa el botón de la webcam (imagen de webcam). Entonces, el sistema se encargará de buscar las cámaras web disponibles. Acto seguido, el componente *Combobox* cambiará. Al pulsar sobre él, se desplegará una lista con las cámaras web disponibles. Al pulsar sobre cualquiera de las opciones disponibles, el sistema automáticamente se conectará a la webcam que se haya seleccionado, y en la parte central del Nodo se cargará la Interfaz de la Webcam.

En la Figura 41 se tiene la interfaz de la Webcam. En la zona central se muestra la imagen capturada por la propia webcam. En la parte inferior, por orden de aparición de izquierda a derecha, se tienen los siguientes componentes:

- Botón para tomar una instantánea. Al pulsarlo se abrirá una ventana solicitando el nombre y la ruta donde se desea guardar la imagen.
- Botón para Congelar/Descongelar la Webcam.
- Botón para realizar una Grabación. Esta opción no está implementada debido a que no se consideraba imprescindible para el proyecto, pero se ha colocado para futuras modificaciones.
- Un *Spinbox* donde se podrá configurar el FPS de la Webcam. Su rango va de 30 a 0.1.
- Nombre de la webcam a la que estamos conectados.

Figura 41. Interfaz de la Webcam del Nodo Multimedia

Si en lugar de conectar la webcam, se desea cargar un video, se debe pulsar sobre el botón con la carpeta azul y acto seguido se abrirá una ventana solicitando el archivo a cargar. Al seleccionar un archivo válido, se cargará automáticamente la interfaz del Reproductor de Video, Figura 42.

Figura 42. Interfaz del repropudctor de Video del Nodo Multimedia.

En la zona central se tiene la imagen del video. Debajo de esta, se tiene una barra indicando la duración total del video, además de la posición en la que se encuentra este. Arrastrando la bola roja hacia adelante o hacia atrás, se puede adelantar o atrasar el video. Los diferentes elementos presentes en la parte inferior son:

- El botón de Play
- El botón de Pause

- El botón de Stop
- El botón de Bucle. Al pulsarlo, el video se reproducirá una y otra vez (modo bucle activado). Pulsando nuevamente sobre el botón se desactivará el modo bucle.
- *Spinbox* para configurar el FPS del reproductor.

En la Figura 3-17, se explica en el Diagrama de Procesos, el conjunto de pasos que realiza el Nodo Multimedia para gestionar las diferentes Interfaces. La interfaz de Video y de la Cámara.

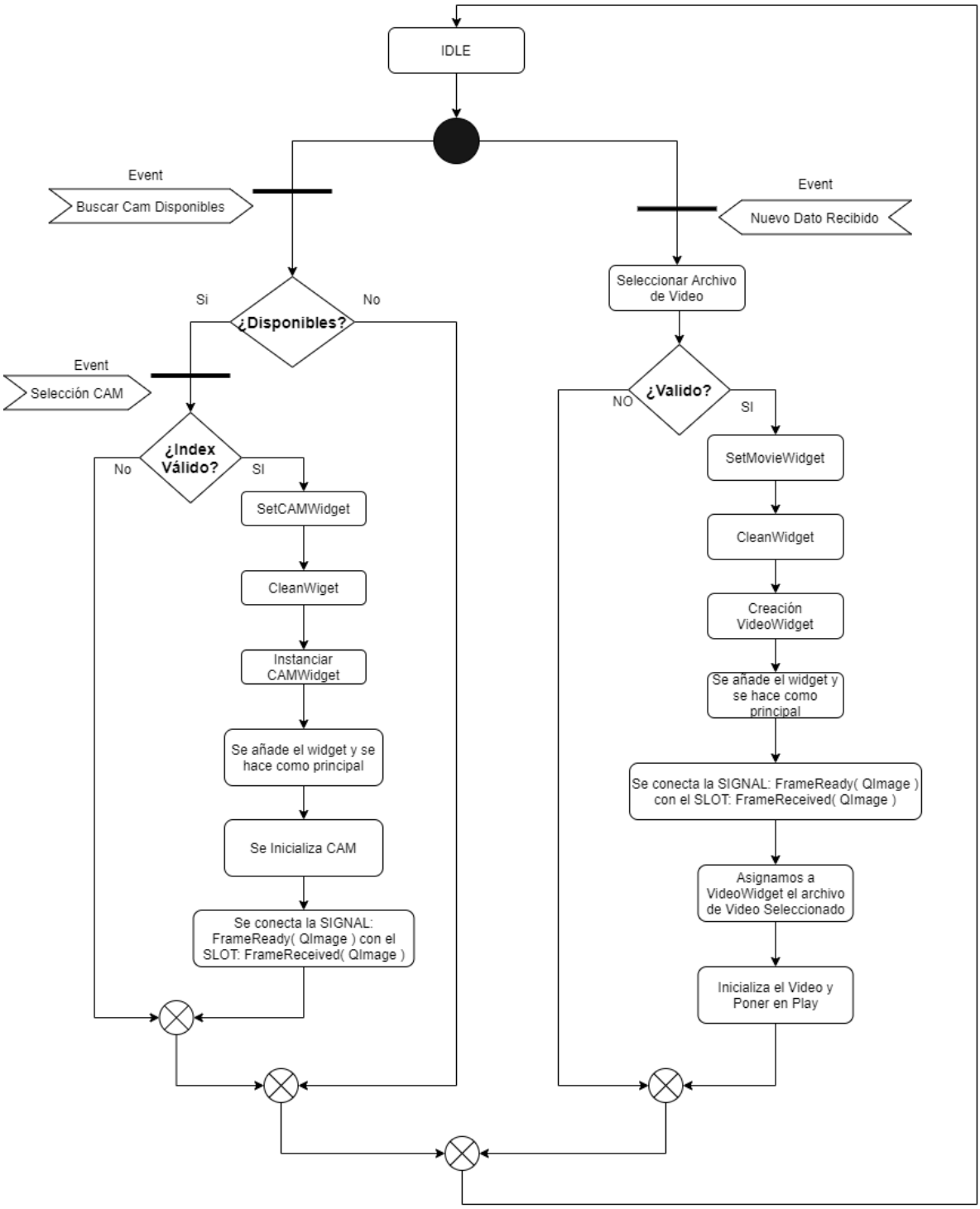


Figura 43. Diagrama de Funcionamiento del Nodo MultiMedia para Gestionar la Interfaz de Archivos de Vídeo o la Interfaz de la Cámara.

En el Diagrama de Clases de la Figura 44 se tienen a las clases involucradas en el funcionamiento de este Nodo. La clase principal es el "*MultiMedia Model*", el cual hereda de "*NodeDataModel*" (flecha con cabeza blanca). En esta clase se debe reimplementar un conjunto de métodos heredados de la clase "*NodeDataModel*", para crear la configuración deseada.

Hay un conjunto de métodos heredados que suelen ser comunes. Estos suelen servir para:

- Indicarle el número de puertos de entradas y salidas.
- Indicar la forma en la que se debe tratar la información recibida por el puerto de entrada. En concreto este nodo no dispone de puertos de entrada.
- Indicar el tratamiento se hace a la información a la salida.
- Indicar el tipo de dato que manejará cada puerto.

Un método propio de esta clase, no heredado, es "*Update_Output_Data (QImage)"*. Este método se encarga de recibir una imagen (Frame) y enviarla por el puerto de salida. Para una mejor comprensión del funcionamiento de este método, se realizan las siguientes aclaraciones:

1. **MultiMedia Model** posee dos variables internas: *ImageData* y *InputVieoWidget*
2. **ImageData** es la clase enviada por el puerto de "*MultiMedia Model"*. Esta clase hereda de "*NodeData"*. Además, posee una variable interna "cv::Mat", la cual es la variable encargada de guardar la Imagen.
3. ***"InputVideoWidget***" es el widget que embebemos dentro del Nodo.

Por lo tanto, ¿Cuándo es ejecutado el método "*Update_Output_Data"*? Al instanciar el Nodo (MutiMedia Model), en el constructor se realiza una conexión entre la "***SIGNAL***" "*FrameRead(QImage)*" de la clase "*InputVideoWidget"* y el "***SLOT***" "*Update_Output_Data*" de la clase "*MultiMedia Model*". De esta manera, cada vez que el widget, el cual ha sido embebido en el Nodo, emite su "***SIGNAL***" se ejecuta automáticamente el "***SLOT***". Una vez que se recibe una imagen, se inserta en la clase ImageData y se envia esa clase por el puerto de salida.

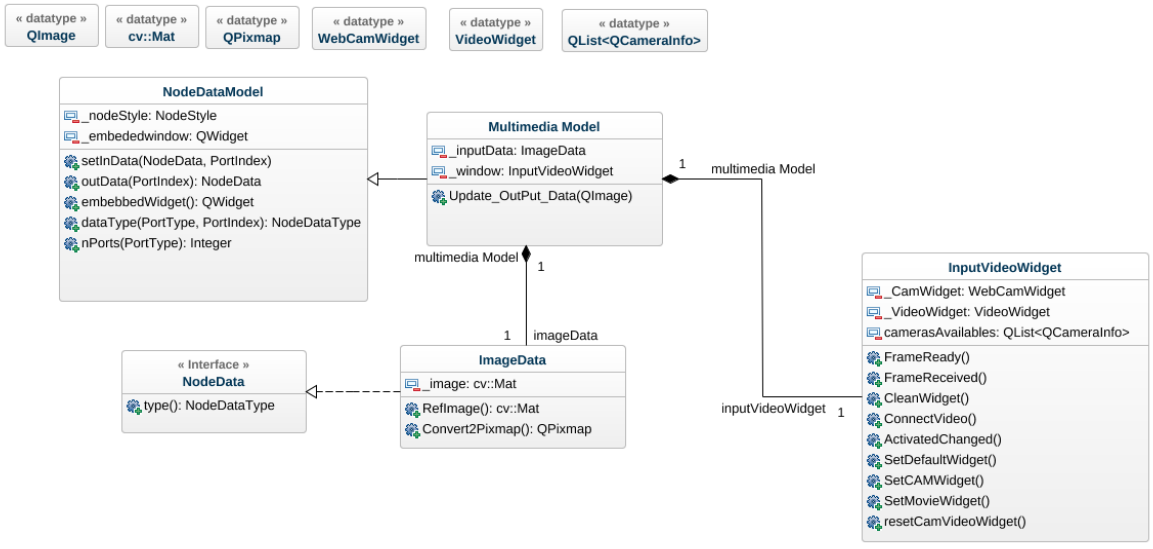


Figura 44. Diagrama de Clases con la Implementacion del Nodo Multimedia.

Finalmente, solo queda explicar el funcionamiento de la Clase "*InputVideoWidget*". Es el widget embebido dentro del Nodo. Los dos primeros métodos visibles en la clase son: "*FrameReady (QImage)"* y "*FrameReceived (QImage)"*. El primero se trata de una "***SIGNAL***", mientras que el segundo se trata de un "***SLOT***". La misión de este "***SLOT"*** es emitir la "***SIGNAL***" con la imagen recibida nuevamente. Es decir, la imagen se vuelve a reenviar.

El tercer método: *CleanWidget()*, se encarga de limpiar el widget central y colocar el widget por defecto. *InputWidget* posee como widget central a un widget especial cuyo funcionamiento es el de actuar como un contenedor de widgets. Pero este únicamente muestra un solo widget al mismo tiempo. Habrá que indicarle cual es el widget que se desea mostrar. Es decir, es como si se tiene a disposición varias ventanas en las cuales se guardan diferentes widget. En cualquier momento se puede escoger cuál de estas se va a visualizar (igual que el funcionamiento que un teléfono móvil).

Los siguientes dos métodos: *ConnectVideo()* y *ActivatedChanged(int idx)*, son en realidad dos "***SLOT***" que se ejecutan cuando se selecciona un video a cargar o una webcam se conecta respectivamente. Es decir, cuando se pulsa sobre el botón de la carpeta azul se emite una "***SIGNAL****"*, la cual es conectada al método *ConnectVideo*. De igual forma sucede con la webcam, cuando en el *Combobox* se selecciona una webcam, se emite una "***SIGNAL***", la cual esta conectada al "***SLOT***" *ActivatedChanged(int idx)*.

Los pasos que siguen los siguientes métodos son los siguientes:

ConnectVideo()

1. Se abre una ventana solicitando el archivo de video a cargar.
2. Se verifica si el archivo es válido.
3. Se ejecuta el método *SetMovieWidget()*
    a. Se ejecuta el método *CleanWidget()*
    b. Se crea una instancia de "*VideoWidget"*
    c. Se inserta en el contenedor de widget el "*VideoWidget*" y se coloca como widget principal
    d. Se conecta "***SIGNAL***" { *FrameReady(QImage)* } del reproductor MultiMedia, con el "***SLOT****"* { *FrameReceived(QImage)* } del InputVideoWidget.
4. Se asigna a "*VideoWidget*" el archivo de video a reproducir
5. Se inicializa el Video y se pone en modo Play

ActivatedChanged(int idx)

1. Se verifica que el idx es válido.
2. Se verifica la existencia de cámaras disponibles.
3. Se verifica si el idx se corresponde en el *Combobox* con la opcion "*NoCam*".
4. Se ejecuta el método *SetCamWidget()*
    a. Se ejecuta el método *CleanWidget()*
    b. Se instancia a "*CamWidget"*
    c. Se inserta el "*CamWidget"* en el contenedor de widgets y se le hace como Widget principal.
5. Se inicializa la "*WebCam*" de la clase "*CamWidget"*
6. Se conecta la "***SIGNAL"*** { *FrameReady(QImage)* } de la webcam del "*WebcamWidget*", con el "***SLOT****"* { *FrameReceived(QImage)* } del "*InputVideoWidget*".

Solo quedaría explicar el funcionamiento de los Widgets encargados de gestionar el video y la cámara web. El Diagrama de Clases de la Figura 45 muestran el conjunto de clases empleados por esos Widgets y la relación que tienen con las diferentes clases que lo gestionan.

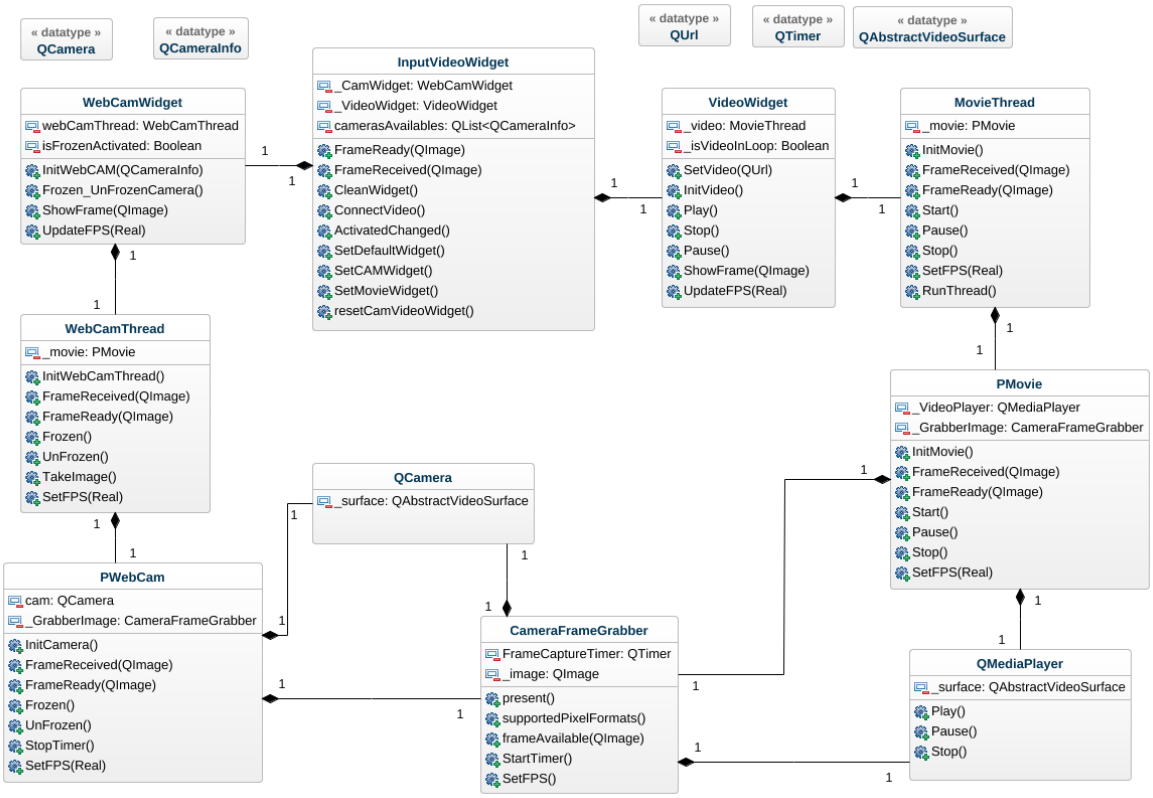


Figura 45. Diagrama de Clases con la Implementación del Widget InputVideoWidget.

Primero se analizará la rama de la derecha (VideoWidget). La clave en esta clase es la presencia de la variable *MovieThread*. Esta clase es la encargada de reproducir el vídeo. La Clase *VideoWidget* actúa como una clase interfaz, la cual permite visualizar el propio vídeo y la posibilidad de interactuar con el mismo, gracias a los diferentes métodos como: *Play, Stop, Pause, UpdateFPS* etc.

Si se observan los métodos de las Clases *MovieThread* y *PMovie* se aprecia que son prácticamente los mismos. Esto se debe a que son la misma clase excepto por un detalle. *MovieThread* es igual a la clase *PMovie*, ejecutándose dentro de un *Thread*. Se ha decidido que debido a la alta velocidad a la cual se tienen que enviar frames, era una buena idea tener al video ejecutándose en un hilo independiente. De esta manera el hilo de la interfaz puede ir más fluido al no sobrecargarlo con más operaciones de la cuenta.

La clase *PMovie*, posee dos variables: *QMediaPlayer* y *CameraFrameGrabber*. Estas son las clases claves encargadas de procesar el *framerate*. *QMediaPlayer* la proporciona Qt para poder cargar archivos multimedia. Esta clase debe apoyarse en la otra, *CameraFrameGrabber*, para su funcionamiento. En resumidas cuentas, *QMediaPlayer* se encarga de gestionar el estado interno del archivo multimedia. De ahí los métodos: *play*, *pause* y *stop.* Pero esta clase no trata directamente con los datos, en nuestro caso los Frames. La clase *CameraFrameGrabber* es la encargada de tratar directamente con los Frames del archivo multimedia.

El método más importante de la clase *CameraFrameGrabber,* la cual hereda de *QAbastractVideoSurface,* es "*present()*". Este método recibe el "buffer" que contiene el siguiente Frame del archivo multimedia. En este se debe reescribir el comportamiento del mismo para indicarle que es lo que se desea hacer sobre el "buffer". En nuestro caso, se usa el "buffer" para crear una QImage. Una vez la imagen ha sido creada, se envía a las capas superiores usando el sistema de ***SIGNAL*** y ***SLOTS***.

En esta última clase comentada, *CameraFrameGrabber,* presenta una variable del tipo *QTimer*. Esta variable permite instanciar un Cronómetro al cual se le pueden configurar diferentes parámetros como: tipo de disparo y tiempo entre disparos. Este timer es el encargado de controlar los FPS. El truco utilizado para controlar los FPS a voluntad, es configurar el tiempo entre disparos de este Timer. La velocidad a la que se reciben los Frames, método "*present()"*, no puede ser configurada puesto que esto viene marcado por el propio video. Sin embargo, sí se tiene control sobre el momento en el cual se decide muestrear un Frame. Es decir, el timer se encarga de indicar cuando procesar un frame. Cada vez que el timer se activa, se activa a la vez una *flag*. Cuando el método

"*present()*" se ejecuta, lo primero que se hace es comprobar si la *flag* esta activada. En caso de que la *flag* este activada, el Frame es procesado. En caso contrario, se descarta sin procesarlo. Acto seguido de que se procese un Frame, la *flag* se resetea y se envía el frame resultante a las capas superiores, empleando el sistema de ***SIGNAL & SLOT***.

Al atender al diagrama de clases de la Figura 45, se puede apreciar el camino que el Frame realiza hasta llegar a la clase *VideoWidget*. El frame es generado en la clase *CameraFrameGrabber*. De aquí pasa a la clase *PMovie*. Después a la Clase *ThreadMovie*, para finalmente llegar a la clase *VideoWidget*. Es decir, las distintas clases cuando reciben un nuevo Frame, lo vuelven a re-enviar a la clase inmediatamente superior en la cadena.

El mecanismo de funcionamiento de la WebCam es exactamente igual al mecanismo del Video. La única diferencia es la variación en algunos de sus métodos. Por ejemplo, en la WebCam no se tiene los métodos: *play*, *stop* y *pause*. En cambio, existen métodos para: congelar o descongelar a la WebCam, tomar una Imagen desde la Webcam etc.

3.3.2 Nodo ImageShowModel

Este nodo es el encargado de mostrar la imagen recibida por su puerto. Tiene unicamente un puerto de entrada, la imagen recibida. El nodo da la posibilidad de redimensionar la imagen visualizada. En la Figura 46 se observan los diferentes estados que presenta el nodo: puerto conectado (imagen recibida) y puerto desconectado (imagen no recibida).

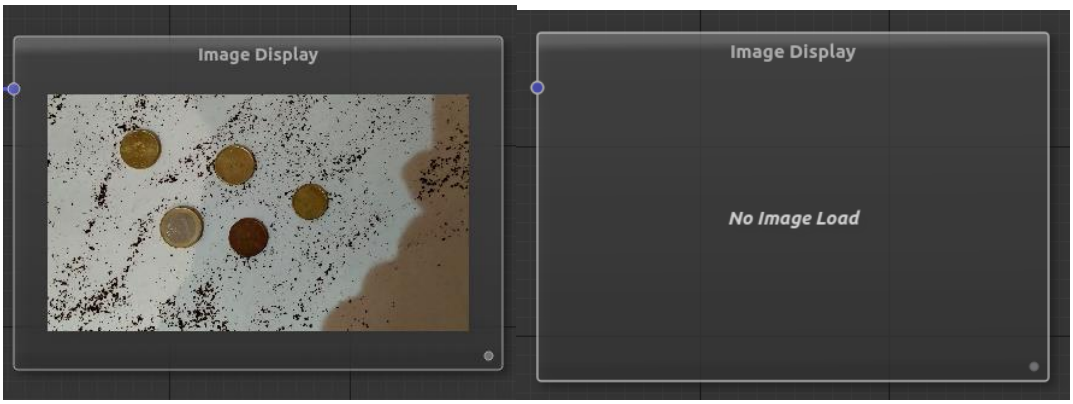


Figura 46. Visualización del Nodo ImageShowModel.

El funcionamiento de este Nodo es relativamente fácil, siendo su diagrama de clases el de la Figura 47.

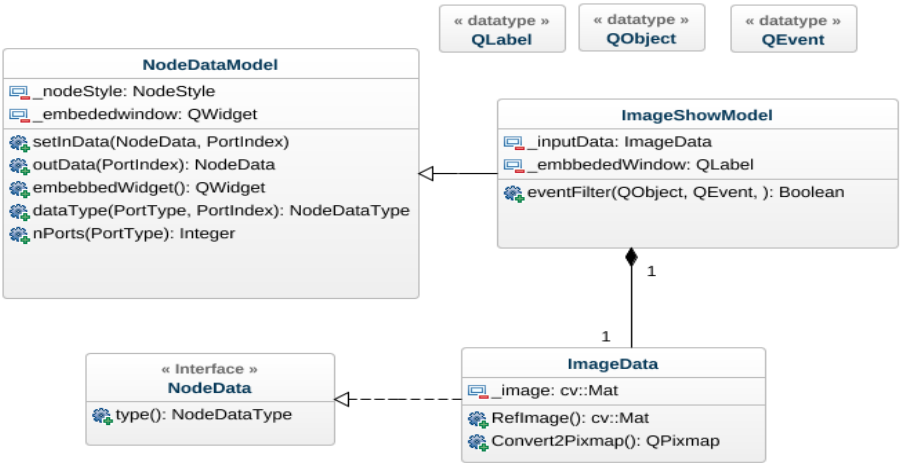


Figura 47. Diagrama de Clases con la Implementación del Nodo ImageShowModel.

En este caso se tiene como widget embebido a un *QLabel*. Debido a la simplicidad de este Nodo, no se ha tenido que desarrollar un widget especializado. Se ha empleado el widget que la propia librería de Qt proporciona.

Se tienen que sobreescribir los métodos esenciales que todo Nodo posee: *nPorts(), setInData(). OutData()*, … tal y como viene siendo habitual en todos los Nodos. El único método particular en este Nodo es *eventFilter()*. Gracias a este método, se puede configurar el comportamiento de diferentes eventos asociados al QLabel. En nuestro caso, se ha configurado el evento encargado del proceso de redimensionado, puesto que el Nodo al crecer deberá redimensionar la imagen que almacena en su interior.

El otro método más importante en este Nodo es *setInData()*. Aquí, al recibir la información por el puerto de entrada se siguen los siguientes pasos:

1. Verificar que el Data ha llegado correctamente.
2. Verificar que la Imagen no esté vacía.
3. Verificar que la variable que guarda la Imagen, efectivamente contiene una Imagen.
4. Asignar a la variable interna del NodeData la imagen que nos ha llegado.
5. Obtener las dimensiones actuales del QLabel embebido en el Nodo.
6. Asignar la imagen que hemos recibido al QLabel, redimensionandola a las dimensiones correctas.

3.3.3 Nodo Threshold

En este nodo se lleva a cabo la primera operación realizada sobre la imagen, *Threshold*. La operación de *Thresholding* es uno de los métodos de Segmentación de Imágenes. El empleo de esta técnica tiene como objetivo la generación de una *Imagen Binaria*, es decir, vista desde la perspectiva del ojo humano solo se verá regiones en blanco o negro. Esta técnica busca aislar las regiones de interés del resto.

El procedimiento llevado a cabo para tal fin es simple. Se suele realizar sobre imágenes en escala de Grises. El proceso consiste en ir recorriendo uno a uno los píxeles que componen la imagen, y para cada píxel comparar el valor de este con un valor de "threshold" proporcionado. Si el valor del píxel es mayor que el del umbral (threhold value), se le asigna un valor específico. En caso contrario, se le asignará otro valor. [13] [15]

La librería de OpenCV nos proporciona una función para realizar esta operación:

```
double cv::threshold(
  cv::InputArray    src,          // Input image
  cv::OutputArray   dst,          // Result image
  double            thresh,       // Threshold value
  double            maxValue,     // Max value for upward operations
  int               thresholdType // Threshold type to use (Example 10-3)
);
```

Figura 48. Función proporcionada por OpenCV para realizar un Threshold.

Además, se disponn de varias configuraciones para el thresholding (variable *thresholdType)*:

Threshold type	Operation
cv::THRESH_BINARY	$DST_I = (SRC_I > thresh)\ ?\ MAXVALUE : 0$
cv::THRESH_BINARY_INV	$DST_I = (SRC_I > thresh)\ ?\ 0 : MAXVALUE$
cv::THRESH_TRUNC	$DST_I = (SRC_I > thresh)\ ?\ THRESH : SRC_I$
cv::THRESH_TOZERO	$DST_I = (SRC_I > thresh)\ ?\ SRC_I : 0$
cv::THRESH_TOZERO_INV	$DST_I = (SRC_I > thresh)\ ?\ 0 : SRC_I$

Figura 49. Tipos de Threshold configurables en OpenCV.

En la tabla de arriba se ven los diferentes tipos de *Threshold* disponibles. Para el primer caso, "*THRESH_BINARY"*, si el pixel es mayor que el valor de threshold, se le asigna a ese píxel el valor *MAXVALUE,* en caso contrario se le asignará el valor de cero. La segunda opción es igual a la primera salvo que invertida. La tercera opción, "*THRESH_TRUNC*", si el píxel supera el valor de thresh, se le asignará un valor específico. En caso contrario se le asignará el valor que tenga ese pixel en concreto. La cuarta opción, "*THRESH_TOZERO*", si el píxel supera el valor de *thresh*, se le asignará el valor que tenga ese píxel en concreto. En caso contrario se

le asignará el valor de cero. La última opción, es idéntica a la cuarta salvo que su resultado es el opuesto. [15]

Como se ha visto hasta ahora, el método de *Thresholding* emplea un valor global para realizar esta operación sobre toda la imagen. Pero se podría pensar en usar un valor distinto de threshold para cada píxel. Este valor podría ser calculado en función de los píxeles cercanos. Esta idea comentada se denomina "*Adaptative Threshold*". El procedimiento es similar al ya comentado, salvo por la peculiaridad de que en esta técnica se aplica un valor de threshold distinto para cada píxel. Este valor es calculado empleando los valores de los píxeles cercanos, siendo los dos métodos empleados para su cálculo: la media o el Gaussiano. [16]

Esta operación se encuentra disponible dentro de las librerías de OpenCV:

```
void cv::adaptiveThreshold(
  cv::InputArray    src,              // Input image
  cv::OutputArray   dst,              // Result image
  double            maxValue,         // Max value for upward operations
  int               adaptiveMethod,   // mean or Gaussian
  int               thresholdType     // Threshold type to use (Example 10-3)
  int               blockSize,        // Block size
  double            C                 // Constant
);
```

Este método arroja mejores resultados bajo condiciones de luz cambiante. En la Figura 50 se ve el resultado que obtenido tras aplicar diferentes combinaciones de esta técnica.

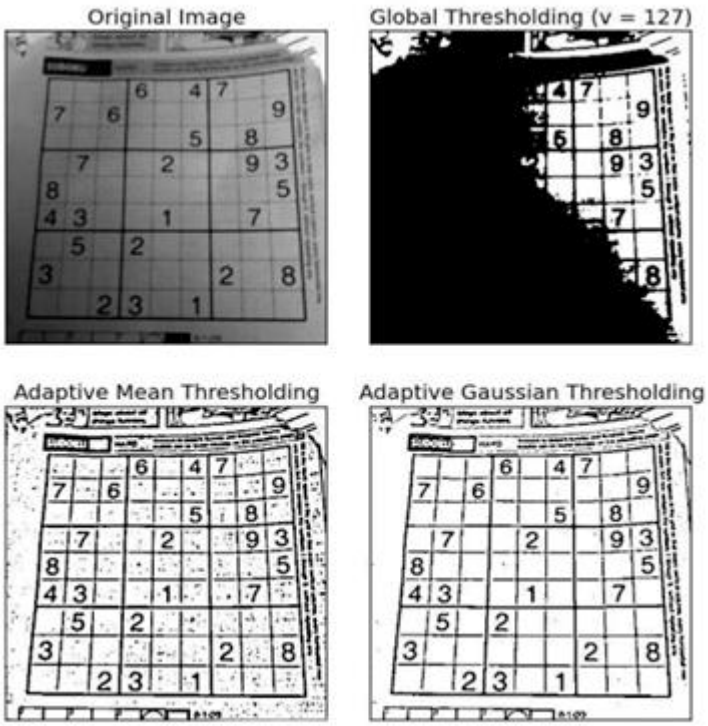


Figura 50. Resultados al aplicar diferentes combiciones de Operaciones de Thresholding.

En las imágenes de abajo se pueden visualizar el Nodo que implementa ambas técnicas comentadas anteriormente. Se aprecia que toda la casuistica de combinaciones distitnas que se le puede configurar, quedan recogidas de una manera simple en la ventana de configuración del Nodo.

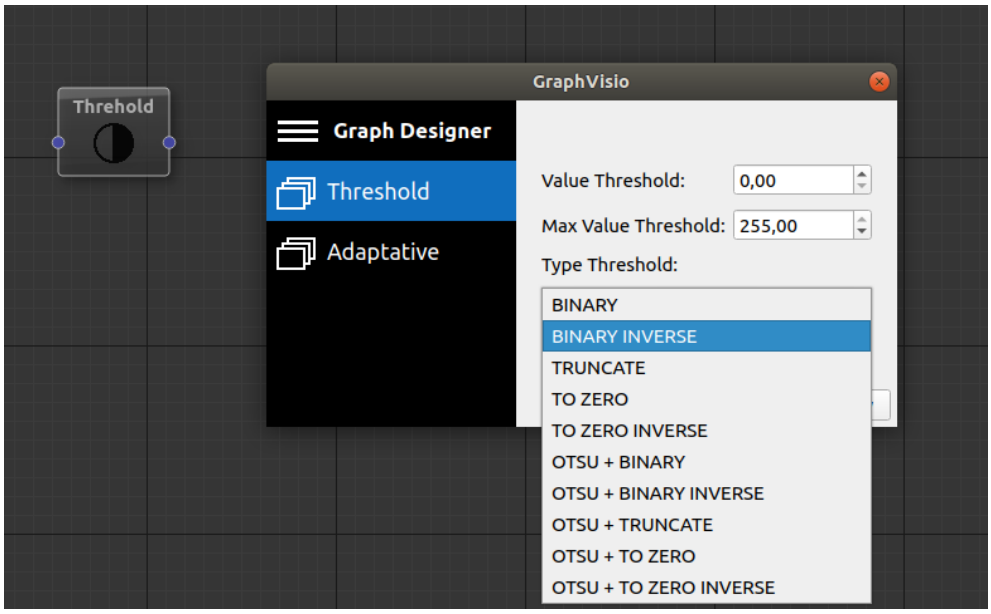


Figura 51. Visualización de la Ventana de Configuración de la Operación de Threshold.

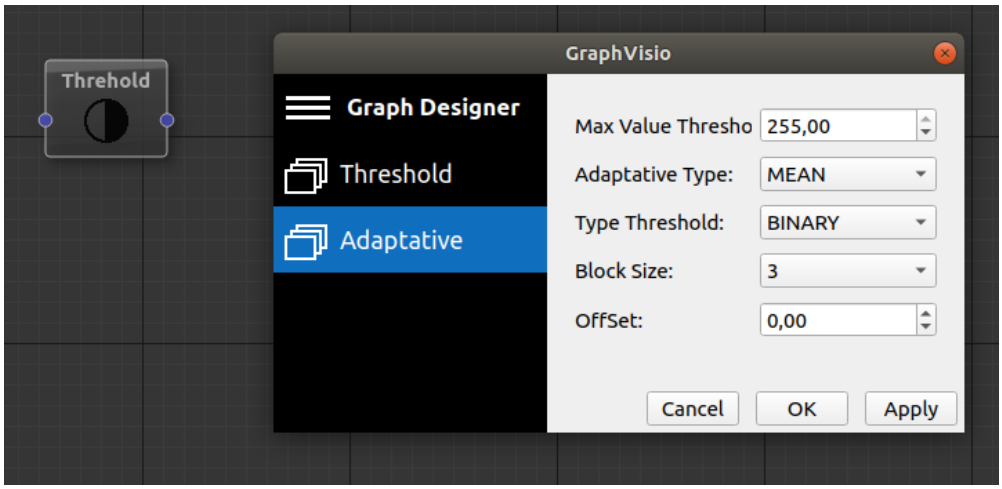


Figura 52. Visualización de la Ventana de Configuración de la Operación de AdaptativeThreshold.

En el diagrama de Clases de la Figura 53 se puede visualizar el conjunto de clases y las distintas interacciones entre clases que se llevan a cabo en la operación.

Se tiene como clase principal a "*ThresHoldModel"*. Esta clase hereda a su vez de "*NodeDataModel*" como se aprecia en el diagrama. Todos los métodos, tanto de la clase "*ThresHoldModel*" como de la clase "*NodeDataModel*", son compartidos. Esto se debe a que la clase "*NodeDataModel*" actua como interfaz. La clase que hereda de esta interfaz está implementando una especializació. Es decir, no se podría, ni tendría mucho sentido, instanciar una clase "*NodeDataModel*" debido a que esta solo sirve de base para construir el resto.

Presenta como atributos:

- _out: ImageData → Clase donde se guarda la Imagen de Salida
- _in: ImageData → Clase donde guardamos la imagen que recibe por el puerto de entrada
- _embebbedWindow → El Widget que se va a embeber en el Nodo.
- _info: ThresHoldInfo → Clase encargada de guardar toda la información de configuración de la operacion de ThresHold
- _configWindow → Widget consistente en la ventana de configuración de la operación.

En los métodos de la clase "*ThresHoldModel*", nos encontramos con los métodos:

- outData(PortIndex): ImageData. → Aquí se define aquello que se desee realizar con los datos que van a ser enviados por el puerto de salida.
- setInData(NodeData, PortIndex) → Aquí se define aquello que se quiera realizar con los datos que se han recibido por el puerto de entrada, indicadole el *PortIndex.*
- embebbedWidget( ): QWidget → En este método solo habrá que indicarle cual es el widget embebido dentro del Nodo.
- dataType(PortType, PortIndex): NodeDataType → Con este método se define el tipo de dato que se envía/recibe por cada puerto.
- nPorts(PortType): int → En este método se definen el número de puertos de entrada/salida que contendrá nuestro Nodo.
- OpenConfigWindow() → Este método será el encargado de instanciar y mostrar la ventana de configuración que puede presentar cada Nodo.
- Update() → Con este método se le indica al Nodo que se desea actualizar el Estado Interno de los Datos.

La clase "*ImageData*", es la encargada de contener la Imagen que se recibe/envía por los puertos de este Nodo. Se puede apreciar que presenta dos métodos:

- RefImage(): cv::Mat → Gracias a este método se puede recibir por referencia la imagen que guarda la clase.
- Convert2PixMap(): QPixMap →Sirve para poder realizar la conversión de "*cv::Mat*" a "*QPixMap*". Un "*QPixMap*" es la clase que se emplea para poder insertar una imagen en un "*QLabel*".

La clase "*ThresHoldConfigWindow*", es la encargada de gestionar a la ventana que se abre al hacer doble clic sobre el nodo. El punto más importante en esta clase es su atributo "*_Ref_Original_Info*". Esta variable es una referencia a la variable que posee la clase "*ThresHoldModel*". No es una copia de la clase, sino una referencia a la misma. Gracias a que se le pasa una referencia, se pueden realizar las modificaciones oportunas sobre la operación.

Se observa que la mayoría de los métodos son lógicos. Por ejemplo: "*Load2GuiInfo*()", se encarga de configurar la ventana en función de la información que tenga previamente configurada la operación.

Los métodos: "*ThresHold_DataDump()"* y "*Adaptative_DataDump()*", necesitan de una explicación. Estos métodos se pueden traducir como: *Volcado de información*. Si se recuerda, la ventana permite configurar la operación. Sin embargo, debajo de la ventana tenemos tres botones: *Aceptar, Aplicar y Cancelar*. Si se recuerda, la información de la operación es recibida a través de una referencia. Por lo tanto, no tendría sentido que cada vez que se cambiase un valor, se actualizasen los datos finales. La técnica empleada consiste en actualizar los datos solo en el caso de pulsar sobre los botones *Apply* o *Ok*.

Finalmente, tenemos a la clase "*ThresHoldInfo*". Esta es la encargada de guardar toda la información relevante a la configuración de la Operación. En su Diagrama de clases se aprecian varias cosas:

- Entre la relación de las clases "*ThresHoldModel*" y "*ThresHoldInfo*", aparece una línea punteada que conecta con una tercera clase "*ThresHold*".
- La clase "*ThresHoldModel*", no tiene como atributo a la clase "*ThresHold*".

Esto se debe a que se ha separado la Lógica (ThresHold) de los Datos (ThresHoldInfo). No tiene sentido tener instanciada a la clase encargada de llevar a cabo la lógica, puesto que solo se deberá instanciar cuando sea necesario. De ahí la línea punteada.

La forma en la cual funciona el sistema para llevar a cabo la operación es la siguiente:

- Cuando se recibe un nuevo dato por el puerto, *setInData()*, se comprueba que este dato es válido.
- Se guarda el dato en la variable "*ImageData"* correspondiente.

- Se invoca al método "*Update()*". Dentro de este método se instancia a la clase encargada de llevar a cabo la lógica de la operación. Se le pasa la imagen de entrada recibida y los datos que se han configurado de la operación (ThresHoldInfo).
- Una vez que se ha instanciado a la clase y se le ha pasado toda la información, se ejecuta el método "*compute()*" de la clase "*ThresHold*". Este método lleva a cabo la operación.
- Finalmente, se le pide a la clase que nos devuelva el resultado con el método "*GetResult()*".
- Una vez que el método "*Update()*" queda fuera de ámbito, la instancia de la clase "*ThresHold*" es eliminada automáticamente.

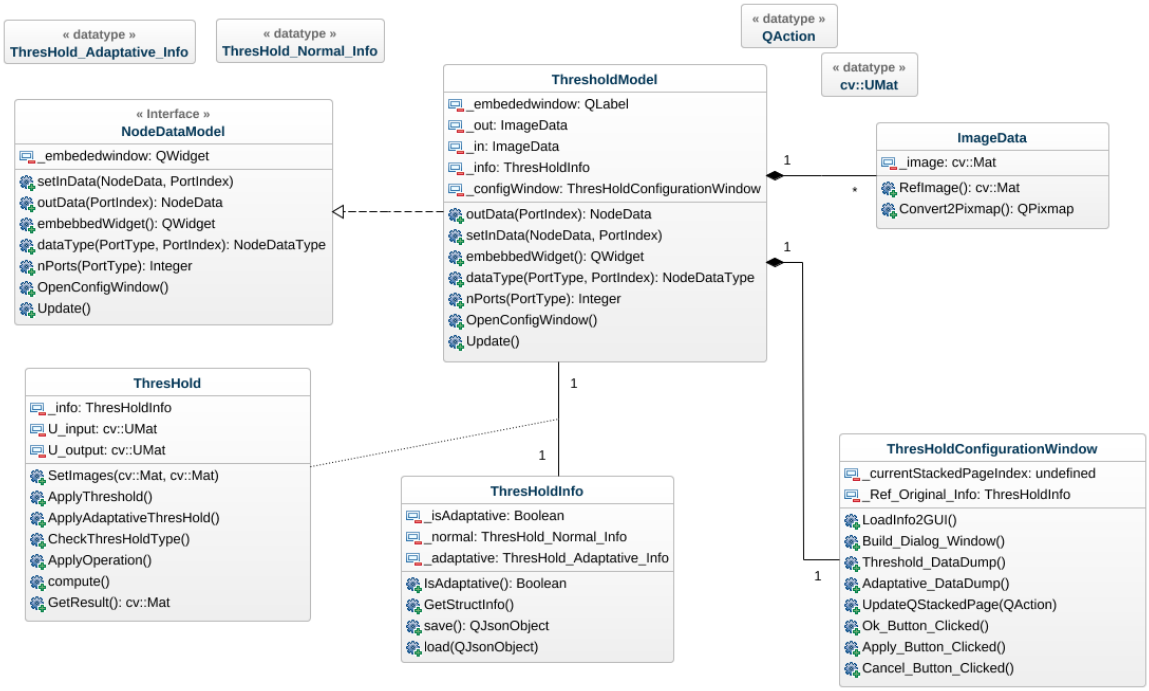


Figura 53. Diagrama de Clases con la Implementación de la Operación de Threshold.

3.3.4 Nodo ImageFill

En este Nodo se lleva a cabo la segunda operación que se realiza sobre la imagen. Se trata de una operación de "*ImageFill*". El principal objetivo de esta operación es rellenar regiones y agujeros presentes en una imagen, a la cual se le ha realizado una operación de "Thresholding" previamente. Las imágenes de la Figura 54 ilustran mejor esta idea.

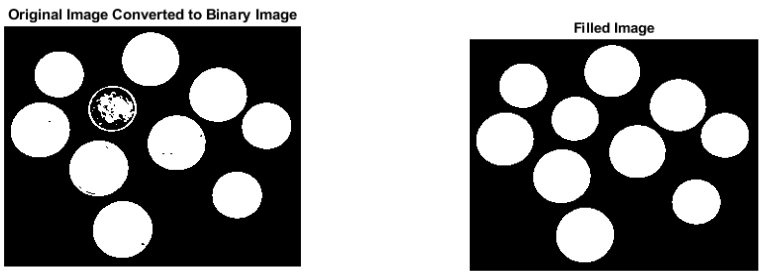


Figura 54. Resultados de realizar la Operación de ImageFill.

Esta operación de "*ImageFill*" se encuentra disponible en Matlab. Sin embargo, no se encuentra implementada en las librerías de OpenCV. Así que se ha tenido que implementar esta funcionalidad. La implementación de este algoritmo es simple:

1. Se parte de una imagen a la cual se le ha realizado una operación de *Thresholding.*
2. Se selecciona una de las esquinas de la imagen y se realiza una operación de Inundación. Esta es la típica operación que tiene el programa "*Paint"* para pintar regiones del mismo color de golpe. Con esta operación se consigue cambiar todo el fondo de la imagen. Si antes el fondo era negro, ahora pasará a blanco y viceversa.

```
int cv::floodFill(
  cv::InputOutputArray image,                  // Input image, 1 or 3 channels
  cv::Point            seed,                   // Start point for flood
  cv::Scalar           newVal,                 // Value for painted pixels
  cv::Rect*            rect,                   // Output bounds painted domain
  cv::Scalar           lowDiff  = cv::Scalar(),// Maximum down color distance
  cv::Scalar           highDiff = cv::Scalar(),// Maximum up color distance
  int                  flags                   // Local/global, and mask-only
);
```

3. La siguiente operación a realizar es una inversión total de la imagen, es decir, se deberá aplicar la operación binaria "*NOT*".
4. Finalmente, se realiza una operación binaria "*OR*" entre la imagen de *Threshold* Original y la imagen obtenida del paso 3. Tras realizar estos pasos todas las regiones internas que presenten huecos serán rellenados.

Figura 55. Pasos del proceso de ImageFill.

En la Figura 56 se muestra el diagrama de clases que implementa esta operación. Se ve que la estructura del diagrama es idéntica al diagrama del nodo que implementa la operación de "*ThresHold"*. Esta similitud no es casualidad. Se ha llegado a la conclusión de que esta forma de implementación resulta ordenada, además de presentar una lógica común para varias operaciones. Debido a que muchas de las funciones que implementa la librería de OpenCV comparte un patrón común, se ha explotado esta peculiaridad creando un diseño o patrón para la implementación de nuevas operaciones. Esto facilita enormemente la implementación de nuevos Nodos.

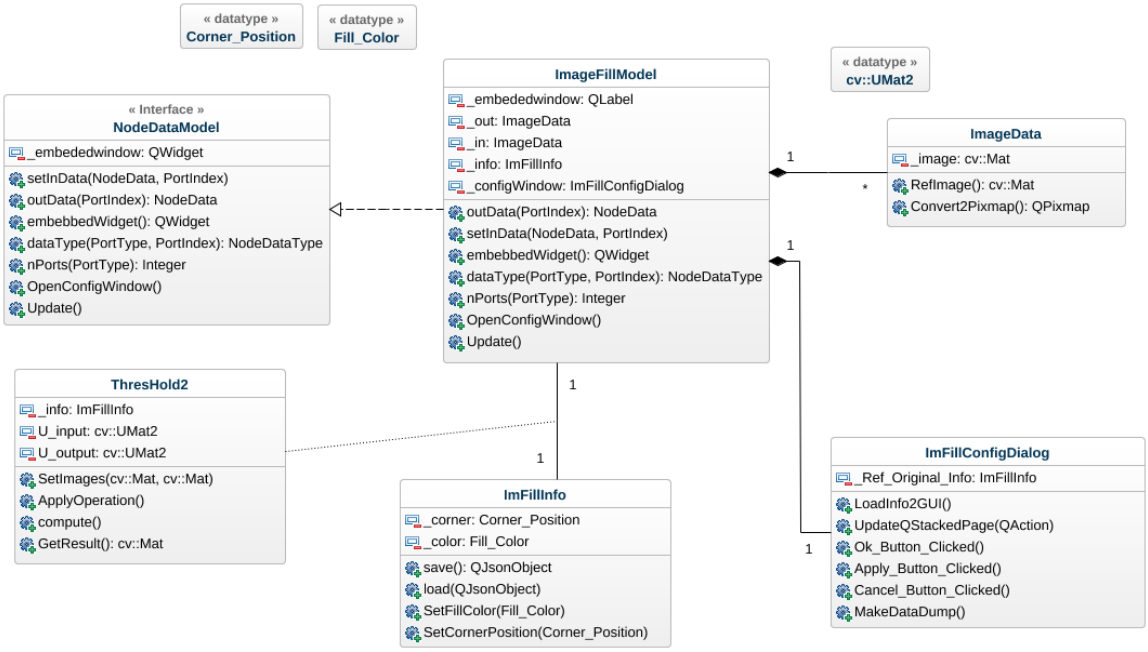


Figura 56. Diagrama de Clases de la operación de ImageFill.

El patrón de diseño empleado para llevar a cabo la implementación de este Nodo es igual al empleado para implementar el Nodo "*ThresHold*". Por lo tanto, todas las explicaciones del funcionamiento de las clases comentadas en aquel Nodo (*ThresHold*) son igualmente válidas en este Nodo.

A partir de ahora, solo se explicarán aquellos Nodos que presenten un Patrón de diseño diferente.

3.3.5 Nodo Morphological

Las operaciones de transformación morfológicas son un conjunto de algoritmos de procesamiento de imágenes que actúan sobre los pixeles de una imagen usando unos *kernels* predefinidos. Por lo tanto, para realizar estas operaciones se necesita la imagen sobre la que se desea realizar dicha operación, generalmente imágenes binarias, y un *kernel* (Matriz de ceros y unos). Aunque se podría emplear cualquier combinación de *kernel* que se quisiera, se suelen emplear unos ya preestablecidos.

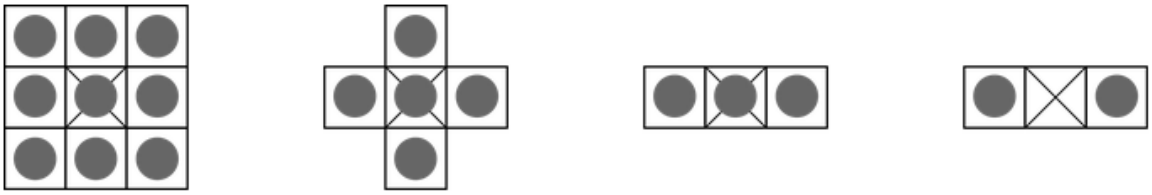


Figura 57. Varias combinaciones de Kernels.

Las dos operaciones fundamentales de las operaciones morfológicas son la Erosión y la Dilatación. Estas operaciones se pueden emplear en situaciones variadas tales como: Remover ruido, aislar elementos individuales, juntar elementos dispares en una imagen, etc. [17]

Erosión

Su principal misión consiste en erosionar los límites del objeto en primer plano. Su funcionamiento se base en desplazar el *kernel* sobre toda la imagen, como en una convolución en 2 Dimensiones, asignándole un valor de uno en caso en que todos los píxeles que coinciden con el kernel sean unos. En caso opuesto, se le asignará un cero. El resultado observado es que las diferentes regiones de la imagen se hacen más finas llegando incluso a desaparecer. [17] [18]

Figura 58. Resultado de una Erosión.

Dilatación

Esta operación se puede ver como la contraria a la Erosión. En este caso, con que dentro del *kernel* caiga al menos un uno, se le asigna a ese píxel el valor de uno. Esta operación tiene como efecto agrandar los bordes de las diferentes regiones en la imagen e incluso fusionar regiones adyacentes. [17] [18]

Figura 59. Resultado de una Dilatación.

El resto de operaciones morfológicas son el resultado de combinaciones de estas dos anteriores. Estas son:

- Apertura → Erosión + Dilatación
- Cierre → Dilatación + Erosión
- Gradiente → Diferencia entre Erosión y Dilatación
- Top Hat → Diferencia entre la imagen de entrada y una Apertura
- Black Hat → Diferencia entre la imagen de entrada y un Cierre

En la Figura 60 se puede ver un resumen con los resultados obtenidos de las diferentes operaciones Morfológicas. Tal y como se ve en las imágenes de la Figura 60, este conjunto de operaciones se puede realizar sobre imágenes en escala de grises. No está limitado exclusivamente a imágenes binarias. [17] [18]

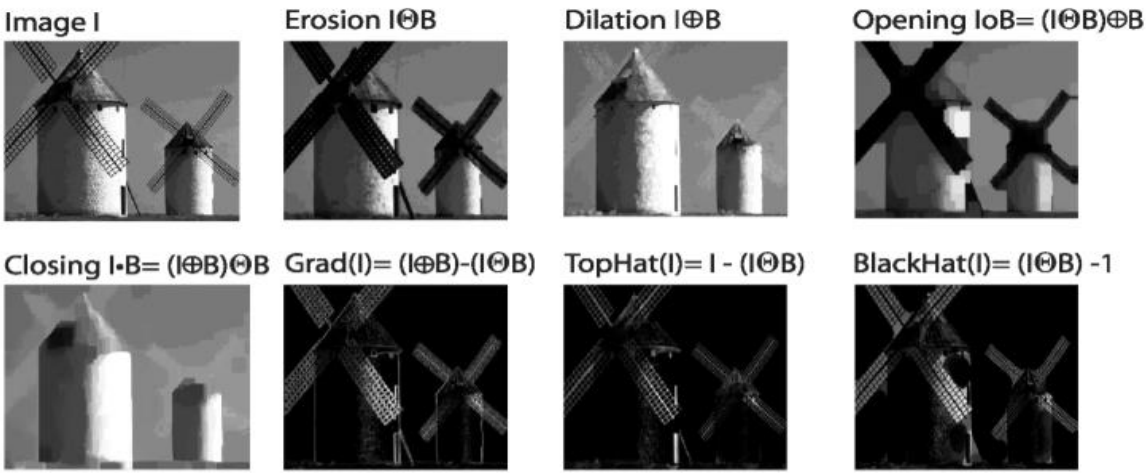


Figura 60. Diferentes tipos de Operaciones Morfológicas.

La forma que nos proporciona la librería de OpenCV para llevar a cabo la implementación de este conjunto de operaciones es la siguiente:

```
void cv::morphologyEx(
  cv::InputArray    src,                      // Input image
  cv::OutputArray   dst,                      // Result image
  int               op,                       // Operator (e.g. cv::MOP_OPEN)
  cv::InputArray    element,                  // Structuring element, cv::Mat()
  cv::Point         anchor      = cv::Point(-1,-1),  // Location of anchor point
  int               iterations  = 1,                 // Number of times to apply
  int               borderType  = cv::BORDER_DEFAULT // Border extrapolation
  const cv::Scalar& borderValue = cv::morphologyDefaultBorderValue()

);
```

En la Figura 61 se ve el Nodo que implementa esta operación y la ventana de configuración de la misma, la cual se desplegará al hacer doble-clic sobre el Nodo. En la ventana se pueden realizar todas las configuraciones proporcionadas por la función de OpenCV. Se observa que presenta una forma muy intuitiva de crear el kernel. Si se quiere cambiar el tamaño del kernel, simplemente se tendrá que pulsar sobre los botones "+" o "-" para añadir o quitar filas o columnas. La casilla azul en la tabla es el denominado punto de ancla. Simplemente, para recolocar el punto de ancla en otra posición, se deberá hacer doble-clic sobre cualquier posición que se desee. Automáticamente esa casilla cambiará su color a azul, siendo ahora el nuevo punto de ancla. En la tabla se puede configurar el valor de cualquier elemento. Sin embargo, solo se aceptarán valores de ceros o uno. Si se colocase cualquier otro valor se cambiaría al valor que tuviese anteriormente.

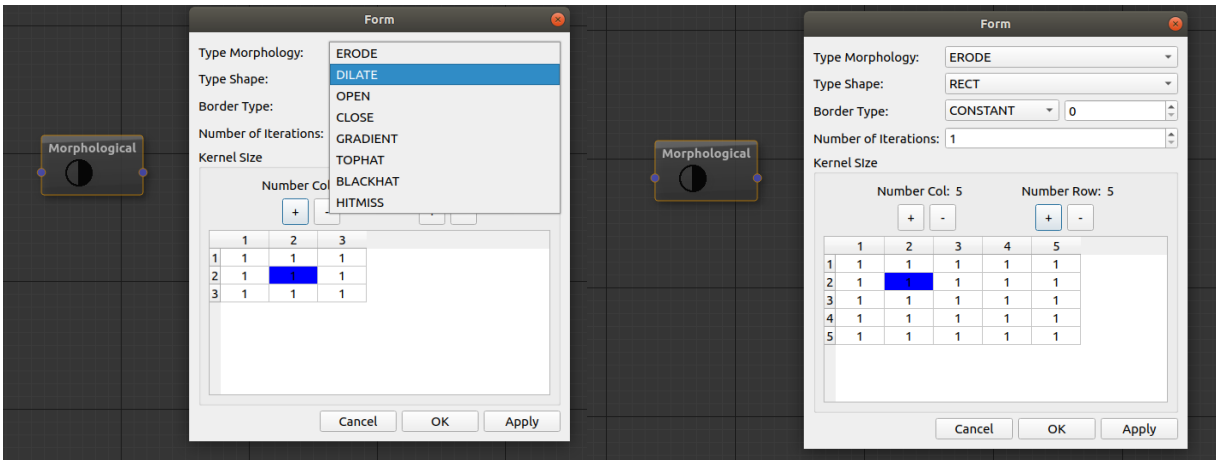


Figura 61. Visualización del Nodo y Ventana de Configuración de la operación Morfológica.

En la Figura 62 se tiene el Diagrama de Clases de este Nodo.

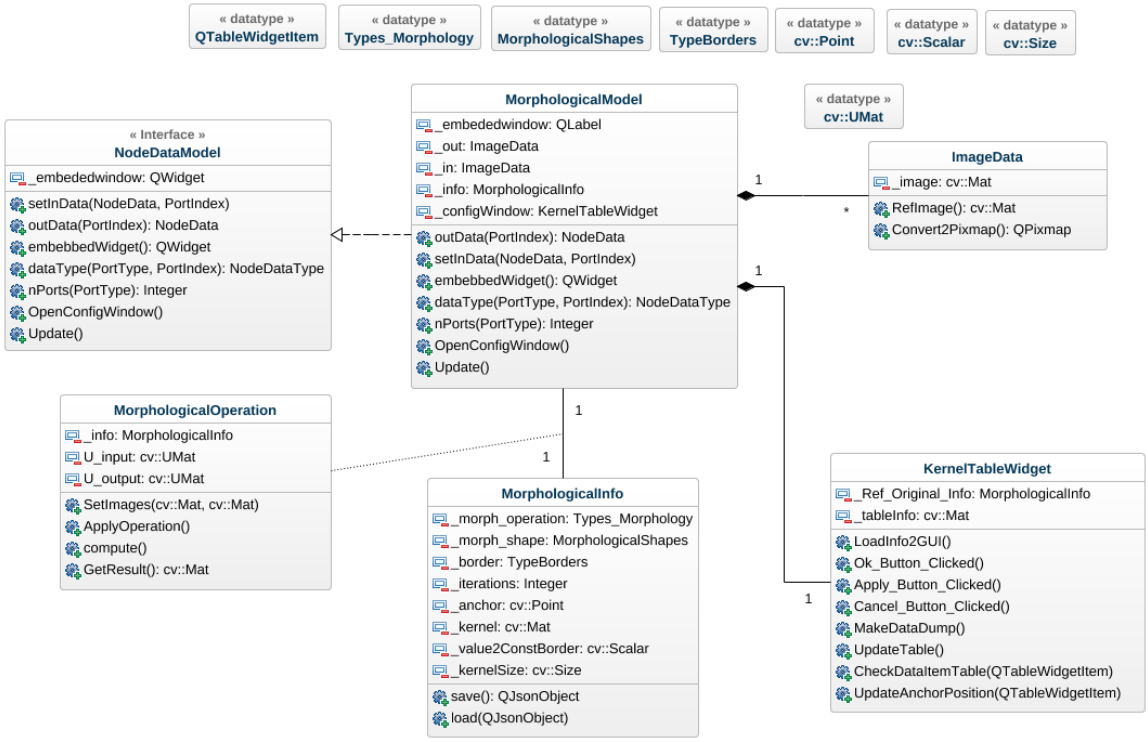


Figura 62. Diagrama de Clases de la Implementación de la Operación de Morfología.

Se puede apreciar que su diagrama de clases presenta el mismo patrón que los anteriores Nodos. Esto era obvio, debido a la forma en la cual se ha decidido diseñar el sistema de operaciones.

Se observa que la mayor variación de un diagrama a otro es la información de configuración de la operación y la ventana de configuración de la operación. La clase encargada de guardar los parámetros de configuración de la operación (*MorphologicalInfo*), presenta un número elevado de atributos. Esto es normal debido a la gran cantidad de parámetros disponibles en la misma. Por motivos de simplicidad, en el Diagrama de clase no se han añadido todos los métodos que presenta. Puesto que, como nos podremos imaginar, se tendrá un par de métodos (setters and getters) por cada atributo presente en la clase.

La clase "*KernelTableWidget*", es la clase empleada para configurar la operación. Es similar al de las anteriores operaciones, salvo que esta posee unos cuantos métodos nuevos:

- UpdateTable() → Este método se ejecuta cada vez que se necesite actualizar la tabla, ya sea porque sus dimensiones cambian, cambiamos el tipo de estructura en el kernel, cambiamos un valor de alguna de sus casillas, etc.
- CheckDataItemTable(QTableWidgetItem) → Este método es el encargado de verificar que el valor introducido en alguna de las casillas es correcto.
- UpdateAnchorPosition(QTableWidgetItem) → Este método es el encargado de actualizar la posición del punto de ancla(casilla en azul).

Todo el procedimiento de procesamiento y funcionamiento de las clases es idéntico a las anteriores operaciones (ThresHold, ImageFill), por lo que no es necesario volver a explicar el funcionamiento del mismo.

3.3.6 Nodo ConnectedComponents

Se trata de una técnica de teoría de grafos en la cual unos subconjuntos de componentes conectadas son etiquetados de manera única. En el procesamiento de imágenes, esta técnica se suele usar después de haber realizado una segmentación sobre la imagen. Gracias a esta técnica, se pueden aislar las diferentes regiones que presenta una imagen binaria, permitiendo así poder trabajar con las regiones una a una. [19] [13]

Figura 63. Visualización del Resultado de la Operación de Connected Components.

La librería de Opencv proporciona las siguientes funciones para llevar a cabo esta operación:

```
int  cv::connectedComponents (
  cv::InputArrayn image,                // input 8-bit single-channel (binary)
  cv::OutputArray labels,               // output label map
  int             connectivity = 8,     // 4- or 8-connected components
  int             ltype        = CV_32S // Output label type (CV_32S or CV_16U)
);

int  cv::connectedComponentsWithStats (
  cv::InputArrayn image,                // input 8-bit single-channel (binary)
  cv::OutputArray labels,               // output label map
  cv::OutputArray stats,                // Nx5 matrix (CV_32S) of statistics:
                                        // [x0, y0, width0, height0, area0;
                                        //  ... ; x(N-1), y(N-1), width(N-1),
                                        // height(N-1), area(N-1)]
  cv::OutputArray centroids,            // Nx2 CV_64F matrix of centroids:
                                        // [ cx0, cy0; ... ; cx(N-1), cy(N-1)]
  int             connectivity = 8,     // 4- or 8-connected components
  int             ltype        = CV_32S // Output label type (CV_32S or CV_16U)
);
```

Figura 64. Funciones proporcionadas por Opencv para realizar la Implementación de ConnectedComponents.

Aunque se aprecian varios parámetros, en esta operación solo se le puede configurar un parámetro, "*connectivity*". En el caso de un plano, este valor hace referencia a los vecinos cercanos que tiene un pixel. En el caso de contar solo los laterales tendremos 4. En el caso de contar las esquinas se pasará a 8.

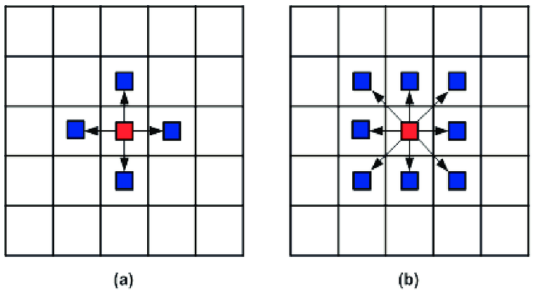


Figura 65. Tipos de Conectividad.

OpenCV proporciona dos funciones para realizar la misma operación. La diferencia entre una y otra, es el número de parámetros que se obtienen en el resultado final. En la primera opción, únicamente se obtendrá una matriz con las etiquetas de las diferentes regiones encontradas, correspondiéndose el cero con el fondo. La segunda opción, a parte de la matriz con las etiquetas, devuelve también información relativa a cada región que el algoritmo ha aislado. Tales como: sus centroides, sus áreas (en pixeles), etc. Para el proyecto que nos ocupa, se ha optado por utilizar exclusivamente la segunda opción.

En la Figura 66 se tiene el Diagrama de Clases que implementa el Nodo:

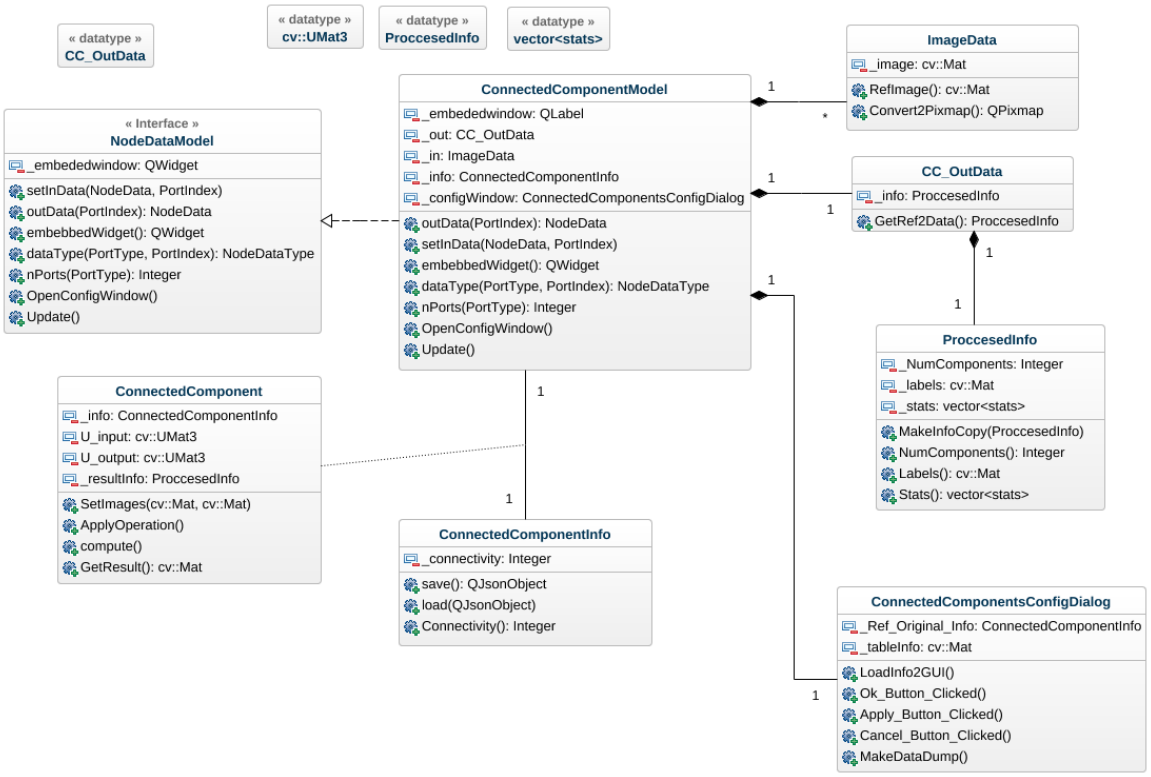


Figura 66. Diagrama de Clases de la Operación ConnectedComponents.

Se ve que el patrón que utiliza este Nodo sigue siendo el mismo que en los anteriores. Sin embargo, hay ciertas cosas que han cambiado un poco:

Clase CC_OutData

Ahora como salida, se tiene otro tipo de clase que no es del tipo *ImageData*. Sino que ahora, se tiene a la clase "*CC_OutData*". Las letras *CC* son la abreviatura para Connected Components. Esta clase actúa como un contenedor para la clase que contiene la verdadera información, *ProccesedInfo*. Esta a su vez tiene tres atributos:

1. _NumComponents → Número de regiones encontradas.
2. _labels → Es la Matrix que contiene la información de las etiquetas.
3. _stats → Es un vector que contiene una estructura con toda la información relevante para cada región, de ahí que sea un vector. La estructura contiene en su interior:
    a. x → Posición x0 de la región.
    b. y → Posición y0 de la región.
    c. width → Ancho de la región.
    d. heigth → Alto de la región
    e. area → Número de píxeles de la región.
    f. centroids
        i. cx → Posición en "x" del centro de la región.
        ii. cy → Posición en "y" del centro de la región.

Toda la información relevante a la operación de "*connected components*" esta agrupada. Sin embargo, al emplear la función proporcionada por OpenCV la información se encuentra desagrupada. Entonces ¿Cuál es el motivo

para agruparla? ¿No sería más lógico tener tres puertos de salida en el Nodo y cada información individualmente? Ese fue su planteamiento inicial, pero se tuvo que descartar y optar por agrupar todo por razones de sincronismo.

Hay que recordar que el sistema funciona por eventos, y por lo tanto si creamos tres puertos de salida la información no se envía de golpe, sino que debería ser enviada una a una por sus respectivos puertos. Supongamos que nos surge la necesidad de crear otro Nodo, el cual debe obtener toda la información de golpe. Por lo tanto, a este bloque se le deberían de añadir tres puertos de entrada y conectar las respectivas entradas con las salidas. Así que tendríamos que esperar a recibir los tres datos y luego procesarlos. Esto ya va sonando mal. Pero ¿Cómo nos aseguramos que al recibir los 3 datos, estos pertenecen al mismo frame que se procesó? ¿Qué pasaría si un dato perteneciera a otro frame? Para evitar estos problemas, mandamos todo agrupado y así nos aseguramos que tenemos toda la información del frame sincronizada.

A excepción de lo ya comentado, el resto de clases presentan el mismo patrón de comportamiento que las anteriores, teniendo claramente sus diferencias en la información que cada una guarda en su interior.

3.3.7 Nodo CC_FilterByArea

Este Nodo se encarga de realizar un filtrado por área (píxeles) de las diferentes regiones. Las iniciales "*CC*" son la abreviatura de *ConnectedComponents*. El nodo debe usarse conjuntamente con el Nodo de "*Connected Components"*. La idea es quedarse solo con aquellas regiones cuyas áreas se encuentren dentro del rango configurado.

La librería de OpenCV no posee una función específica para realizar esta acción. Por lo tanto, esta operación se ha desarrollado por completo. La operación ha resultado ser fácil de programar, gracias a que se tiene toda la información de "*connected componentes"* agrupada. Lo único que se debe hacer para implementar esta funcionalidad es:

1. Iterar por el vector de "*Stats*" quitando aquellos elementos cuya área se salga fuera de los límites establecidos.
2. A medida que se itera por el vector, se deberá llevar la cuenta de la posición actual. Si la componente es eliminada, se debe guardar en otro vector esta posición.
3. Actualizar la matriz de las etiquetas de las regiones. Al iterar por el vector de "Stats", se ha ido guardando el número de las diferentes componentes que fueron eliminadas. Por lo que ahora, hay que usar esa información para eliminar de la matriz de etiquetas aquellas que coincidan con estas. Esto se hace simplemente sustituyendo su valor por cero, es decir, convertir dicha región a fondo.

En las siguientes vemos al Nodo, además de su ventana de configuración de la operación.

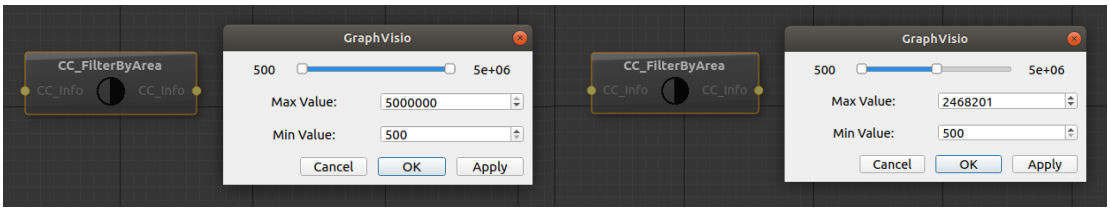


Figura 67. Visualización del Nodo y su Ventana de Configuración.

La ventana de configuración de la operación es simple. En ella se puede configurar el área máxima y mínima que deben cumplir las regiones. La configuración de esta área se realiza a través del "*Doble Slider*" o empleando los *SpinBoxes*. En el *DoubleSlider*, los valores fijos en los extremos representan los valores límite de configuración. El valor mínimo límite de 500 se asigna siempre automáticamente. El valor límite máximo se asigna en función de la imagen, asignándole siempre el número total de píxeles presentes en la imagen.

En la Figura 68 muestra el Diagrama de Clases del Nodo.

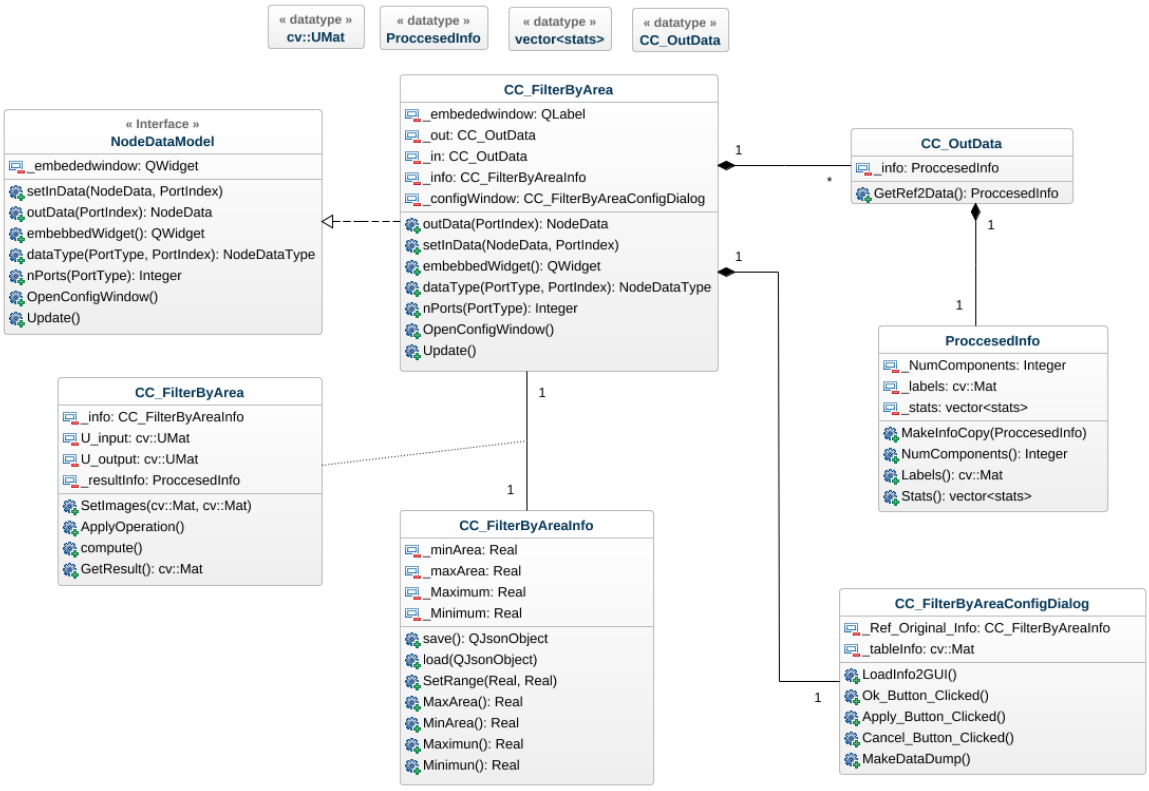


Figura 68. Diagrama de Clases de la operación de CC_FilterByArea.

Nuevamente, la operación sigue el mismo patrón de diseño que se ha estado usando hasta ahora. Por lo tanto, no resulta necesario explicar su funcionamiento. Lo único que se debe aclarar es la clase "*CC_FilterByAreaInfo*". Esta clase guarda la información de configuración de la operación. En esta clase, se tienen los atributos: _minArea, _maxArea, _Maximum y _Minimum. Los dos primero hacen referencia a los valores entre los cuales deben oscilar las regiones de interés. Mientras que los otros, sirven para configurar los límites físicos entre los que pueden oscilar las regiones. Es decir, estos valores representan los valores extremos del *Slider*. Los otros, representan los valores o posiciones que tienen actualmente configurados los "*Slider*".

3.3.8 Nodo SplitStructInfo

Este Nodo tiene como objetivo separar toda la información asociada a la operación de "*Connected Components*" en sus elementos más básicos: Stats, Centroides y Matriz de Etiquetas.

En la Figura 69 se puede apreciar como el Nodo presenta un puerto de entrada y tres puertos de salida.

- CC_Info → Información de las Stats
- Markers2Draw → Centroides o Puntos de las diferentes regiones.
- IMAGE → Matriz con las diferentes etiquetas.

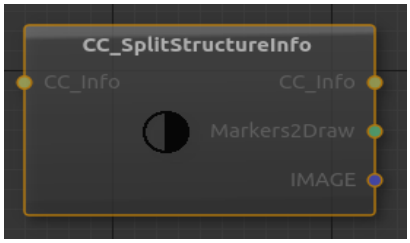


Figura 69. Visualización del Nodo CC_SplitStructInfo.

El Diagrama de clases del Nodo se muestra en la Figura 70.

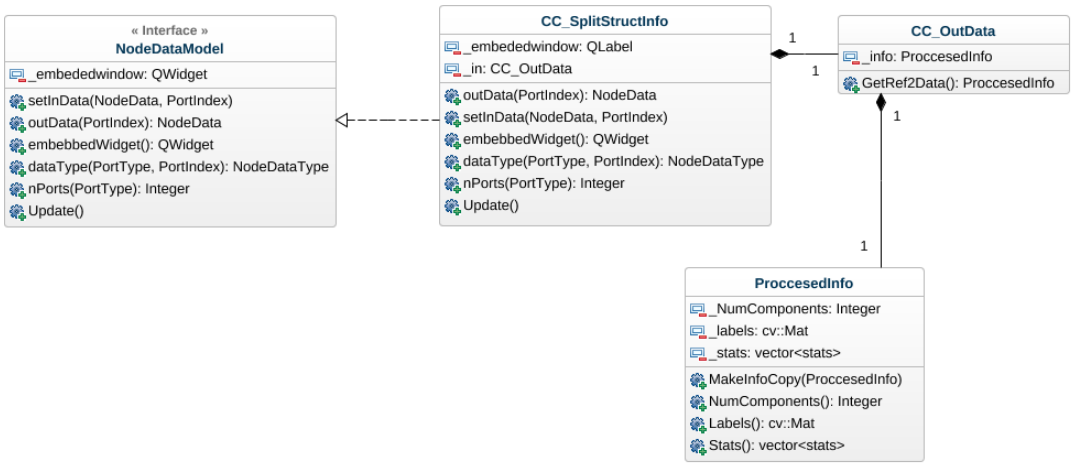


Figura 70. Diagrama de Clases del Nodo CC_SplitSturctInfo.

Para este, caso no se tiene el mismo patrón que se tenía en los anteriores Nodos. Esto se debe a la simplicidad del mismo. Debido a que este Nodo simplemente se encarga de separar la información que recibe en sus componentes más básicas. No posee clases para configurar una operación, puesto que no realiza una operación como tal. En la clase "*CC_SplitStructInfo*", solo se tiene como atributo la variable correspondiente a la entrada y no se tiene las otras tres variables que se corresponderían con la información relativa a los puertos de salida. Esto se debe a que no es necesario para el bloque guardar en variables esa información. Simplemente, cuando sea necesario se extrae de la variable de entrada. En el Diagrama de la Figura 71 se explica el funcionamiento del sistema.

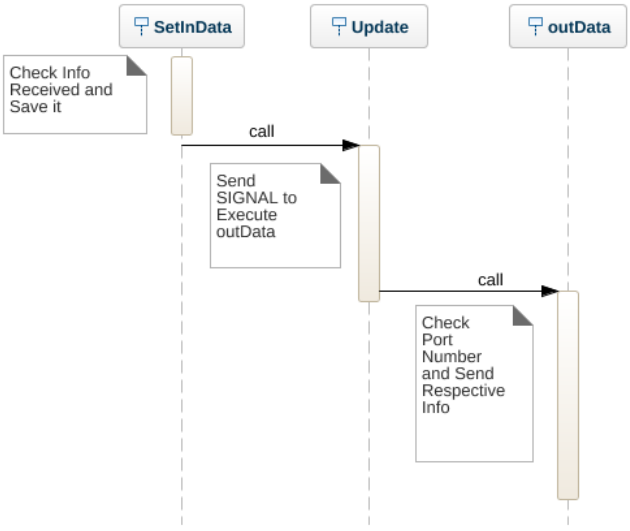


Figura 71. Esquema del Funcionamiento del Nodo CC_SplitStructInfo.

Cuando se ejecuta el método "*setInData()*", lo primero que se hace es comprobar que el dato recibido es válido. Si resulta serlo se guarda en el atributo "*_in*". Acto seguido, este método llama al método "*Update()"*. El cuál es el encargado de lanzar la señal para que se ejecute el método "*outData()*". Se envía una señal por cada puerto (un total de tres señales). El método "*outData()*" es el encargado de comprobar en qué puerto se debe enviar la señal y finalmente extraer la información y enviarla por el puerto correspondiente.

3.3.9 Nodo Matrix2Label

La Matriz de etiquetas, la cual contiene las diferentes regiones, no es propiamente una imagen. Sino que más bien es una Matriz. La librería de OpenCV usa la misma clase para guardar matrices que imágenes, diferenciándolas únicamente por el tipo de dato. Por lo tanto, aunque se utiliza la misma clase, no tienen el mismo formato y se deberán hacer ciertos cambios sobre la matriz. Estos cambios se realizan en este Nodo.

La matriz de etiquetas es, como su nombre indica, una matriz en la cual los ceros representan el fondo, el uno representa una región en la imagen, el dos otra región, así sucesivamente con todas las regiones. De forma resumida, el trabajo de este Nodo es recorrer la matriz y cambiar todos los valores mayores o iguales a uno en valor de 255. Puesto que en una imagen en escala de grises los píxeles solo pueden tomar valores entre 0 y 255. Con esta operación se convierte la Matriz a una imagen Binaria, donde el fondo se corresponde con el valor de cero y las diferentes regiones se corresponden con el valor de 255.

El diagrama de clases de este Nodo es el siguiente:

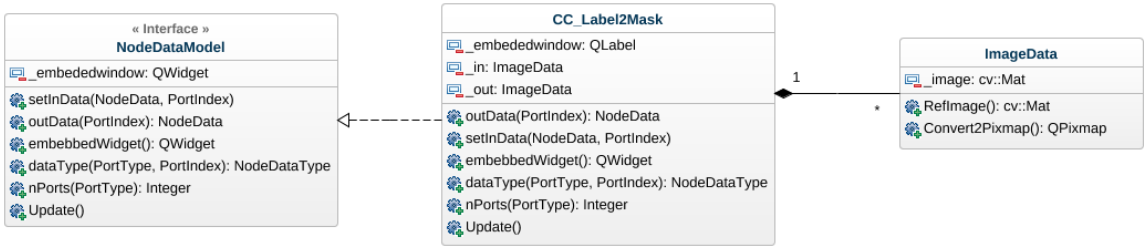


Figura 72. Diagrama de Clases de la operación Label2Mark.

Nuevamente, debido a la simplicidad de este Nodo, no ha sido necesario emplear el patrón utilizado en anteriores Nodos. En este caso, simplemente cuando se ejecuta el método de *Update()* se lleva a cabo el proceso comentado arriba utilizando una función diseñada para tal fin.

3.3.10 Nodo DrawPoints

El Nodo presenta dos puertos de entrada, un puerto por donde se reciben los puntos a dibujar (Posición). Por el otro puerto, se recibe la imagen sobre la que se desea dibujar estos puntos. El puerto de salida devuelve una imagen con los puntos dibujados sobre la imagen a la entrada. En las sucesivas figuras se muestra el aspecto del Nodo en la aplicación y su diagrama de Clases.

En la clase "*DrawMarks2Model*", tenemos tres atributos. Cada uno se corresponde con los tres datos que maneja este Nodo:

- _in → Imagen sobre la que se desea pintar los puntos
- _in_Points → Vector que contiene la posición de los diferentes puntos a dibujar.
- _out → Imagen con los puntos dibujados.

Nuevamente, debido a la simplicidad de este Nodo, no ha sido necesario emplear el patrón que se ha estado utilizando en anteriores Nodos. En este caso, cuando se ejecuta el método de *Update()* se lleva a cabo el proceso comentado anteriormente, empleando para ello la función "*DrawMarks2Image(cv::Mat&, cv::Mat&, vector<MarketPoint_Info>&)*".

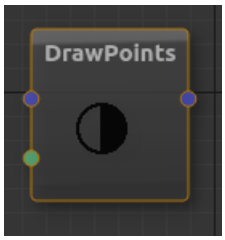


Figura 73. Visualización del Nodo DrawPoints.

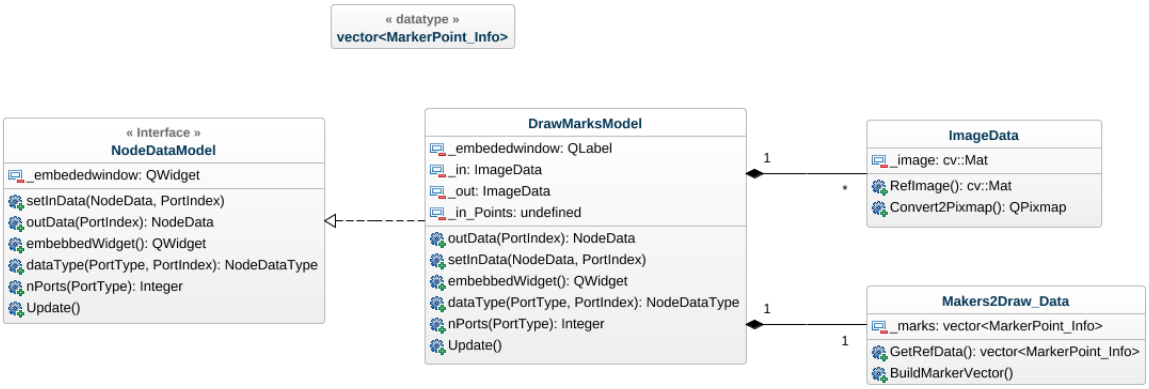


Figura 74. Diagrama de Clases de la Implementacion del Nodo DrawPoints.

4 SOFTWARE EN FUNCIONAMIENTO

La aplicación ha sido desarrollada bajo un entorno Linux, debido a la facilidad que presenta este sistema para la instalación de librerías externas. Gracias a la utilización de una librería multiplataforma (Qt), el programa se podría portar a otros entornos tales como Windows o macOS sin apenas problemas.

En este apartado, se va a proceder a enseñar el funcionamiento del programa, de cara al usuario. Explicando cómo manejar el programa paso a paso. En primer lugar, se recordará el algoritmo objetivo a realizar y se ira mostrando poco a poco como se va desarrollando el algoritmo, visualizando los resultados en todo momento.

Al iniciar la aplicación, lo primero que se vera es lo siguiente:

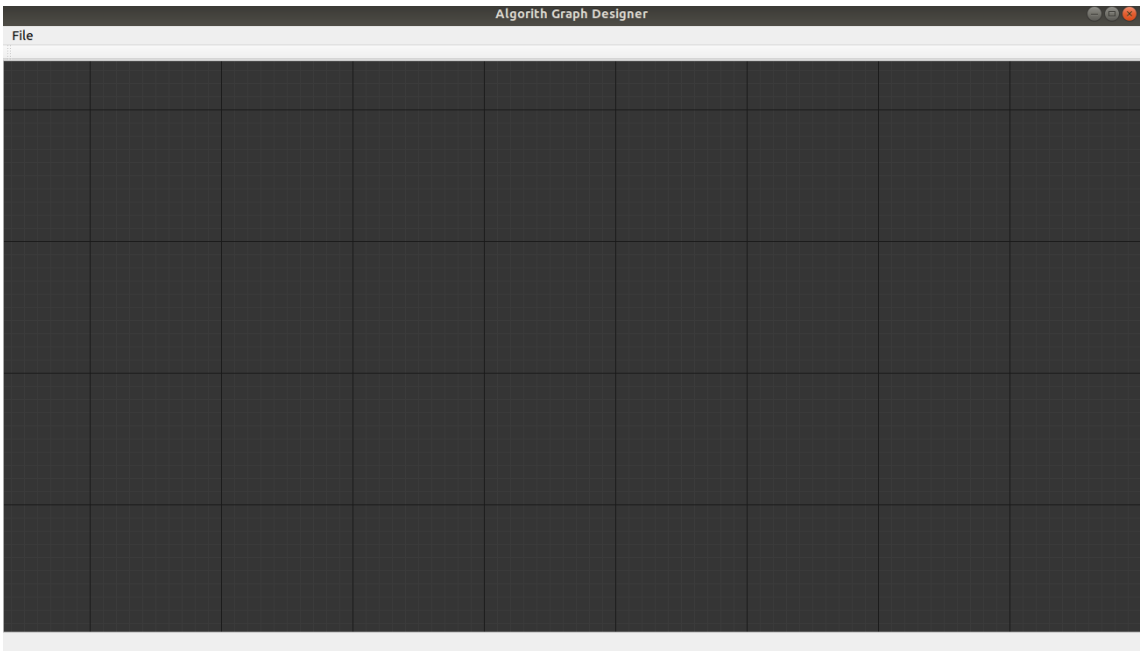


Figura 75. Venta Inicial de la Aplicación.

En la Figura 75, se puede apreciar que se dispone de una cuadrícula en Negro. Esta es nuestra escena, donde se irán colocando los diferentes Nodos. Además, se puede ver arriba a la izquierda un menú con la opción File. Si se pulsa sobre este, se desplegará un conjunto de opciones {New File, Open File, Save File, Clear Scene}. Tal y como los nombres indican, la opción *Open File,* permite cargar un archivo, con el diseño de un algoritmo ya previamente guardado. De forma parecida, la opción *Save File*, permite guardar el conjunto de Nodos y sus respectivas configuraciones, para después cargarlo. La opción *Clear Scene*, permite limpiar toda la escena, es decir, se encarga de eliminar todos los Nodos y Conexiones presentes en la escena.

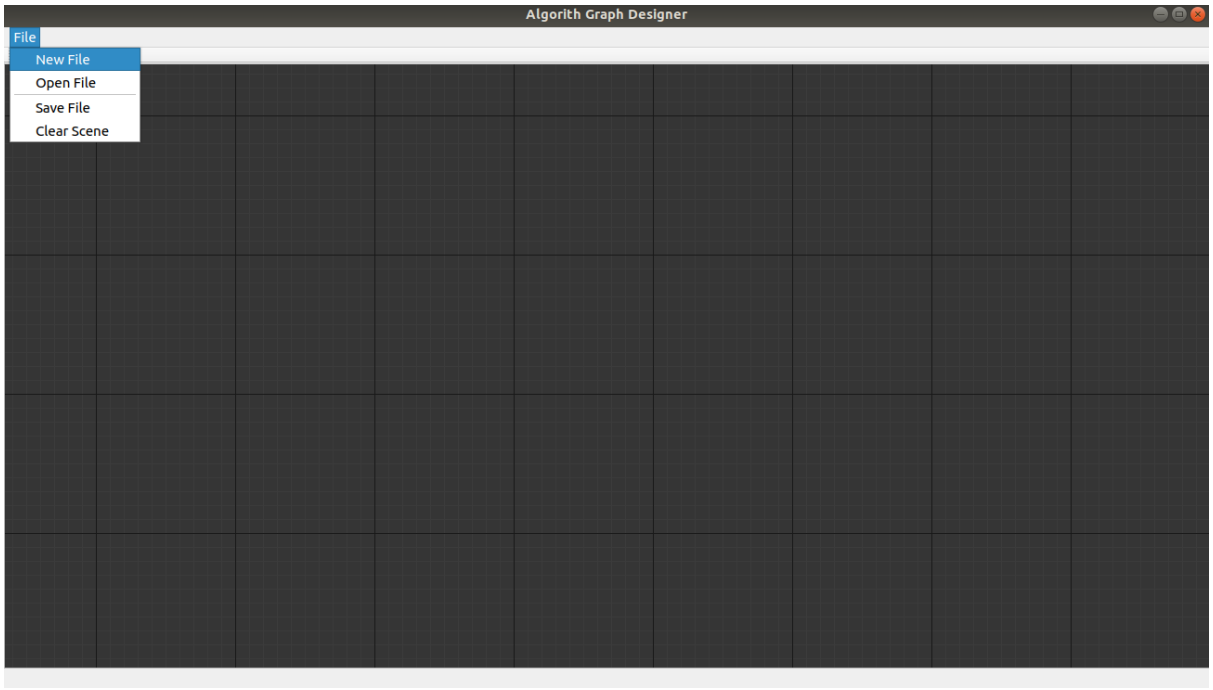


Figura 76. Ventana Inicial de la Aplicación y Opciones disponibles en File.

Actualmente, la manera que se dispone para añadir bloques a la escena, es pulsando el botón derecho del ratón sobre la misma escena. En el Menú Contextual que se desplegará, se podrá elegir alguno de los bloques que se haya implementado. Al hacer clic sobre alguna de las opciones, aparecerá automáticamente el Nodo en la escena en la misma posición donde se haya hecho el doble clic.

El el Diagrama de la Figura 4-3, se puede apreciar el proceso que sigue la aplicación para crear un nuevo Nodo.

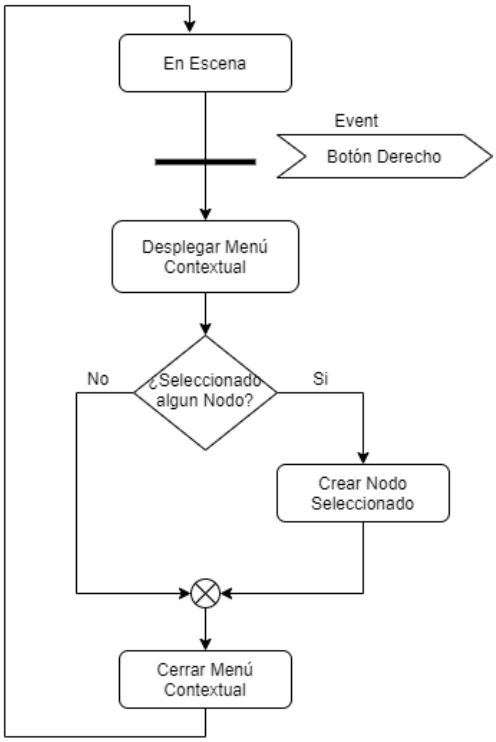


Figura 77. – Diagrama del Proceso para crear un Nuevo Nodo.

.

Cabe mencionar, que el menú contextual, tiene una estructura de árbol. Este agrupa los bloques por categorías.

Pudiendo minimizar o maximizar estas categorías. Además, se dispone de un buscador en la parte superior del menú contextual, el cual ayudará a filtrar los Nodo por su nombre.

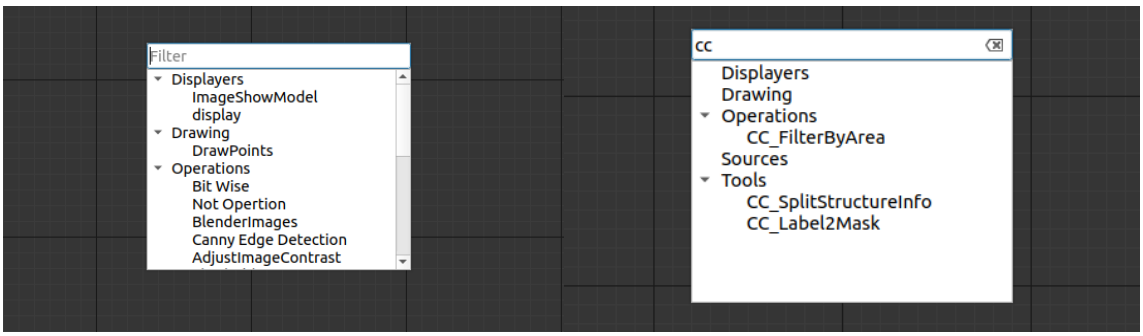


Figura 78. Menu Contextual mostrando los diferentes Nodos Disponibles.

Si recordamos del apartado 3.1, el ejemplo que se iba a implementar era la localización de un conjunto de monedas en una imagen con presencia de mucho ruido y sombras.

Figura 79. Problema de Partida. Localización de las Monedas.

Además, en esa misma sección se presentó el algoritmo que se iba a implementar para resolver el problema. Recordando los nodos necesarios para implementar:

- Read Source (Video or Image)
- Threshold Operation
- Image Fill Operation
- Morphological Operation
- Connected Components
- Filter by Area Connected Components
- Convert MatLabel to Mask
- Draw Points in the Centroids Position

A continuación, se irán mostrando imágenes con la configuración de los Nodos para llevar a cabo el desarrollo del algoritmo.

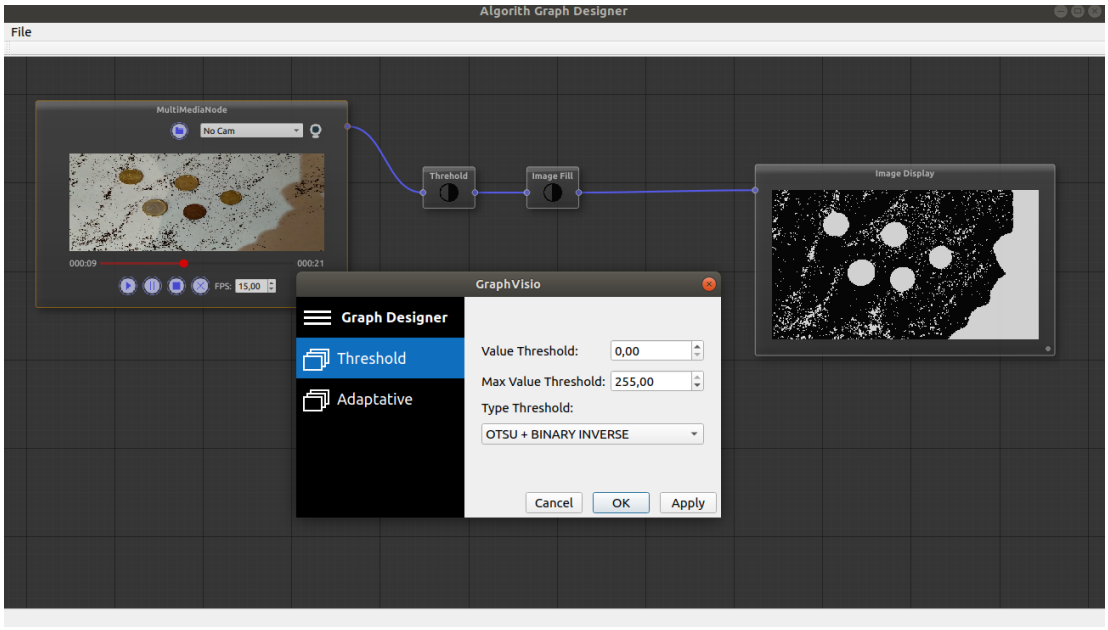
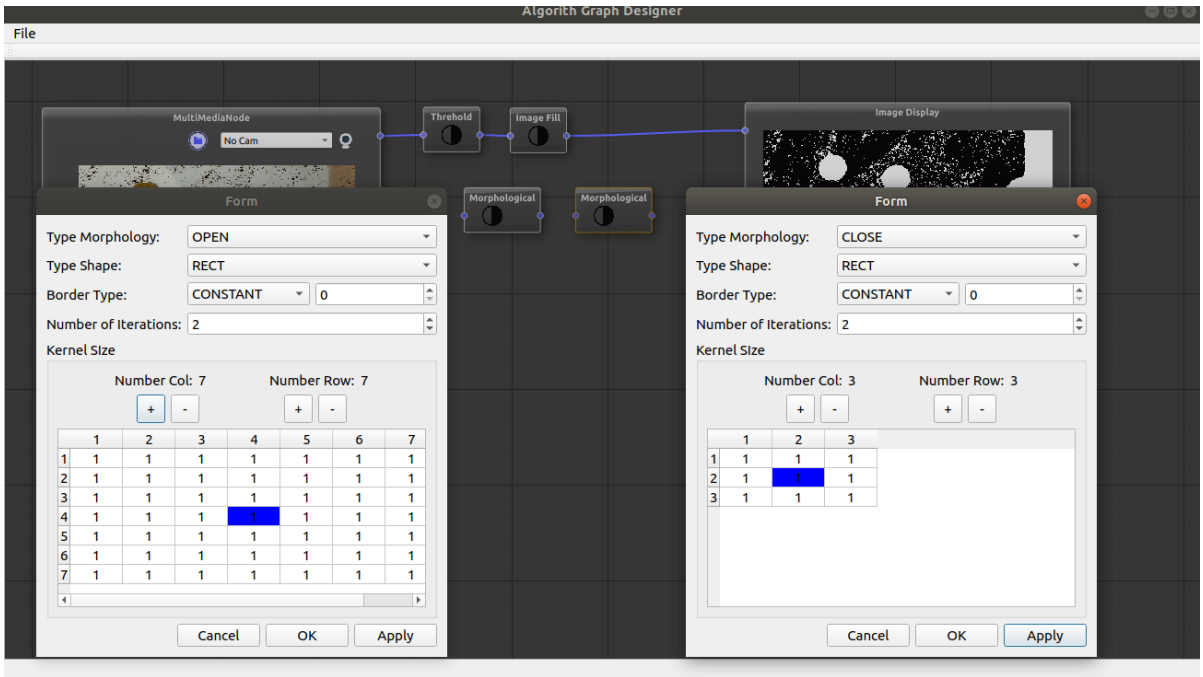


Figura 80. Proceso de Desarrollo de la Solución empleando el Sistema de Nodos.

En la Figura 80, se puede ver que se ha creado el Node encargado de reproducir y enviar los Frames, Nodo de la izquierda. A la derecha, se ve un Nodo para visualizar el resultado. Los dos Nodos entre la imagen fuente y el resultado son los nodos de *Threshold* y *ImageFill*. La ventana del centro, es la ventana de configuración que se ha usado para el nodo de *Threshold*. Se ve que se ha empleado una operación de *Threshold* y se ha usado el método de "OTSU + BINARY INVERSE".



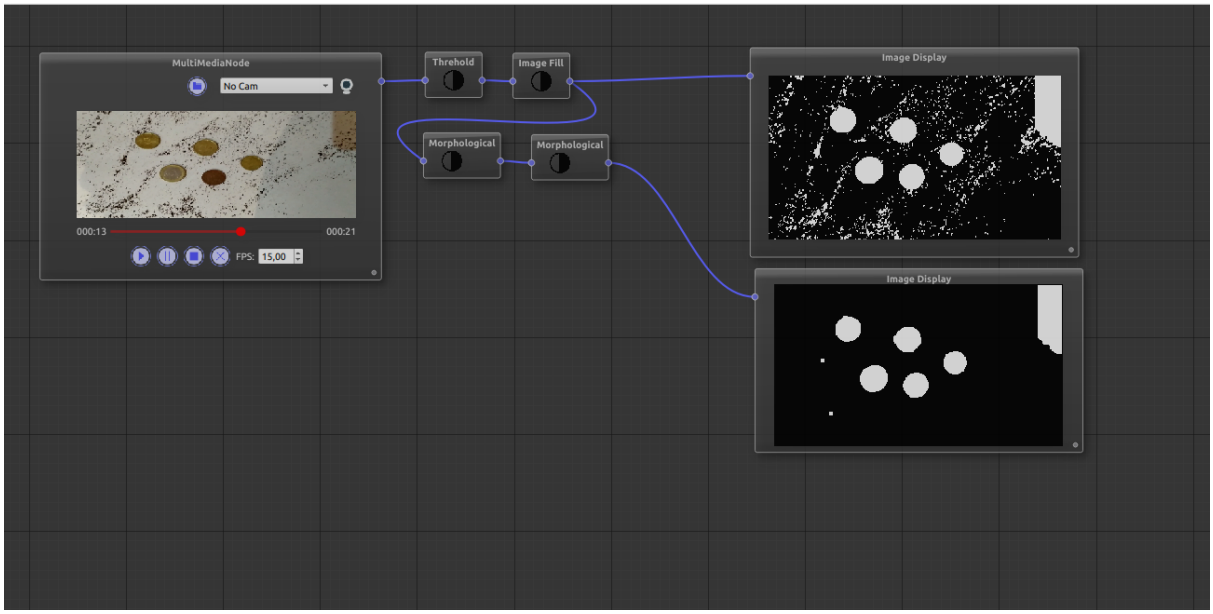


Figura 81. Proceso de Desarrollo de la Solución empleando el Sistema de Nodos.

En la Figura 81, se ve el conjunto de operaciónes empleadas. En este caso, se han encadenado dos operaciones Morfológicas, ambas con configuraciones completamente distintas.

En la Figura 81, se ven las ventanas correspondientes a la configuración de ambas operaciones. Siendo la ventana más a la derecha, la configuración correspondiente al Nodo de la operación morfológica situado más a la derecha. Para este caso, se ha empleado un "*Close"* con un kernel de 3x3 con forma rectangular y el número de iteraciones igual a dos.

Para la otra operación morfológica, ventana y Nodo más a la izquierda, la configuración de la operación ha sido un "*Open"* con un kernel de 7x7 con forma rectangular y el número de repeticiones igual a dos.

Además, se ha añadido otro visualizador para verificar las diferencias después de aplicar estas dos operaciones morfológicas. Se puede observar una disminución del ruido muy acentuada. Sin embargo, se ha colado una gran sombra a la derecha. En las siguientes operaciones se tratará de corregir estos defectos y dejar únicamente las monedas.

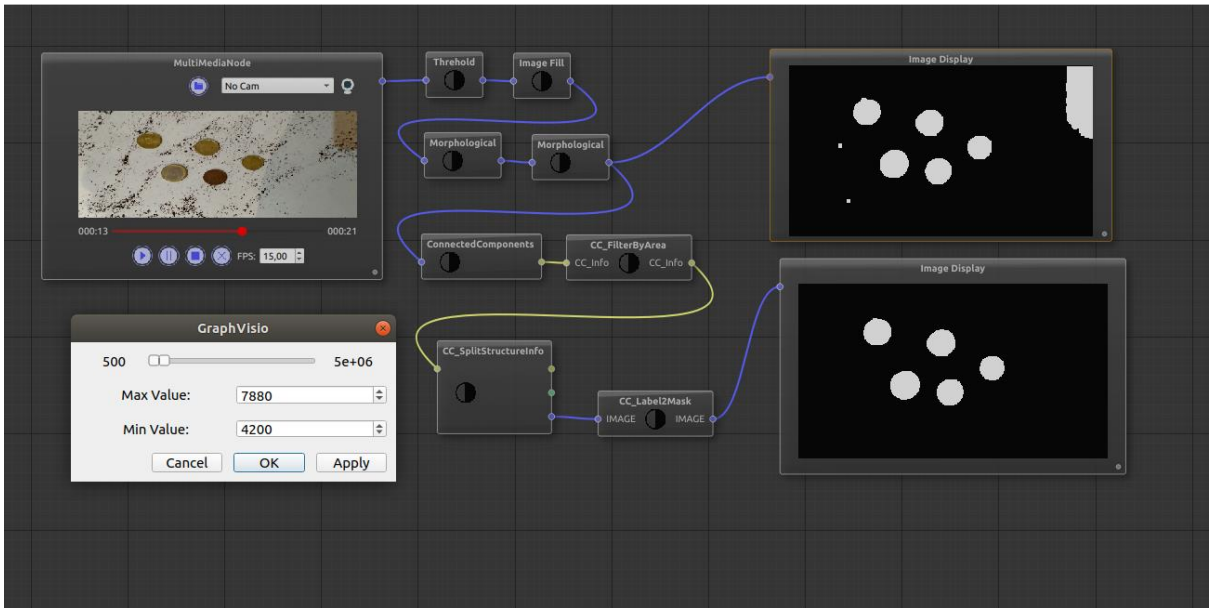


Figura 82. Proceso de Desarrollo de la Solución empleando el Sistema de Nodos.

Las siguientes operaciones aplicadas, son una operación de "*Connected Components*". Gracias a la cual se

identifican las diferentes regiones en la imagen. Una vez identificadas las diferentes regiones presentes en la imagen, se procede a realizar un filtrado por área.

La ventana de configuración del filtrado por área, se puede apreciar abajo a la izquierda en la Figura 82. Esta ventana permite seleccionar el rango de tamaño, el número de píxeles, con el que se desea quedarse. Toda región que se encuentre fuera del rango especificado, pasará a convertirse en fondo automáticamente. Finalmente, se obtienen los resultados esperados: aislar las regiones de las diferentes monedas del ruido y de las sombras.

Los dos bloques de abajo del todo, son bloques necesarios para llevar a cabo las visualizaciones finales. El bloque de la izquierda, "*CC_SplitInfoStruct*", se encarga de separar la información procedente de "*Connected components*" en sus elementos más básicos. Esto se puede apreciar en el número de puertos a la entrada y a la salida, teniendo un puerto a la entrada y tres a la salida. Los tres puertos a la salida son: información básica de *Connected Components* (tamaño en píxeles, ancho, alto, etc), los centroides de las diferentes regiones y una Matriz de números, etiquetando las distintas regiones. A esta matriz se la conoce como matriz de labels.

Esta manera de proceder con la información, obedece a criterios de sincronismo y reducción de conflictos. Es mucho más fácil y lógico agrupar toda esa información y enviarla por su puerto en lugar de separarla. Ya que, si se separase antes, se crearían problemas de sincronismo en los datos. Además de posibles conflictos.

Esto se debe a que todo el sistema funciona por eventos, es decir, cuando por cualquiera de los puertos de entrada de algún nodo llega nueva información, ese nodo la procesa y envía su resultado al puerto de salida. Lo mismo sucederá con aquel nodo que reciba esta información. Provocando así, la propagación de la información.

Supongamos un caso hipotético en el cual separamos la información a la salida del nodo de "Connected Components". El nodo encargado de realizar el filtrado por área, debería poseer ahora tres puertos de entrada. Supongamos ahora que con esos tres puertos le llega información por alguno de ellos. El sistema tendría que procesarla. Pero ¿cómo la procesa? Ha llegado solo un dato de los tres necesarios. Podríamos optar por posponer el procesamiento hasta que lleguen los otros dos datos. Pero ahora, teniendo los otros dos datos ¿Cómo estamos seguros que los 3 datos provienen del mismo frame procesado y no de frames distintos? Vemos que, al querer separar la información, complicamos tontamente el problema. Por lo tanto, la solución más simple es agruparlos y crear un bloque especial cuyo objetivo sea el separar esta información.

Finalmente mostramos la configuración empleada de todos los nodos.

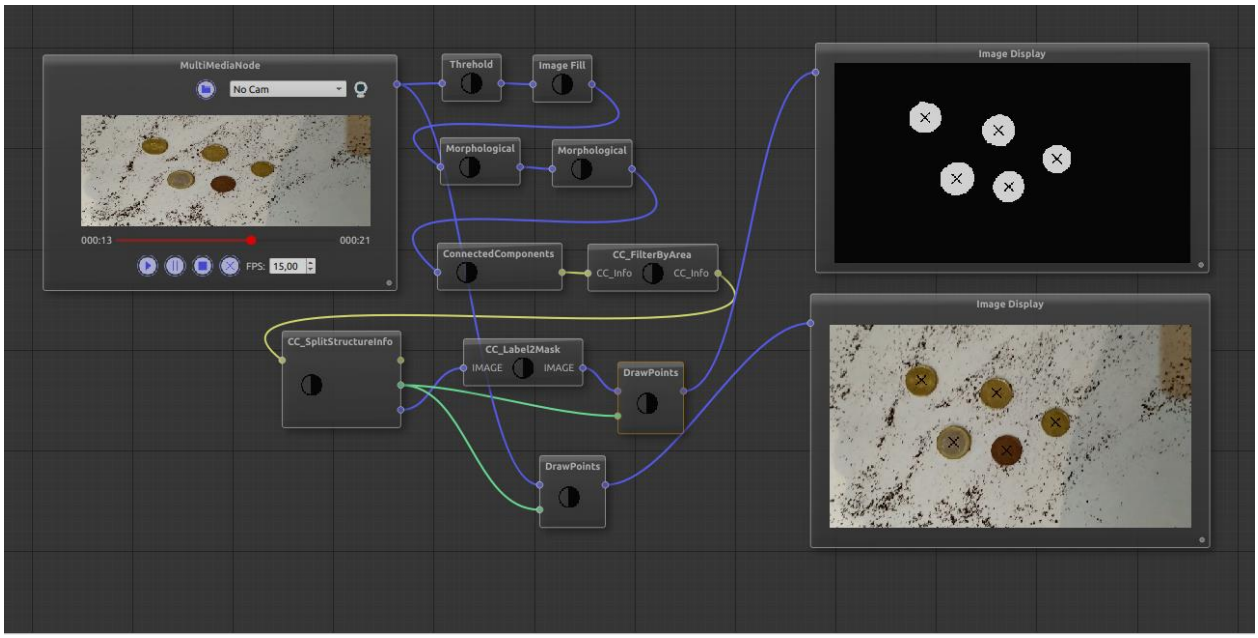


Figura 83. Resultado Final del Problema donde se observan el conjunto de Nodos empleados, sus conexiones y Resultado.

En Figura 83, se ven los resultados finales. Tanto a la imagen original como a la máscara con las monedas, se les ha dibujado únicamente la posición de los centros de estas monedas.

Para dibujar los Puntos se ha empleado el Nodo de "*DrawPoints".* A este nodo, simplemente se le tiene que pasar la posición de los puntos a dibujar, los centroides en este caso, y sobre qué imagen dibujar estos puntos.

Finalmente, como dice el refrán: *"Una imagen dice más que mil palabras".* Se proporciona el siguiente enlace para ver un video de demostración de la aplicación en ejecución.

ENLACE AL VIDEO

5 CONCLUSIONES Y LÍNEAS FUTURAS

5.1 Conclusiones

El presente proyecto surge con la idea de proporcionar una forma alternativa de programación distintas a la forma tradicional, escribiendo código, por una programación visual basada en la utilización de bloques. Esta forma resulta ser mucho más intuitiva y fácil de realizar.

El objetivo del proyecto era crear una aplicación que implementara esta idea aplicada al campo de la visión por ordenador. Debido a que lo que se buscaba era la viabilidad de la idea, en la aplicación solo se ha implementado lo esencial para demostrar dicha viabilidad, tratando de resolver un problema clásico del campo de la visión por ordenador empleando exclusivamente bloques.

Las diferentes conclusiones que sacadas son:

- La conclusión más importante que se obtiene del proyecto es la confirmación de su viabilidad, tal y como se ha podido apreciar en la sección 4: *Software en Funcionamiento.* Quedando solucionado el ejemplo propuesto.
- La aplicación presenta una gran flexibilidad, ya que permite probar todas las combinaciones, tanto a nivel de configuración de una misma operación, como de combinaciones de operaciones que se quieran, y todo ello en lo que podemos denominar prácticamente a Tiempo Real. Además, facilita la visualización de los resultados obtenidos en todo momento.
- Se disminuye el tiempo de desarrollo. Escribiendo código, se suele emplear, como mínimo, el doble de tiempo. Debido a que primero se tiene que aprender la utilidad de cada técnica y su campo de aplicación. Luego, se tiene que aprender a implementarla escribiendo código. En el caso de esta aplicación, solo se emplearía el primer punto, puesto que para el segundo la propia aplicación se encarga de ocultárnoslo.
- Se elimina el tiempo de compilación ya que simplemente debemos ejecutar una única vez la aplicación. Una vez esté en ejecución, se pueden realizar todas las configuraciones que se deseen. Escribiendo código, cada vez que se modificase un parámetro se tendría que volver a compilar.
- La fluidez de la aplicación depende del hardware donde se ejecute. Hay que tener en cuenta que los algoritmos de visión por ordenador suelen llevar una carga de procesamiento elevada.
- Tal y como se ha observado, incluso en un campo exigente, a nivel de cómputo, como es el de la visión por ordenador, la aplicación es capaz de funcionar correctamente. Esto abre la posibilidad de extender su desarrollo y utilidad a casi cualquier campo que se desee. Algunos de estos campos se comentarán a continuación en el apartado de Futuras Mejoras.

5.2 Futuras Mejoras

Debido a que el siguiente proyecto compone una prueba piloto para ver la viabilidad de esta forma alternativa de programación, las mejoras que se le pueden añadir al proyecto son prácticamente ilimitadas. Por lo tanto, en este apartado sólo se comentarán aquellas más interesante a desarrollar.

- La primera mejora que se debe añadir al proyecto es la verificación de que puede compilarse sin ningún problema en otros sistemas operativos: Windows y Mac OS. Al estar desarrollado con librerías multiplataforma, no debería dar muchos problemas poder realizar esta mejora.
- Refactorizar el código existente. Aunque actualmente el código funciona, hay secciones del mismo que deberían reestructurarse.
    - Un claro ejemplo de esta reestructuración es el diseño de la ventana de configuración de la operación Morfológica. En esta ventana, se dispone de una tabla para configurar el kernel. Sería buena idea crear una clase independiente que se encargase unicamente de crear y gestionar la tabla (Widget). De esta forma, en la ventana de configuración solo se tendría que instanciar a esta clase, simplificando el diseño actual. Además, a la hora de implementar la operación de convolución, la cual emplea un kernel, el diseño de la ventana se simplificará enormemente, puesto que la tabla para configurar el kernel ya estaría diseñada.
    - Otro ejemplo, sería fusionar el Nodo ImageLoader (cargar una imagen) dentro de MultimediaLoader. Así se tendría agrupado en un solo Nodo todas las funcionalidades de carga de archivos o empleo de webcam.
- Separar el diseño empleando los bloques y sus conexiones de las visualizaciones. Actualmente, se dispone de Nodos que permiten visualizar los resultados que se van obteniendo al combinar distintas operaciones. En estos nodos se visualiza el resultado obtenido. Sería buena idea llevar estas visualizaciones a otro lado. Permitiendo así tener una pestaña para el diseño y otro para visualizar resultados. La idea sería crear un Widget que permitiese poder seleccionar que visualizador/es se desea ver.

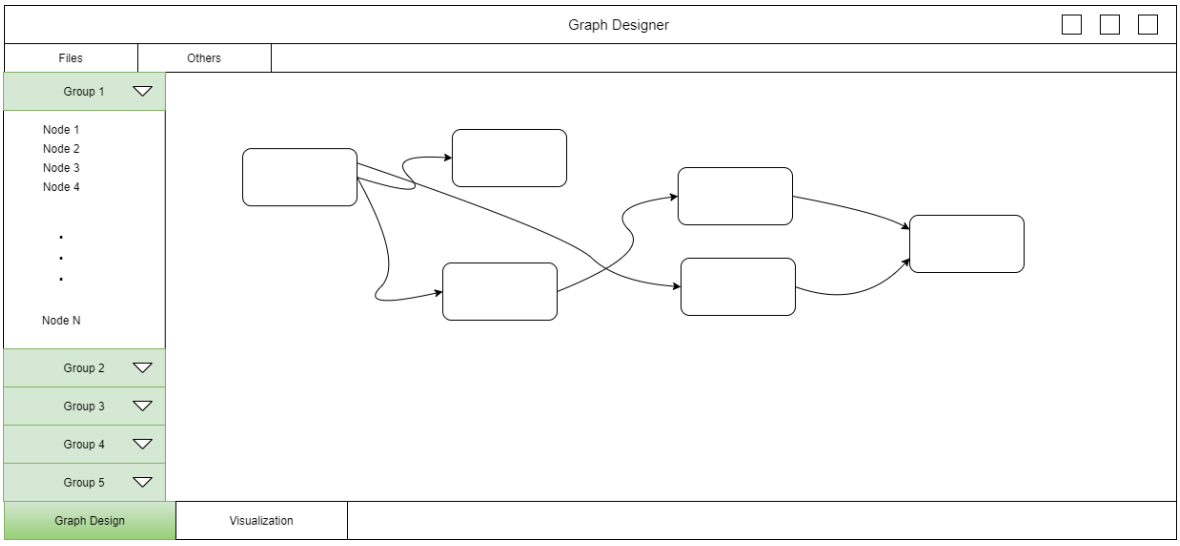


Figura 84. Idea Conceptual de las Futuras Mejoras. Pestaña "Graph Design"

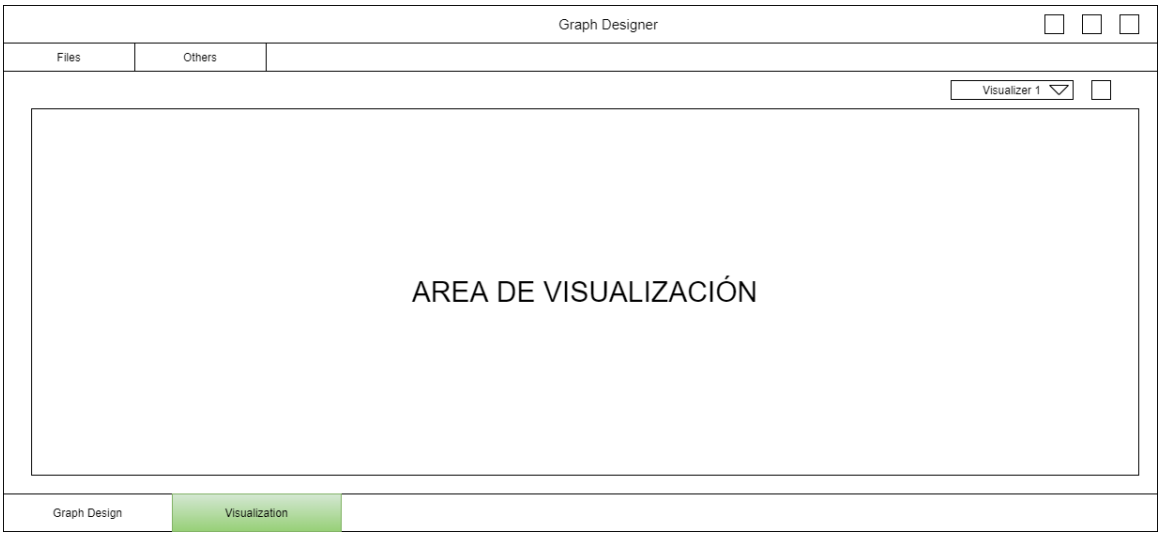


Figura 85. Idea Conceptual de las Futuras Mejoras. Pestaña "Visualization".

- La forma actual para añadir bloques es pulsando el botón derecho y seleccionando uno de la lista que se despliega. En esta mejora, nos gustaría poder tener una sección en la cual se pudiese visualizar en todo momento todos los nodos disponibles. Además, darnos la facilidad de poder añadir cualquiera de estos Nodos, simplemente arrastrándolos a la escena. Imitando así el comportamiento de Simulink.
- En este proyecto, se han implementado varias funciones de la librería de OpenCV para llevar a cabo el ejemplo. Todas las funciones implementadas pertenecen a una sección de la librería llamada "*Image Processing*". Estaría bien terminar de implementar todas las funciones disponibles en esta sección.
- Añadir nuevos Nodos que implementen nuevas funcionalidades de otras librerías.
    - Se podría emplear la librería matemática "Armadillo", para empezar a implementar operaciones con vectores y matrices.
- Añadir Nodo que nos permitiesen recibir información de Hardware externo.
    - Se podría añadir un Nodo que proporcionase la capacidad de recibir datos procedentes de hardware externo, como podría ser un Arduino, una Raspberry, una Beaglebone, etc. La idea, sería poder conectar estas placas por el puerto serie del ordenador. El Nodo debería ser capaz de detectar a la placa y conectarse a ella para empezar a recibir datos que la placa estuviese enviando. De esta manera, se recibirían datos en tiempo real con los cuales se podrían realizar diferentes acciones como: visualizarlos, operar con ellos…

REFERENCIAS

[1] «Leguajes de Programacion: Primera Generación,» [En línea]. Available: https://sites.google.com/site/lenguajesdeprogramacion1233/lenguajes-de-programacion/lenguaje-de-programacion-primera-generacion.

[2] U. O. d. Catalunya. [En línea]. Available: http://cv.uoc.edu/moduls/XW02_79049_00373/web/main/m4/v2_2.html.

[3] M. Deeb, «Programming Language Generations,» [En línea]. Available: https://deeb00.wordpress.com/2018/04/28/programming-language-generations/.

[4] «Fifth-generation programming language,» [En línea]. Available: https://en.wikipedia.org/wiki/Fifth-generation_programming_language.

[5] N. INSTRUMENTS, «LabVIEW,» [En línea]. Available: https://www.ni.com/es-es/shop/labview.html.

[6] MathWorks, «Simulink,» [En línea]. Available: https://es.mathworks.com/products/simulink.html.

[7] WIKIPEDIA, «Qt (software),» [En línea]. Available: https://en.wikipedia.org/wiki/Qt_(software).

[8] T. C. Qt, «About Us,» [En línea]. Available: https://www.qt.io/company.

[9] M. Á. Acera García, «Capítulo 16. Programación Orientada a Objetos,» de *Manual Imprescindible C/C++*, ANAYA, 2010, pp. 305-323.

[10] R. Lafore, Object Oriented Programming in C++, SAMS.

[11] «Atlantic International University,» [En línea]. Available: http://cursos.aiu.edu/Lenguajes%20de%20Programacion%20Orientados%20a%20Objetos/PDF/Tema%204b.pdf.

[12] OpenCV, «About,» [En línea]. Available: https://opencv.org/about/.

[13] A. Kaehler y G. Bradski, Learning OpenCV 3 COMPUTER VISION IN C++ WITH THE OPENCV LIBRARY, O'REALLY, 2016.

[14] WIKIPEDIA, «OpenCV,» [En línea]. Available: https://en.wikipedia.org/wiki/OpenCV.

[15] A. Karhler y G. Bradski, «Threshold Operations,» de *Learning OpenCV 3 COMPUTER VISION IN C++ WITH THE OPENCV LIBRARY*, 2016, pp. 255-256.

[16] A. Karhler y G. Bradski, «Adaptative Threshold,» de *Learning OpenCV 3 COMPUTER VISION IN C++ WITH THE OPENCV LIBRARY*, 2016, pp. 259-260.

[17] A. Kaehler y G. Bradski, «Image Morphology,» de *Learning OpenCV 3 COMPUTER VISION IN C++ WITH THE OPENCV LIBRARY*, 2016, pp. 275-289.

[18] OpenCV, «Image Processing - Miscellaneous - Image Filtering,» [En línea]. Available: https://docs.opencv.org/4.2.0/d4/d86/group__imgproc__filter.html.

[19] OpenCV, «Image Processing - Miscellaneous - Structural Analysis and Shape Descriptors,» [En línea]. Available: https://docs.opencv.org/4.2.0/d3/dc0/group__imgproc__shape.html#ga107a78bf7cd25dec05fb4dfc5c9e765f.

[20] OpenCV, «Image Processing - Miscellaneous - Image Transformations,» [En línea]. Available: https://docs.opencv.org/4.2.0/d7/d1b/group__imgproc__misc.html#gae8a4a146d1ca78c626a53577199e9c57.

[21] OpenCV, «Image Processing - Miscellaneous - Drawing Functions,» [En línea]. Available: https://docs.opencv.org/4.2.0/d6/d6e/group__imgproc__draw.html#ga644c4a170d4799a56b29f864ce984b7e.

[22] T. Q. Company, «Qt Documentation - Signals & Slots,» [En línea]. Available: https://doc.qt.io/qt-5/signalsandslots.html.

[23] H. M. Deitel y P. J. Deitel, CÓMO PROGRAMAR EN C++. SEXTA EDICIÓN, PEARSON, 2008.

6 ANEXOS

https://github.com/JesusCRIS90/GraphdDesingAlgorithm

https://github.com/paceholder/nodeeditor