UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

ENTORNO DE DESARROLLO DE APLICACIONES DE VÍDEO-SEGURIDAD MULTICÁMARA.

Carlos Sánchez Bueno.
Tutor: José María Martínez Sánchez

-PROYECTO FIN DE CARRERA-

Departamento de Tecnología Electrónica y de las Comunicaciones
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Junio 2014

ENTORNO DE DESARROLLO DE APLICACIONES DE VÍDEO-SEGURIDAD MULTICÁMARA.

Carlos Sánchez Bueno
Tutor: José María Martínez Sánchez

Video Processing and Understanding Lab
Departamento de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Junio 2014

Trabajo parcialmente financiado por el gobierno español bajo el proyecto TEC2011-25995 (EventVideo)

Resumen.

El objetivo principal del proyecto ha sido elaborar una metodología de desarrollo de aplicaciones de vídeo-vigilancia que integren algoritmos de análisis de secuencias de vídeo. El entorno de desarrollo de las aplicaciones es una plataforma de procesado distribuido de vídeo, la cual permite la interconexión de los diferentes elementos que la componen. El proyecto también incluye mejoras en el funcionamiento de la plataforma y nuevas funcionalidades.

Durante la realización del proyecto se han integrado dos algoritmos en el sistema para probar la metodología desarrollada. Para cada algoritmo se han cubierto las diferentes etapas de desarrollo definidas a lo largo del proyecto, para finalmente obtener aplicaciones con interfaz gráfica que permiten procesar las imágenes obtenidas por los sistemas de captura de la plataforma y observar sus resultados.

Abstract.

The main objective of this project has been to develop a methodology for the development of video surveillance applications that make use of video sequence analysis. The development environment of the applications is a framework for distributed video processing, which allows the connection between different elements that form part of the system. The project also includes upgrades in the operation of the framework as well as new functionalities.

During the development of this project two algorithms have been integrated in the system in order to assess the developed methodology. For each algorithm all the steps of development have been covered to finally obtain applications with a graphic user interface that allow the processing of images obtained by the platform's capturing systems and observe its results.

Agradecimientos.

Quiero agradecer en primer lugar a Chema, mi tutor, el haber confiado en mí para realizar este proyecto y el haberme dado la oportunidad, junto a Jesús, de formar parte del VPU. También agradecer a los compañeros del laboratorio, Luis, Rafa, Marcos, Álvaro y Santiago por un lado y a los compañeros de carrera y laboratorio Pencho, Ángel, Raúl, Alberto, Alejandro y al resto que habéis hecho ayudado a crear un ambiente de trabajo excelente en el laboratorio.

Agradecer también a todos los amigos que he hecho durante la carrera, especialmente a Carlos, Hugo, Sergio y Jose, compañeros de prácticas, de negocios varios y de momentos inolvidables. Habéis sido piezas fundamentales para que logre terminar la carrera. No me quiero olvidar de los compañeros del "Macabi de Levantar", grandes momentos dentro y fuera de los campos de baloncesto junto a vosotros.

No me puedo olvidar de mi familia, mi padres Germán y Alicia y a mi hermana Cristina. Gracias a ellos que siempre me han apoyado y me han ayudado a conseguir lo que me proponía, sin ellos posiblemente no habría llegado hasta aquí. Agradecer especialmente a mi hermana, que ha tenido la paciencia suficiente para aguantarme tantos años y siempre ha estado a mi lado.

Agradecer también a mi novia Cristina, todos los buenos momentos que hemos vivido y los que viviremos juntos. Gracias por haber sabido llevar mis largas tardes de prácticas y estudio en las que te parecía que trabajaba a velocidad de caracol. Muchas gracias por ayudarme a desconectar cuando era necesario. Gracias por todo.

Por último, pero no menos importante agradecer a Irene, Pablo, Álvaro, Jose, Eduardo, Carlos y Sergio por todos estos años de amistad. Nos conocemos desde hace mucho y juntos llegamos a evolucionar como personas y como grupo de amigos. Gracias porque con vosotros se que nunca me va a faltar diversión.

Gracias a todos.

Carlos Sánchez Bueno.

Junio 2014.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción.

1.1. Motivación.

En la actualidad el interés por el análisis de secuencias de vídeo está en constante crecimiento debido en gran parte al aumento de los sistemas de vídeo-vigilancia. Es común que estos sistemas estén formados por múltiples cámaras supervisadas por uno o más operarios. Para facilitar el trabajo de estos se está trabajando en algoritmos de análisis y generación de alarmas o eventos. Algunos tipos de algoritmos utilizados en vídeo-vigilancia son de detección de robo/abandono, seguimiento de personas, etc.

Todos estos algoritmos se desarrollan y prueban en un entorno *offline* utilizando secuencias grabadas, pero en el ámbito de la seguridad lo interesante es integrarlos en aplicaciones que funcionen en tiempo real [1].

La motivación de este proyecto será abordar la problemática de la creación de aplicaciones de vídeo-vigilancia que funcionen dentro de un entorno multicámara y que puedan analizar tanto vídeos como secuencias obtenidas de cámaras.

1.2. Objetivos.

El objetivo principal del proyecto es establecer una metodología completa de integración de algoritmos dentro de la plataforma de análisis de vídeo distribuido DiVA (Distributed Video Analysis [2]) desarrollada dentro del laboratorio VPULab. Esta metodología incluirá el proceso completo, que comprende desde la programación del algoritmo hasta la creación de un demostrador del mismo. Se establecerán ciertas reglas de sintaxis que deberán cumplir los algoritmos para que la integración posterior en la plataforma sea lo más rápida y sencilla posible.

Para alcanzar la meta se ha planteado la integración de dos algoritmos diferentes.

Tras una primera aproximación a la metodología, esta se irá refinando conforme se avance en el desarrollo de las aplicaciones que implementen el algoritmo. Además el objetivo es definir un conjunto de plantillas y ejemplos para facilitar futuras integraciones de algoritmos en la plataforma.

Como objetivos paralelos a la elaboración de la metodología se hará un estudio de distintos aspectos mejorables en la plataforma y se intentará ponerles solución. Se revisarán los módulos capturadores de imágenes para evaluar si es necesario cambiar alguno. Se estudiará transmitir las imágenes codificadas, y como afectaría esto al ancho de banda necesario y a la velocidad de ejecución de los algoritmos. Por último se analizará introducir soporte para algoritmos multicámara.

1.3. Estructura de la memoria.

La memoria del proyecto se divide en los siguientes capítulos:

- Capítulo 1. Introducción: introducción, motivación y objetivos del proyecto.
- Capítulo 2. Estado del arte: sistemas de vídeoanálisis previos, plataforma DiVA, OpenCV y Qt.
- Capítulo 3. Diseño: mejoras realizadas en la plataforma DiVA y especificaciones de diseño de aplicaciones.
- Capítulo 4. Desarrollo de aplicaciones: metodología para la integración de algoritmos en aplicaciones dentro de la plataforma DiVA.
- Capítulo 5. Aplicaciones desarrolladas y resultados: aplicaciones desarrolladas siguiendo la metodología expuesta.
- Capítulo 6. Conclusiones y trabajo futuro.
- Referencias y anexos.

Capítulo 2

Estado del arte.

Este capítulo proporciona una visión general del trabajo realizado previamente en las áreas relacionadas con el objetivo del proyecto. En las siguientes secciones se describirán sistemas ya existentes (sección 2.1) anteriores a la plataforma DiVA, el sistema desarrollado en el VPULab y sobre el que se realizará el trabajo (sección 2.2), y alguna de las herramientas utilizadas en el proyecto (secciones 2.3 y 2.4).

2.1. Sistemas previos.

Originalmente los sistemas de vídeo-vigilancia estaban formados por un circuito cerrado de televisión (CCTV). Es decir, un conjunto de cámaras que capturaban las secuencias de vídeo que eran transmitidas hasta los paneles de visualización. Tradicionalmente, aunque la señal de vídeo fuera inicialmente digital, la transmisión se realizaba mediante cable coaxial utilizando una señal analógica de vídeo compuesto que llegaba hasta la sala de visualización que normalmente estaba compuesta por una matriz de monitores donde podían verse las imágenes de todas las cámaras. Estos sistemas formarían la primera generación de sistemas de vídeo-vigilancia.

El salto a la segunda generación de sistemas de vídeo-vigilancia se produce al aprovechar que la imagen es capturada en formato digital para realizar un procesado sobre ella. Ahora la imagen no se convierte a formato analógico, sino que es recibida directamente por un ordenador que la procesa y ejecuta sobre las imágenes algún algoritmo de análisis. La mayor parte de los algoritmos utilizados, son algoritmos de detección de eventos en tiempo real que ayuden al supervisor del sistema a detectar eventos producidos en las escenas que graban las cámaras facilitando su labor. Estos sistemas aunque son más avanzados que los de la primera generación tienen el problema de que para gestionar un gran número de cámaras y procesar algoritmos

sobre las señales de vídeo que proceden de las cámaras hacen falta potentes equipos informáticos conectados a las cámaras.

En la tercera generación se produce un avance en cuanto al diseño de los sistemas. Se comienza a plantear una arquitectura distribuida frente a la tradicional centralizada. También se remarca la importancia de que el sistema debe ser escalable al estar en constante aumento el número de cámaras presentes en los entornos vídeovigilados. Este aumento del número de fuentes de vídeo abre las puertas a algoritmos multicámara que utilicen simultáneamente imágenes de diferentes dispositivos de captura.

La descentralización producida en los sistemas de esta generación permitirá distribuir las diferentes etapas de procesado entre diferentes elementos de la red, pudiendo de esta manera ejecutar algoritmos más complejos en tiempo real al ser procesada cada parte del algoritmo en una máquina diferente. Por ejemplo, para un algoritmo que primero detecte objetos, luego los identifique, haga un seguimiento de ellos a través de la escena y que finalmente haga una detección de eventos podrían separarse cada una de estas partes para ser ejecutadas en equipos diferentes consiguiendo una mayor capacidad de cómputo sin tener una máquina especialmente diseñada para ello.

Para establecer una arquitectura de procesado distribuido es necesario diseñar la red de distribución de cuadros para sustituir los tendidos de cable coaxial que existían en las generaciones anteriores. Esta red deberá adecuarse a las necesidades del sistema teniendo en cuenta aspectos importantes como el ancho de banda, la privacidad de la red, etc.

Estos sistemas de vídeo-vigilancia se encuentran implantados en multitud de escenarios. Aquí se muestran algunos ejemplos: [3] propone un sistema de vídeo-vigilancia en un parking con detección de eventos. [4] muestra un sistema de vídeo-vigilancia de carreteras en el que se utilizan algoritmos para monitorizar el tráfico de vehículos y poder mejorar la seguridad en las carreteras.

2.2. Distributed Video Analysis Framework (DiVA).

La plataforma DiVA[2] desarrollada dentro del VPULab está enmarcada dentro de los sistemas de vídeo-vigilancia multicámara de procesado distribuido (tercera generación según [1]). Fue desarrollada para proporcionar un entorno en el que poder implementar aplicaciones de vídeo-seguridad.

Los principales criterios de desarrollo tenidos en cuenta en el desarrollo de la plataforma fueron escalabilidad, eficiencia y generalidad. La plataforma tenía que ser escalable para permitir añadir nuevos módulos en el sistema como nuevos algoritmos, nuevas fuentes de vídeo, etc. Debía ser eficiente y añadir la mínima carga computacio-

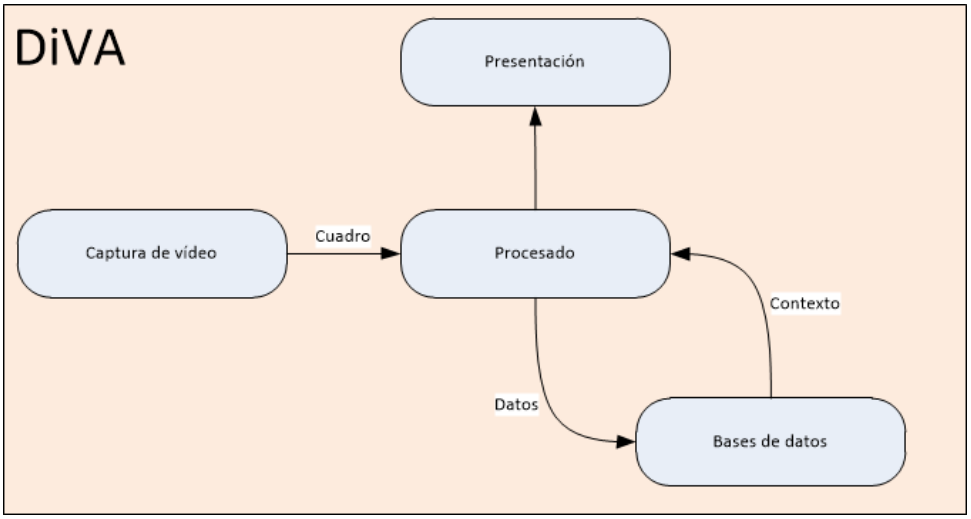

Figura 2.1: Arquitectura de DiVA.

nal al procesado de los algoritmos. En cuanto a la generalidad, la plataforma debía desarrollarse con protocolos y herramientas genéricas.

Al ser una plataforma de análisis de vídeo distribuido se diseñó una arquitectura descentralizada que separara los diferentes subsistemas funcionales de DiVA. En total fueron definidos cuatro subsistemas: Captura de datos, Procesamiento, Presentación y Bases de datos.

El subsistema de captura de datos es el encargado de capturar cuadros de las fuentes de vídeo para enviarlos al subsistema de procesado.

El subsistema de procesado cumple la función de analizar los cuadros recibidos del sistema de captura y también sirve de interfaz entre los algorimtos y el resto de la plataforma.

El subsistema de presentación se encarga de mostrar por pantalla los resultados del procesado de las imágenes.

El subsistema de bases de datos cumplirá las tareas de almacenar los resultados de análisis de los algoritmos y proporcionar datos de contexto a los algoritmos de análisis.

Esta arquitectura se puede observar en la figura 2.1.

Para la interconexión entre los diferentes subsistemas de la plataforma y la distribución de la información la arquitectura escogida fue cliente-servidor utilizando TCP/IP sobre Ethernet debido a las necesidades propias del sistema. Al ser un entorno de procesado de secuencias de vídeo era necesario utilizar un protocolo que asegurara la integridad de los datos, por esta razón se descartó el uso de protocolos no orientados a conexión como UDP.

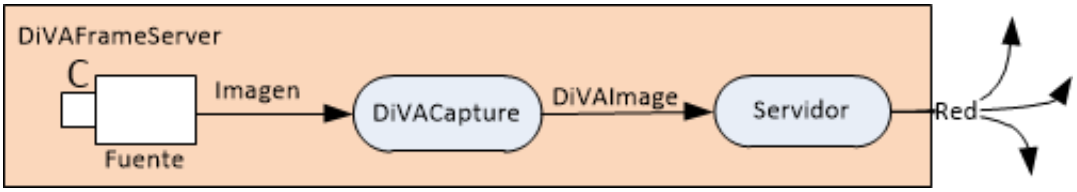

Figura 2.2: Esquema subsistema de captura.

2.2.1. Subsistema de captura.

Los subsistemas de captura de la plataforma DiVA se pueden dividir en tres partes: fuente de cuadros, interfaz y servidor de cuadros.

La fuente de cuadros es el elemento hardware del sistema de captura, es decir, las diferentes cámaras que se utilicen para obtener las imágenes que luego se procesarán. También es posible que la fuente de cuadros sea un archivo de vídeo.

Para introducir las imágenes capturadas en la plataforma se define un elemento intermedio que hace de interfaz entre ambas partes. Este elemento se implementa con la clase `DiVACapture` creando una clase derivada de esta para cada tipo de fuente (cámara IP, cámara Firewire, cámara USB, fichero de vídeo, etc.). El objetivo principal es convertir las imágenes obtenidas en el formato propio de la plataforma: `DiVAImage`.

Finalmente con las imágenes en el formato de la plataforma se crea un servidor de cuadros (FrameServer) para poder distribuirlas al resto de la plataforma. Para cada capturadora disponible en el sistema existe un servidor de cuadros que implementa el subsistema de captura completo. La figura 2.2 muestra un esquema de todo el subsistema.

2.2.2. Subsistema de procesado.

El subsistema de procesado de la plataforma DiVA permite tomar las imágenes que proceden del subsistema de captura, procesarlas y enviarlas al subsistema de presentación o almacenarlas. Durante el procesado puede ser necesaria información adicional que se recibirá del subsistema de bases de datos. También es posible que el resultado de procesar el algoritmo sea información que almacenar en el subsistema de bases de datos para ser utilizado por otros módulos diferentes.

El subsistema de procesado puede estar compuesto por un único módulo de procesado o por varios (figura 2.3). La comunicación entre los diferentes módulos se realiza nuevamente mediante una arquitectura cliente-servidor. La comunicación con el subsistema de presentación se verá en la sección 2.2.3 y con el subsistema de bases de

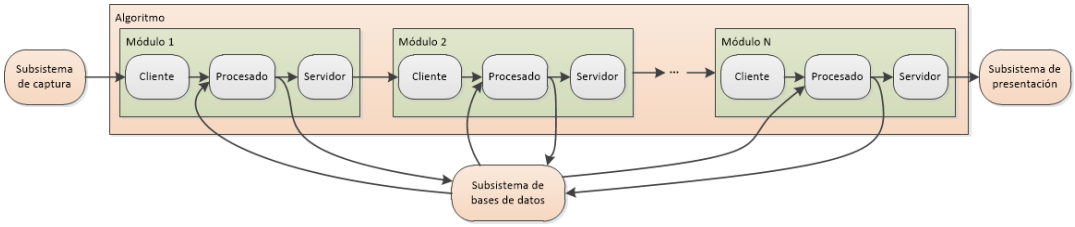

Figura 2.3: Subsistema de procesado con varios módulos.

datos en la sección 2.2.4.

Cada módulo del subsistema de procesado estará compuesto por un cliente que se conecte a la fuente de cuadros (un DiVAFrameServer u otro módulo de procesado) y el algoritmo que procesa la imagen.

Una vez procesadasn las imágenes estas serán enviadas al subsistema de presentación o se almacenarán los resultados en archivos o se almacenarán en el subsistema de bases de datos.

2.2.3. Subsistema de presentación.

El subsistema de presentación es el encargado de mostrar los resultados producidos en el subsistema de procesado. Este subsistema puede estar implementado junto al subsistema de procesado o puede ser independiente del mismo.

En el caso de que la implementación se haga en una sola aplicación (subsistema de procesado y subsistema de presentación en un mismo programa) no es necesario establecer un sistema de comunicación cliente-servidor entre ambos módulos, es posible mostrar los resultados directamente tras procesarlos.

En el caso de encontrarse separados ambos subsistemas (procesado y presentación) se establecerá una arquitectura cliente-servidor entre ambos sistemas, en la que el sistema de procesado creará un servidor de cuadros y el sistema de presentación actuará como cliente recibiendo las imágenes para mostrarlas.

2.2.4. Subsistema de bases de datos.

El subsistema de bases de datos de la plataforma DiVA maneja distintos tipos de información. Actualmente el subsistema maneja tanto información proveniente del análisis de datos como información de contexto de aplicación. Para poder utilizar de una manera eficiente ambos tipos de información se han diseñado dos tipos diferentes de bases de datos:

- Una base de datos para almacenar resultados del análisis de los algoritmos. Este

módulo, que recibe el nombre de DataServer, se encarga de almacenar los datos, parciales o finales, arrojados por los algoritmos de análisis, así como información acerca de la configuración utilizada en la captura de las imágenes, procesado y etiquetado de estas.

- Una base de datos para almacenar descripciones de contexto de aplicación, modelos de objetos, parámetros de algoritmos,... en resumen almacena conocimiento que puede ser usado tanto a priori como a posteriori por los algoritmos de análisis. Este módulo, que recibe el nombre de ContextServer, se encarga de proporcionar un contexto o dominio de aplicación a cada algoritmo (información antes de la etapa de procesado de los algoritmos).

2.3. OpenCV.

OpenCV (Open Computer Vision[1]) es una librería multiplataforma muy extendida en el ámbito del tratamiento de imágenes. Se distribuye bajo licencia BSD y es de código abierto. Funciona en Windows, Mac OSX y Linux y está programada en C/C++, Python y Java. Inicialmente fue desarrollada por el departamento Intel Research, perteneciente a Intel y en la actualidad el desarrollo y soporte de la librería lo continúa la fundación OpenCV.org.

OpenCV implementa en la actualidad más de 2500 algoritmos de procesado de imágenes. La librería cuenta con una gran variedad de funciones, desde funciones básicas para trabajar con una imagen hasta algoritmos de detección y seguimiento de objetos, visión artificial para robots, creación de modelos 3D o reconocimiento facial.

Se ha escogido utilizar OpenCV para la realización del proyecto por ser una librería ampliamente utilizada dentro del laboratorio, facilitando el uso de los algoritmos ya existentes y su posterior integración en la plataforma DiVA que también hace uso de esta librería en sus funciones internas. Otra de las razones que han impulsado la elección, es que al ser *software* libre se distribuye de forma gratuita.

La versión que se ha utilizado de OpenCV ha sido la 2.4.3 por ser la última disponible al comienzo del proyecto.

2.4. Qt.

Qt[2] es una librería multiplataforma ampliamente utilizada para desarrollar aplicaciones con interfaz gráfica de usuario (GUI). Originalmente fue desarrollada por la

[1] `http://opencv.org/`(consultado mayo 2014)
[2] `http://qt-project.org/`(consultado mayo 2014)

empresa Trolltech y publicada por primera vez en 1992. En el año 2008 fue adquirida por Nokia quién continuó con el desarrollo de la librería. En 2012 Digia adquirió Qt y es quien desarrolla actualmente la librería. Qt cuenta con un sistema triple de licencias. Tiene una licencia gratuita GPLv2/v3 para el desarrollo de aplicaciones de código abierto y libre, una licencia de pago QPL para el desarrollo de aplicaciones comerciales y, las últimas versiones, una licencia gratis LGPL pensada para aplicaciones comerciales.

Las aplicaciones desarrolladas con Qt son multiplataforma y se pueden ejecutar tanto en sistemas de escritorio como en sistemas móviles y embebidos. Los sistemas de escritorio con los que es compatible son: Windows, Mac OS X, Linux/X11 y Solaris. La compatibilidad con los sistemas móviles ha sido incluida en la versión 5.2 de la librería y permite desarrollar aplicaciones para Android, iOS, Windows Phone y Blackberry 10.

El lenguaje de programación utilizado en Qt es C++. En la librería se establece una jerarquía de clases y herencias en la que la mayoría de ellas está orientada al diseño y programación de interfaces gráficas. Sin embargo ésta no es la única función de Qt, también cuenta con módulos para otras tareas, por ejemplo: comunicación con bases de datos, ejecución de aplicaciones multihilo,...

Qt se distribuye junto al entorno de programación QtCreator especialmente diseñado para la creación de aplicaciones con Qt. En este IDE se puede hacer el tanto el diseño de la interfaz como el desarrollo del código que se ejecuta bajo la interfaz. También se suministra un *Add-in* que permite la integración de Qt dentro de Microsoft Visual Studio. Dentro de este *Add-in* se encuentra QtDesigner, que es una aplicación que permite crear interfaces de forma gráfica.

Se ha escogido la librería Qt para desarrollar las interfaces gráficas de la plataforma DiVA por varios motivos. El principal es la compatibilidad de código al estar ambas escritas en C++. También ha influido en la decisión la amplia documentación existente sobre Qt y la gran comunidad de desarrolladores que utilizan Qt y hacen uso de foros para resolver dudas y otros problemas.

Para la realización del proyecto se ha utilizado la versión 5.1.1 al ser la más reciente en el inicio del mismo.

Capítulo 3

Diseño.

3.1. Introducción.

Después de un análisis detallado de la plataforma DiVA (ver sección 2.2) se han encontrado algunos puntos débiles que deben ser mejorados. Se van a plantear dos líneas de mejora diferentes en la plataforma. Por un lado se realizarán mejoras internas en el sistema modificando alguno de los módulos para solventar las deficiencias encontradas en el sistema (transmisión de imágenes, ...). Por otro lado se va a mejorar la usabilidad del sistema de cara a la integración de algoritmos en el mismo y la creación de demostradores de esos algoritmos.

Al ser un sistema de procesado de vídeo distribuido que utiliza la arquitectura cliente/servidor es de esperar que se transmita una elevada cantidad de información por la red, datos de conexión con el servidor, imágenes, etc. Para poder enviar todos estos datos es necesaria una velocidad alta de transmisión para que no se produzca el efecto de cuello de botella que ralentiza el procesado de cuadros, llegando incluso a perderse algunos al llenarse por completo el *buffer* del servidor. Analizando los datos enviados, la mayor parte del ancho de banda es ocupado por imágenes. Estas imágenes actualmente se envían sin codificar. Por este motivo se propone modificar la rutina de envío de imágenes para que sean codificadas antes de enviarlas para reducir el ancho de banda utilizado y poder aumentar la tasa de imágenes servidas. El problema de añadir la codificación de imágenes al servidor es el aumento del trabajo computacional necesario. Esto puede llegar a ser un problema si el equipo encargado de codificar las imágenes no es lo suficientemente potente para la codificación.

Otro problema que presentaba DiVA estaba en el módulo de captura de imágenes, que utilizaba librerías propietarias con licencias de pago (MIL de Matrox[1]) para cap-

[1] `http://www.matrox.com/imaging/en/products/software/mil/`

turar cuadros de las cámaras Firewire disponibles. La solución era buscar una librería que fuera gratuita, que permitiera utilizar este tipo de cámaras y que fuera posible utilizarla con nuevas cámaras que pudieran ser añadidas al sistema en un futuro.

Otro de los puntos débiles encontrados en el análisis de la plataforma es la no existencia de soporte para algoritmos multicámara. Si bien se podían ejecutar diferentes algoritmos sobre diferentes cámaras o un único algoritmo sobre varias cámaras (en distintas ejecuciones de cada programa) no era posible integrar un algoritmo que necesitara recibir imágenes de más de una cámara a la vez.

A la hora de integrar algoritmos y crear demostradores nos encontramos con la problemática de no existir una metodología definida que guíe el proceso de integración. Hasta el momento cada persona encargada de integrar un algoritmo lo hacía a su manera, complicando la reutilización de código por parte de otras personas o el simple hecho de añadir nuevas funcionalidades a demostradores ya creados. En este capítulo se establecerá la metodología para la integración de algoritmos en la plataforma DiVA solucionando la problemática a nivel de uso existente. Se definirán diferentes niveles de integración que permitan introducir los algoritmos por etapas y siguiendo todos una estructura común. Esta estructura posibilitará crear una guía de integración para facilitar la integración de nuevos algoritmos.

3.2. Mejoras realizadas en la plataforma DiVA.

3.2.1. Transmisión codificada de imágenes.

Para solucionar el problema de la codificación de imágenes se han utilizado funciones de la librería OpenCV (ver sección 2.3), ya utilizada en la plataforma para otras tareas. En concreto las funciones utilizadas han sido `imencode` para la codificación e `imdecode` para la decodificación. Estas funciones permiten la codificación/decodificación de imágenes en memoria, reduciendo el tiempo de proceso al no necesitar guardar los datos en el disco duro. OpenCV soporta formatos de codificación de imágenes con pérdidas y sin pérdidas. A continuación se muestra una lista con los formatos disponibles:

- Mapas de bits de Windows (extensiones .bmp y .dib):
- JPEG (extensiones .jpeg, .jpg y .jpe):
- JPEG 2000 (extensión .jp2):
- *Portable Network Graphics* (extensión .png):

- *Portable image format* (extensiones .pbm, .pgm y .ppm):
- *Sun rasters* (extensiones .sr y .ras):
- TIFF (extensiones .tiff y .tif):

En la implementación de la codificación se han elegido dos de las posibles codificaciones, JPEG y PNG, es decir, una con pérdidas (JPEG) y otra sin pérdidas (PNG). Ambos formatos soportan parámetros que configuran la calidad y el nivel de compresión.

El formato JPEG tiene el parámetro `CV_IMWRITE_JPEG_QUALITY` que puede tomar un valor entre 0 y 100 (95 por defecto). Indica la calidad de la compresión de la imagen. Un valor más alto resultará en una imagen de más calidad pero menos comprimida.

Para el formato PNG el parámetro `CV_IMWRITE_PNG_COMPRESSION` toma valores entre 0 y 9 (3 por defecto). Indica el nivel de compresión de la imagen, un valor más alto nos proporcionará una imagen que ocupa menos espacio pero tardará más tiempo en realizar la codificación.

Para simplificar el uso de esta funcionalidad se han definido cinco modos de codificación con diferentes configuraciones de parámetros y uno de no codificación.

Modo 1 Imágenes sin codificar.

Modo 2 Codificación JPEG con `CV_IMWRITE_JPEG_QUALITY = 100`.

Modo 3 Codificación JPEG con `CV_IMWRITE_JPEG_QUALITY = 90`.

Modo 4 Codificación JPEG con `CV_IMWRITE_JPEG_QUALITY = 50`.

Modo 5 Codificación PNG con `CV_IMWRITE_PNG_COMPRESSION = 2`.

Modo 6 Codificación PNG con `CV_IMWRITE_PNG_COMPRESSION = 8`.

Estos modos de funcionamiento deben indicarse al crear un servidor de imágenes. Después de la llamada al constructor de la clase `DiVAServer` se utiliza el método `DiVAServer::setTypeCodification` que recibe como argumento un número entero que corresponde con uno de los modos indicados antes. Si no se especifica, por defecto no se utilizará ninguna compresión de imágenes (Modo 1).

En el cuadro 3.1 se indica la correspondencia entre los modos de codificación y el valor que debe recibir la función `DiVAServer::setTypeCodification`:

Para añadir esta funcionalidad a la plataforma la implementación se ha extendido la clase de imágenes definida en DiVA (`DiVAImage`) añadiendo dos nuevos métodos a la misma, uno para codificar y otro para decodificar imágenes.

Modo de compresión	Valor recibido por setTypeCodification
Modo 1	Cualquiera excepto 1, 2, 3, 4 y 5
Modo 2	1
Modo 3	2
Modo 4	3
Modo 5	4
Modo 6	5

Tabla 3.1: Correspondecia de modos de compresión.

Para codificar una imagen se hace una llamada al método `DiVAImage::encode`. Recibe como argumentos una cadena de caracteres que contiene la extensión del formato de la imagen al que queremos codificar (`".JPG"`, `".PNG"`,...), un vector de números enteros con los parámetros de la codificación y un puntero a un vector de caracteres en el que se escribirá la imagen codificada. Al ejecutarse, el método codifica los datos del objeto `DiVAImage` declarado utilizando el formato y los parámetros establecidos y devuelve la imagen codificada en el vector de salida. Los datos de la imagen original permanecen inalterados.

Para la decodificación el método a utilizar es `DiVAImage::decode`. Este recibe un vector de caracteres con los datos de la imagen a decodificar y un número entero indicando el modo de color de la imagen (escala de grises, color, canal alpha,...). Esta función detecta automáticamente el formato de compresión de la imagen y la descomprime guardando los datos en el objeto propietario de la función.

En el apéndice B se incluyen los resultados obtenidos tras realizar pruebas de transmisión en el entorno de trabajo real para imágenes de diferentes tamaños y para los diferentes modos de codificación disponibles en el sistema (Tabla 3.1).

3.2.2. Capturadora Firewire.

Hasta el momento para poder capturar cuadros de las cámaras disponibles que utilizan el estándar IEEE 1394 (FireWire) se utilizaba la librería MIL (Matrox Image Library) desarrollada por Matrox. Esta librería, que no dispone de licencia de código abierto, es de pago y además no es posible utilizarla para diferentes modelos de cámaras. Se ha sustituido por la librería VideoInput[2], que es una capturadora de vídeo para Windows. Ha sido diseñada e implementada por Theodore Watson y se distribuye de forma gratuita. La principal función de la librería es simplificar las operaciones con la API DirectShow desarrollada por Microsoft que sustituye a la antigua Video for Windows (VFW) y permite interactuar con datos multimedia.

[2] `http://www.muonics.net/school/spring05/videoInput/` (consultado mayo 2014)

De todas las funciones que proporciona las más destacables y que se han utilizado en la capturadora desarrollada son las siguientes:

listDevices: nos permite enumerar todos los dispositivos conectados al ordenador y conocer el número de dispositivo asociado a ellos.

setIdealFramerate: permite establecer la velocidad (fps) a la que queremos capturar imágenes.

setupDevice: función que sirve para conectar con la cámara y a la que podemos especificar la resolución de las imágenes que recibiremos.

getPixels: función que obtiene la imagen de la cámara.

showSettingsWindow: muestra la ventana de propiedades de configuración de la cámara (brillo, contraste, etc. pero no velocidad ni resolución).

Para el diseño de la capturadora de cuadros se ha utilizado como plantilla la capturadora FireWire antigua sustituyendo las funciones de la librería antigua por las de la nueva. Este módulo capturador será el encargado de extraer imágenes de las cámaras e introducirlas en el la plataforma DiVA, donde luego podrán procesarlas los algoritmos integrados.

Es posible configurar los siguientes parámetros de captura de la cámara:

colorFormat: parámetro que define si la imagen es a color o en blanco y negro.

fps: sirve para controlar la tasa de imágenes capturadas por la cámara.

camera: permite elegir entre las diferentes cámaras instaladas en el equipo. El valor corresponde con el número asignado por DirectShow a cada cámara.

Las opciones establecidas para la cámara deben ser compatibles con la cámara, si no lo fueran la capturadora está diseñada para probar otras opciones automáticamente hasta encontrar la configuración más parecida a la decidida por el usuario.

3.2.3. Soporte a algoritmos multicámara.

La gran parte de los algoritmos existentes utilizan únicamente una fuente de imágenes, pero están apareciendo nuevos algoritmos que requieren el uso de más de una fuente de vídeo. Por este motivo era necesario incluir en DiVA la posibilidad de conectar un algoritmo con más de un servidor de cuadros.

Para lograrlo se ha creado una nueva clase dentro de la librería libDiVABasic llamada `DiVA_Algorithm_multicam`. Esta clase funciona igual que `DiVAAlgorithm`

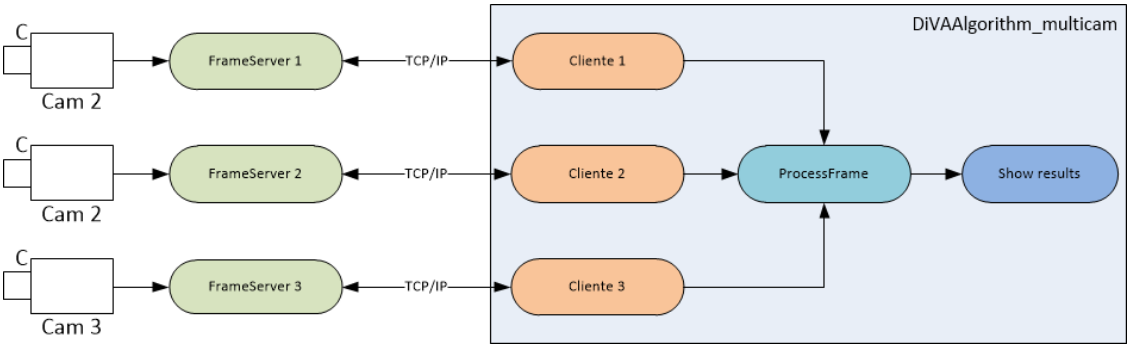

Figura 3.1: Esquema de `DiVA_Algorithm_multicam`.

que implementa la misma funcionalidad para una única cámara. Se han introducido los cambios necesarios para que la nueva clase tenga tres clientes que puedan conectarse a tres servidores de cuadros distintos, en lugar de uno como tenía la clase original. Se ha elegido este número de cámaras porque el escenario en el que el laboratorio tiene instalado un mayor número de cámaras es el hall de la escuela en el que hay tres cámaras PTZ (pan-tilt-zoom). En caso de necesitar soporte para más cámaras habría que repetir el proceso de replicación de elementos llevado a cabo para pasar de una cámara a tres. Si fuera necesario dar soporte para un número grande de cámaras pierde sentido la aproximación utilizada para solucionar el problema y habría que replantear una nueva solución.

Esta es la lista de todos los cambios introducidos en la nueva clase.

- Se han añadido nuevos atributos para almacenar la dirección y el puerto de los servidores de cuadros.
- Se ha añadido por triplicado los servidores.
- Se ha añadido una variable que almacena el último identificador de imagen recibida para cada cliente. También los métodos que se encargan de actualizar cada uno de estos valores.
- Se ha modificado el constructor que ahora recibe como argumentos el puerto y la dirección de cada servidor e internamente se han triplicado las funciones que antes se hacían para un único servidor.
- El destructor ahora también libera los recursos de los nuevos clientes.
- El método `init` ahora inicializa los tres clientes.
- El método `conect2FrameServer` ahora conecta los tres clientes de la clase con los tres servidores especificados en el constructor de la clase.

- Se ha modificado el método `receiveFrame` para que ahora los tres clientes reciban la imagen siguiente a la última recibida de cada servidor.
- En el método `process` se ha modificado la rutina de reconexión con los servidores para que en caso de pérdida de conexión con uno intente reconectar los tres.
- La función `setEnd` finaliza los tres clientes.
- Se ha cambiado la sintaxis de la función `processFrame` que ahora recibe las tres imágenes como argumentos. Internamente no se ha modificado nada porque esta función tiene que ser sobrescrita en clases derivadas.
- En el método `getFrame` se ha cambiado el uso de la función `receiveFrame` para utilizar la versión multicámara y para los casos en los que la imagen se obtiene de un *buffer* se ha modificado para que ahora lo reciba de los tres *buffers* que hay.

3.3. Metodología de integración.

Para la integración en DiVA de nuevos algoritmos se ha definido una metodología que divide el proceso en tres niveles de integración. En cada nivel el programa que ejecuta el algoritmo deberá cumplir unos requisitos diferentes.

3.3.1. Nivel 1: Preparación para la integración.

Este primer nivel es la antesala a la integración del algoritmo en la plataforma DiVA. Su finalidad es preparar el código para facilitar la implementación de los siguientes niveles en los que el algoritmo se ejecutará dentro de DiVA.

El programa resultado de este nivel debe ser capaz de leer ficheros de vídeo, procesarlos, y generar un archivo de salida con los resultados obtenidos por el algoritmo. Además debe cumplir una serie de requisitos para su posterior integración en la plataforma.

1. Debe estar programado en C++. Si el algoritmo está escrito en otro lenguaje deberá hacerse la migración.
2. Tiene que haber una clase que contenga la implementación del algoritmo. No obstante, es posible que esta clase internamente haga uso de otras clases distintas según sea necesario o no.

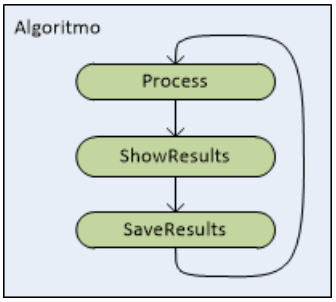

Figura 3.2: Esquema nivel 1 integración.

3. La clase que implementa el algoritmo obligatoriamente tendrá, como mínimo, los siguientes métodos:

   a) Constructor que inicialice los parámetros necesarios, bien sea con unos valores por defecto o leyéndolos de un archivo de configuración, y que reserve los recursos que se usarán en la ejecución del algoritmo.

   b) Destructor que libere los recursos reservados.

   c) Método para procesar las imágenes, deberá ser público porque será llamado desde el encapsulador de la clase DiVA. Este método recibirá como argumento una imagen en formato OpenCV (`IplImage`).Por homogeneidad el método tendrá la siguiente sintaxis:

```
1 void * process(IplImage *frame);
```

   d) Método que muestre por pantalla los resultados de ejecución del algoritmo. Deberá mostrar tanto resultado final como resultados parciales. Si el algoritmo produce resultados que no son vídeo se tendrá en cuenta a la hora de visualizarlos o no. Tendrá el siguiente formato:

```
1 void showresults();
```

   e) Método que guarde en ficheros los resultados del algoritmo. Deberá almacenar en un vídeo los resultados parciales y finales. También almacenará los resultados no visuales. El formato de la función será el siguiente:

```
1 void saveResults();
```

   f) Adicionalmente puede tener métodos para configurar parámetros de ejecución del algoritmo aunque estos no son obligatorios. El formato recomen-

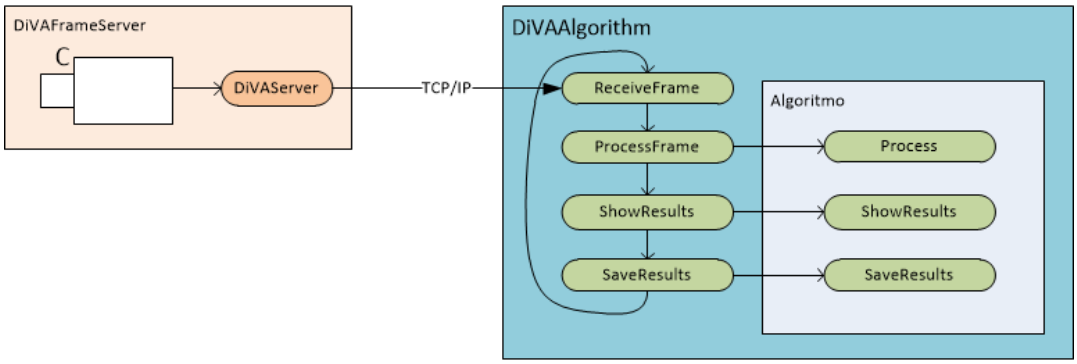

Figura 3.3: Esquema nivel 2 de integración.

dado para estos métodos es el siguiente (dependiendo del tipo de dato del parámetro la definición variará):

```
1 void setParameter(int value);
```

3.3.2. Nivel 2: Integración en la plataforma DiVA.

En este nivel el algoritmo ya se integra dentro de DiVA. Un algoritmo que se encuentre en este nivel de integración deberá producir un programa que se conecte con los servidores de imágenes de la plataforma, reciba imágenes de ellos, las procese y guarde los resultados en un archivo o en los servidores de bases de datos de DiVA.

La clase desarrollada en el nivel anterior se encapsula utilizando la clase encargada de implementar la conectividad con el entorno de procesado distribuido (`DiVA_Algorithm`). Hará uso de los métodos especificados en el nivel anterior.

Como se puede observar en la figura 3.3, en este nivel la fuente de imágenes será un servidor de cuadros que capture imágenes de una cámara o de un archivo de vídeo. Utilizando una arquitectura cliente servidor sobre TCP/IP recibirá las imágenes y las procesará utilizando el método `process` definido en el nivel anterior dentro de la función `ProcessFrame` disponible en DiVAAlgorithm con el siguiente formato:

```
1 void processFrame(DiVAImage *frame);
```

En este nivel hay dos formatos de salida, por pantalla y en archivo de vídeo. Para mostrar por pantalla los resultados de procesar el algoritmo la función `showResults` de `DiVAAlgortithm` llamará a la función `showResults` del algoritmo implementada en el nivel 1. Finalmente se generará un archivo de vídeo con los resultados utilizando la función `saveResults` de DiVA que llamará al método `saveResults`

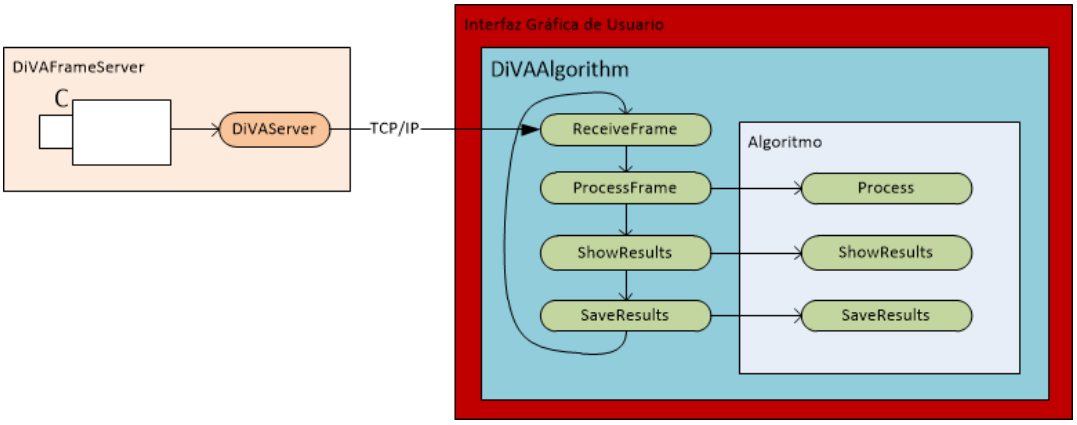

Figura 3.4: Esquema nivel 3 de integración.

del algoritmo.

3.3.3. Nivel 3: Interfaz gráfica de usuario (GUI).

Este es el último nivel de integración que un algoritmo puede tener dentro de la plataforma DiVA. Hasta este punto todos los programas se ejecutaban en una consola de comandos y la interactividad de la que disponían era bastante limitada o nula, según el caso de cada algoritmo.

Este es el nivel de mayor complejidad y el que más posibilidades de desarrollo tiene, por ello se han definido tres subniveles dentro del mismo. Cada subnivel dará lugar a una aplicación con su interfaz gráfica.

3.3.3.1. Nivel 3.1: Aplicación de desarrolladores.

Este nivel define la aplicación más completa de todas porque su público objetivo es el de personas que se dedican al desarrollo de algoritmos. Cualquier aplicación de este nivel debe cumplir las siguientes especificaciones:

1. Permitir elegir el servidor de cuadros con el que conectarse, desconectarse y cambiar entre diferentes servidores.

2. Mostrar los resultados (totales y parciales) integrados dentro de la interfaz con la posibilidad de elegir cuáles se muestran y cuáles no.

3. Configurar los parámetros de ejecución del algoritmo dentro de la misma ventana de ejecución.

3.3.3.2. Nivel 3.2: Aplicación de clientes.

Las aplicaciones de este nivel están destinadas a un público no especializado, por esta razón son más sencillas que en el nivel anterior. Las especificaciones para este nivel son:

1. Permitir elegir el servidor de cuadros con el que conectarse, desconectarse y cambiar entre diferentes servidores.
2. Mostrar el resultado final dentro de la interfaz.
3. Configurar los parámetros en una ventana independiente de la aplicación.

3.3.3.3. Nivel 3.3: Aplicación de clientes ligera.

En este nivel se hace un uso más intensivo de la capacidad de procesado distribuido de la plataforma DiVA. Se separa en dos partes, como mínimo, el procesado del algoritmo. Por un lado estarán los módulos del algoritmo y por otro el visor de resultados. La comunicación entre ambos utilizará una arquitectura cliente-servidor aprovechando que esta característica ya está implementada en la plataforma.

La idea principal de este nivel es separar en dos máquinas diferentes la ejecución del algoritmo y la visualización del mismo, permitiendo tener diferentes algoritmos ejecutándose en un mismo ordenador a los que luego se conectará el visor de algoritmos para ver los resultados. Con esto conseguimos poder ver los resultados desde un ordenador con menos potencia computacional sin reducir la tasa de imágenes procesadas por segundo.

El resultado de este nivel serán por lo tanto dos aplicaciones independientes, por un lado habrá una aplicación de consola que ejecuta el algoritmo (sin interfaz gráfica) y por otro lado estará el visor (con interfaz gráfica). Las especificaciones que deben cumplir los programas que ejecutan el algoritmo son:

1. Conectar con un servidor de cuadros.
2. Crear un servidor de cuadros para enviar los resultados del algoritmo.
3. Configuración desde un archivo de texto, tanto de los parámetros como los datos de conexión del servidor de cuadros y los datos para crear el servidor de cuadros.

Para el visor de resultados la única característica necesaria, además de contar con interfaz gráfica en la que se integren las imágenes recibidas del servidor creado en el módulo de procesado, será permitir elegir el servidor de cuadros al que conectarse, conectarse/desconectarse del mismo y cambiar a otro servidor.

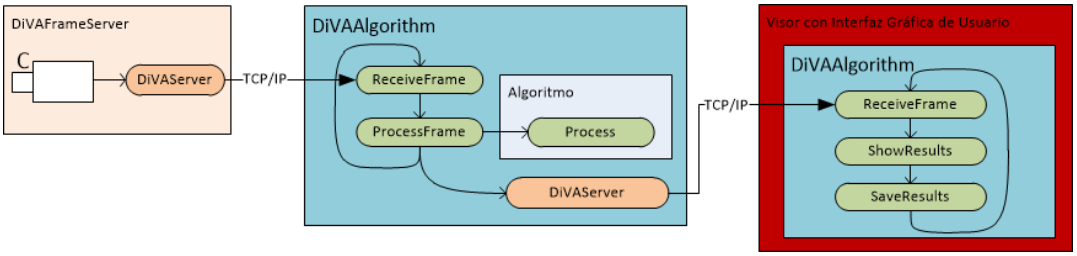

Figura 3.5: Esquema nivel 3.3 de integración.

La limitación de las aplicaciones de este nivel es la posibilidad de cambiar parámetros en tiempo de ejecución que habría que cambiarlos en el equipo que ejecuta el algoritmo utilizando la aplicación de consola. No es posible hacerlo desde la interfaz gráfica del visor.

Capítulo 4

Desarrollo de aplicaciones.

En este capítulo se tratará el proceso a seguir para integrar aplicaciones en la plataforma DiVA pasando por cada uno de los niveles definidos anteriormente (ver sección 3.3). Se detallará de forma genérica para un algoritmo cualquiera para en el capítulo 5 ejemplificar el proceso con dos algoritmos diferentes.

Se documentarán las librerías proporcionadas en el *framework* de DiVA, explicando las clases definidas en ellas y su funcionamiento. También se hará referencia a las plantillas y clases diseñadas para facilitar la implementación de la interfaz gráfica de nuevos algoritmos.

4.1. Guía de integración.

El proceso de integración de algoritmos en DiVA, como se definió en la sección 3.3, debe llevarse a cabo a través de los diferentes niveles definidos, comenzando por el nivel 1 y terminando en las aplicaciones del nivel 3.

4.1.1. Integración nivel 1.

En el primer nivel de integración el objetivo es preparar el algoritmo para su integración, es decir, el resultado deberá ser una clase de C++ que cumpla con los requisitos establecidos. Para ello el primer paso es crear la clase que implementará el algoritmo.

```
1 class ALGORITHM_NAME{
2 public:
3 // Default constructor
4         ALGORITHM_NAME();
5 // Constructors with parameters
```

```
6           ALGORITHM_NAME(param1, param2,...);
7
8   // Destructor
9           ~ALGORITHM_NAME();
10
11          void * process(IplImage *frame);
12          void showresults();
13          void saveResults();
14  // Auxiliar methods
15          void method_aux1(...);
16          void method_aux2(...);
17
18  // Optional methods
19          void set_parameter1(int value);
20          int get_parameter1();
21          void set_parameter2(double value);
22          double get_parameter2();
23
24  // Functions returning result images
25          IplImage * getResult1();
26          IplImage * getResult2();
27
28  private:
29  // Algorithm internal methods
30          void funcion1(...);
31          void funcion2(...);
32
33  // Internal variables
34          int variable1;
35          char *variable2;
36  }
```

- La clase podrá tener tantos constructores como se desee, pero al menos tendrá un constructor por defecto que inicialice los parámetros necesarios para la ejecución del algoritmo con unos valores predefinidos en el código o escritos en un archivo de configuración. Estos reservarán los recursos necesarios para su funcionamiento.
- El destructor se encargará de liberar los recursos que se hubieran reservado durante la ejecución del algoritmo.
- La función `process`, que recibirá como argumento un puntero a una imagen de

OpenCV (estructura `IplImage`), será la función utilizada para procesar cada cuadro, pudiendo esta llamar a otros métodos auxiliares definidos en la clase. Si el método necesita imágenes anteriores a la procesada o posteriores deberá ser la propia clase quién se encargue de almacenarlas. El valor de retorno es un puntero sin formato (`void *`) para dejar libertad a la hora de devolver algún valor, imagen, etc. Esta función obligatoriamente tiene que ser pública porque será invocada desde fuera de la clase. Los resultados totales y parciales obtenidos se guardarán en variables de la clase para que los métodos `showresults` y `saveResults` puedan acceder a ellos.

- El método `showresults` no recibe argumentos ni devuelve ningún valor. Este método internamente se encargará de mostrar por pantalla los resultados de procesar el algoritmo. Deberá mostrar tanto los resultados parciales como el resultado final. Al igual que `process`, tiene que estar declarado como público para que la clase encapsuladora de DiVA pueda referenciarla. La manera de presentar las diferentes imágenes obtenidas durante la ejecución será todas juntas en la misma ventana indicando a que corresponde cada imagen con un título que la identifique. Para la representación de la cuadrícula de imágenes se ha desarrollado en el laboratorio la clase `GridDisplay`.

- El método `saveResults` tampoco recibirá ni devolverá ningún valor. Su función es guardar en un archivo de vídeo los resultados del procesado del algoritmo y en otros archivos los resultados que no sean de vídeo. En el archivo de vídeo se guardarán todas las imágenes resultado dispuestas en una cuadrícula con títulos de igual forma que se mostraban por pantalla en el método `showresults` (utilizando la clase `GridDisplay`). Este método también tiene que implementarse como público.

- Puede haber tantas funciones auxiliares como sea necesario para la ejecución del algoritmo y estas serán preferiblemente privadas porque solo la función `process` hará uso de ellas. En cuanto a los parámetros recibidos y valores devueltos dependerá de la funcionalidad de cada método quedando sin restricciones en este aspecto.

- La clase del algoritmo incluirá todas las variables necesarias para su ejecución, incluyendo posibles declaraciones de objetos de otras clases que ejecuten parte del procesado de la imagen. Estas variables de la clase se definirán como privadas y se implementarán métodos para acceder a ellas desde otras clases.

- Métodos de acceso a las variables. Comúnmente entre las variables definidas en

la clase se encontrarán los parámetros del algoritmo. Al estar definidas como privadas la única forma de acceder a ellas será creando dos métodos públicos que accedan a ella, un método `get_parameter` para la lectura del parámetro y otro método `set_parameter` para modificarlo. La sintaxis de estas funciones queda a decisión del desarrollador, aunque una posible implementación sería que el método `get_parameter` retorne el valor del parámetro elegido y el método `set_parameter` reciba el nuevo valor como argumento.

- Métodos para obtener las imágenes resultado si están declaradas como privadas dentro de la clase. Estas funciones se utilizarán más adelante cuando se necesite mostrar en la interfaz gráfica los resultados.

Una vez creada la clase que implementa el algoritmo el siguiente paso es diseñar el programa principal que ejecutará el algoritmo. El funcionamiento del programa tendrá las siguientes etapas:

1. Inicializa el módulo de obtención de imágenes. Su origen puede ser un archivo de vídeo o una secuencia de imágenes individuales. Una opción es utilizar la clase `VideoCapture` de OpenCV.

2. Inicializar el algoritmo utilizando el constructor de la clase. Si los parámetros del algoritmo que se quieren utilizar no son los establecidos por defecto habrá que utilizar los diferentes constructores con parámetros.

3. Ejecutar en bucle hasta que se terminen las imágenes para procesar la siguiente secuencia:

   a) Capturar una imagen.

   b) Invocar el método `process` del algoritmo pasando como argumento la imagen capturada.

   c) Llamar al método `showresults` para mostrar por pantalla los resultados.

   d) Llamar al método `saveResults` para guardar los resultados en un archivo de vídeo.

4. Una vez finalizado el bucle se deben destruir los módulos inicializados y terminar el programa.

4.1.2. Integración nivel 2.

Para el segundo nivel de integración partimos de una aplicación del nivel 1, es decir, que cumple con las características especificadas para el mismo. Gracias a esto la integración será mucho más sencilla.

El objetivo principal de este nivel es encapsular la clase creada en el nivel 1 dentro de la clase `DiVA_Algorithm` que ya tiene implementada la conectividad con los diferentes módulos de la plataforma. Para realizarlo el proceso a seguir es el que se describe a continuación.

El primer paso es crear una nueva clase que herede de `DiVA_Algorithm`. Esta contendrá un objeto de la clase que implementa el algoritmo y se reescribirán las funciones declaradas como virtuales en la definición de `DiVA_Algortithm`.

```
1        /************************************************/
2        /*To be overwriten*****************************/
3        //Data bases
4        virtual int receiveData(void** data){return 0;};
5        virtual int receiveContentData(void** data){return 0;};
6        virtual int releaseData(void* data){return 0;};
7        virtual int releaseContentData(void* data){return 0;};
8        virtual int dumpData(){return 0;};
9        virtual int dumpContentData(){return 0;};
10       virtual int dumpResultsInFiles(){return 0;};
11       //Image Processing
12       virtual int processFrame(DiVAImage* pImage, void* pdata=
             NULL, void* pContentData=NULL);
13       //Displaying
14       virtual int refreshDisplay(){return 0;};
15       virtual int sendFrameQt(){return 0;};
16       /************************************************/
```

Las funciones `receiveData`, `receiveContentData`, `releaseContentData`, `dumpData` y `dumpContentData` están relacionadas con la conexión con un servidor de datos. En caso de ser necesarias para el funcionamiento del algoritmo deberán ser reimplementadas. La función `dumpResultsInFiles` será la encargada de guardar los resultados de procesar las imágenes en un archivo. `processFrame` será la función que procese los cuadros. Por último `refreshDisplay` y `sendFrameQt` serán las funciones utilizadas para mostrar por pantalla los resultados, la primera de ellas se utilizará en el nivel 2 y la segunda en el nivel 3. Estas últimas cuatro funciones virtuales serán reescritas en todos los algoritmos que se integren en la plataforma

DiVA.

La implementación de los métodos `dumpResultsInFiles`, `processFrame` y `refreshDisplay` será prácticamente igual en todos los algoritmos si la integración de nivel 1 se ha hecho correctamente. La implementación del método `sendFrameQt` se tratará en capítulo 4.1.3 por ser una función que solo se utilizará en el nivel 3.

dumpResultsInFiles llamará a la función `saveResults` definida en la clase que implementa el algoritmo porque esta era la que se encargaba de guardar los resultados en archivos.

refreshDisplay llamará a la función `showResults` definida en la clase que implementa el algoritmo porque esta era la que se encargaba de mostrar por pantalla los resultados.

processFrame tendrá dos funciones principales. La primera será realizar la conversión entre el formato de imagen `DiVAImage` y `IplImage`. Para ello podrá utilizar la clase `OpenCVConverter` incluida dentro de la librería libDiVAbasic. La segunda función será llamar al método `process`, de la clase que implementa el algoritmo, pasando como argumento la imagen ya convertida.

Por último, solo faltaría implementar el constructor o constructores y el destructor de la clase. En ellos habrá que incluir la inicialización de los recursos necesarios y la liberación de los mismos según corresponda.

La inicialización del algoritmo, clase implementada en el nivel 1, se puede hacer bien en el constructor bien dentro del método `processFrame`. La ventaja de hacerlo en `processFrame` es que el módulo de DiVA ya estaría inicializado, ya se habría recibido un primer cuadro y por lo tanto en caso de ser necesario se podría utilizar este primer cuadro para construir el algoritmo, útil si necesita recibir como parámetro una imagen de muestra, el tamaño de las imágenes que tendrá que procesar, etc. En este caso habría que asegurar que esta construcción solo se realiza una vez y no al procesar cada cuadro.

Tras implementar la clase DiVA que encapsula el algoritmo solo queda escribir el programa principal que la use. En primer lugar se construirá el objeto de DiVA con el algoritmo, para ello el constructor recibirá como parámetros la dirección IP y el puerto al que conectar con un servidor de cuadros. Además también recibirá otros parámetros que indican si se utilizarán servidores de datos, si se quiere mostrar por pantalla los resultados, almacenarlos,...

El siguiente paso es llamar a la función `DiVAAlgorithm::init()` que iniciará la conexión con los servidores de cuadros y/o de datos necesarios y crea el directorio en el que se guardarán los resultados. Acto seguido se llamará a la función

`DiVAThread::start()` que pondrá en funcionamiento el hilo en el que se ejecutará el algoritmo y que se encontraba parado inicialmente.

Por último, como el algoritmo se ejecuta en un hilo separado del programa principal habrá que hacer que el programa principal se quede ejecutando un bucle hasta que el usuario decida terminarlo. Se puede aprovechar esto para diseñar una pequeña consola de comandos del programa que se ejecute en paralelo con el algoritmo y nos de opciones de detenerlo, reiniciarlo, modificar sus parámetros y terminar la ejecución.

Una vez terminada la ejecución se detendrá el hilo en el que se ejecuta el algoritmo utilizando la función `DiVAThread::stop()` y luego se eliminará el objeto.

4.1.3. Integración nivel 3.

El objetivo principal de este nivel de integración es dotar a los algoritmos de una interfaz gráfica que proporcione mayor facilidad de uso. De los tres niveles éste es el que deja más libertad al desarrollador. No obstante cada interfaz gráfica tendrá que cumplir una serie de requisitos. Éstos serán diferentes en cada versión de aplicación (desarrollo, cliente o cliente ligera) como se definió en las secciones 3.3.3.1, 3.3.3.2 y 3.3.3.3.

Para implementar las interfaces gráficas de usuario de la plataforma DiVA se ha elegido utilizar el framework Qt (ver sección 2.4). Por esta razón este apartado de la guía tratará de explicar como crear la interfaz utilizando Qt y las plantillas y funciones auxiliares que se entreguen estarán orientadas a la programación con Qt. En el Anexo C se proporciona una introducción a la creación de interfaces y también se definen algunos de los conceptos utilizados en esta sección.

La interfaz gráfica se creará en una nueva clase que herede de la clase `QMainWindow`. Este proceso se puede hacer de forma automática al crear un proyecto de Qt si está instalado en el ordenador el *Add-in* de Qt para Visual Studio. En las ventanas de creación del proyecto solo habrá que indicar los módulos de Qt que se quieren utilizar y el nombre que tendrá la clase nueva. Al finalizar, el asistente crea todos los archivos necesarios del proyecto con el código mínimo para que funcione.

En esta nueva clase habrá que introducir toda la funcionalidad relacionada con la interfaz gráfica. También habrá que incluir un objeto de la clase `DiVAAlgorithm`. En el caso de utilizar ventanas auxiliares también habrá que declararlas dentro de esta clase.

El funcionamiento esquemático de la interfaz es el que se explica a continuación. En un hilo se ejecuta la ventana encargada de mostrar las imágenes procesadas y en otro hilo se ejecuta el algoritmo que procesa las imágenes. La comunicación entre ambas partes se lleva a cabo utilizando el sistema de `signals` y `slots` de Qt. Entre

las funciones virtuales de `DiVAAlgorithm` se dejó sin redefinir `sendFrameQt` que será la encargada de establecer la comunicación.

La función `sendFrameQt`, que se ejecuta cada vez que se procesa un cuadro, emitirá una `signal` con todas las imágenes resultado del algoritmo (finales y parciales). Esta `signal` estará conectada con un `slot` definido en la clase de la ventana. Al recibir la `signal` se ejecutará el `slot` encargado de actualizar las imágenes en la ventana. Como para emitir una `signal` es necesario que esta esté definida en un objeto de Qt se ha creado la clase `DiVA_QtSignals` (ver sección 4.2.2) en la que habrá que declarar las diferentes `signals` necesarias para el funcionamiento del programa.

La interfaz gráfica por cómo se definió debe permitir elegir con que servidor de cuadros se quiere conectar la aplicación. Para ello se utilizará la plantilla proporcionada "ventana_camaras" en la que no habrá que introducir ningún cambio. Los detalles de la clase que implementa esta ventana y como utilizarla se describirán en la sección 4.2.5.

También existirá una ventana con un menú de ayuda en la que se explique el funcionamiento de la aplicación, los diferentes parámetros que se puedan configurar, etc. Para esta ventana también se proporciona una plantilla que se explicará en la sección 4.2.7.

Habrá una ventana "Acerca de..." en la que se incluirá la información del desarrollador del algoritmo y de la aplicación, la versión en la que se encuentra, etc.

La forma de acceder a estas ventanas puede ser bien desde botones en la interfaz, desde un menú o desde iconos en una barra de herramientas. Las aplicaciones también contarán con botones que permitan iniciar y detener el algoritmo.

4.1.3.1. Aplicación de desarrollo.

Por cómo se han definido las aplicaciones de desarrollo, la ventana principal estará dividida en varias zonas. En una zona se encontrarán los botones de control agrupados. En otra zona se encontrarán los parámetros que se pueden cambiar. En otra zona se encontrarán los elementos que permitan seleccionar qué resultados parciales se quieren mostrar y cuáles no. Y finalmente estarán la zona en la que se muestren las imágenes resultado y la zona reservada para los logos de las entidades involucradas (ver figura 4.1).

Para facilitar la programación de la zona encargada de representar los resultados se ha creado la clase `DiVA_QtGridDisplay` que como se verá más detalladamente en la sección 4.2.3 nos permitirá añadir tantas imágenes como deseemos con su título sin afectar al tamaño de la ventana.

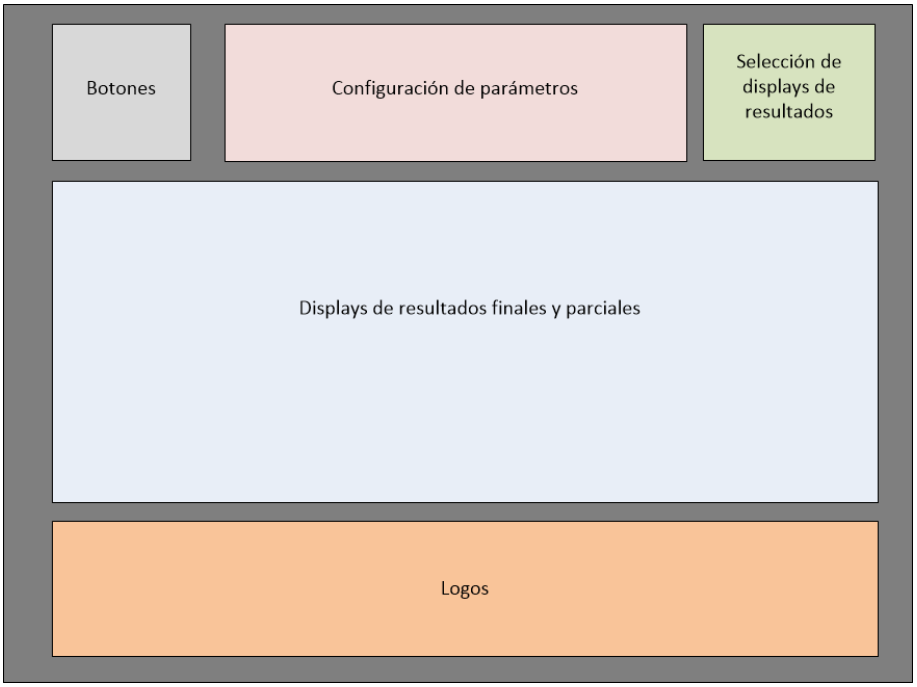

Figura 4.1: *Layout* esquemático desarrollo.

4.1.3.2. Aplicación de cliente.

En esta aplicación la interfaz se simplifica de modo que solo se pueden diferenciar tres zonas. Una primera zona en la que se muestra el resultado final de procesar la imagen, otra en la que se encuentran los botones de iniciar, detener y los que sirvan para abrir otras ventanas como la de selección de cámara y otra para colocar los logos (ver figura 4.2).

En esta aplicación la modificación de parámetros no se encuentra en la ventana principal si no que está en una ventana independiente. Para esta ventana se proporciona la plantilla "ventana_parametros" (ver sección 4.2.6).

Para estas aplicaciones se define un "modo compacto" que consiste en modificar la ventana para que solo se muestre el resultado de procesar el algoritmo y en la que quedarían ocultos los menús, las barras de herramientas y los botones, en definitiva todo lo que no sea la imagen resultado.

4.1.3.3. Aplicación de cliente ligera.

Esta aplicación realmente se compone de dos ejecutables diferentes. Por un lado está el programa que ejecuta el algoritmo y por otro el que lo visualiza. Para la comunicación entre ambos módulos se establece una arquitectura cliente-servidor en

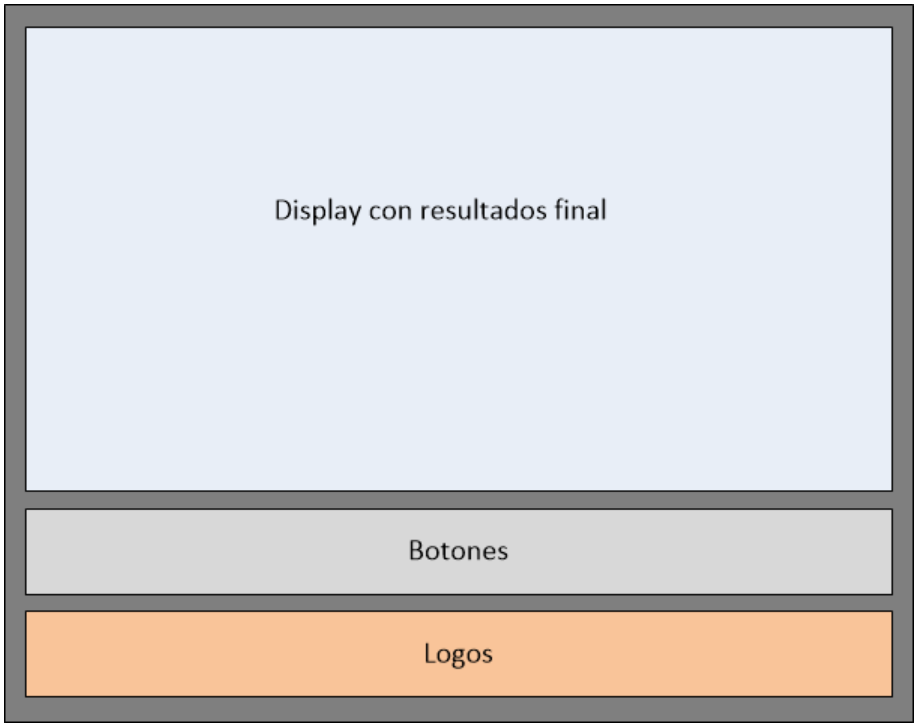

Figura 4.2: *Layout* esquemático cliente.

la que el visor de algoritmos actúa como cliente.

El programa que ejecuta el algoritmo no tiene interfaz gráfica y la única diferencia que tiene con las aplicaciones del nivel 2 es que implementa un servidor de imágenes. Este servidor de imágenes se implementa como cualquier otro de los existentes en DiVA. Su misión será servir las imágenes ya procesadas para que otras aplicaciones se puedan conectar a él y recibirlas.

Para incluir el servidor en el programa se parte de la clase de DiVA en el nivel 2. El primer paso es añadir a la clase un objeto de la clase `DiVAImageBuffer`, otro de la clase `DiVAServer` y una variable de tipo `long`.

```
1         DiVAImageBuffer * _buffer;
2         DiVAServer * _server;
3         long _imageid;
```

`_buffer` es el *`buffer`* en el que se irán guardando las imágenes procesadas y de donde las tomará el servidor.

`_server` objeto que implementa el servidor de imágenes.

`_imageid` es el identificador que se dará a cada imagen que inicialmente valdrá 0.

El siguiente paso es construir los objetos `_buffer` y `_server`, se hará después de construir la clase del nivel 1 que implementaba el algoritmo. Para el *buffer* es necesario indicar la longitud del *buffer*, una imagen de muestra (o su tamaño, número de canales y tipo), un nombre que lo identifique y el tiempo de espera. Para construir el servidor se pasará como argumento el *buffer* que se ha creado antes y el puerto en el que dará servicio. Una vez creado, se establecerá el formato de codificación que se utilizará en la transmisión (ver sección 3.2.1) con la función `DiVAServer::setTypeCodification`. Finalmente para iniciar el funcionamiento del servidor se utilizará la función `DiVAServer::start`.

Solo falta añadir las imágenes procesadas al *buffer* para que el servidor las pueda transmitir. Para llevar a cabo esta tarea el primer paso es obtener una imagen en formato `DiVAImage` con el resultado final. A esta se le pone el identificador de imagen con la función `DiVAImage::setId` y se incrementa el valor guardado de `_imageid`. Finalmente utilizando la función `DiVAImageBuffer::put` se introduce la imagen en el *buffer*.

En este punto ya tenemos la primera aplicación del nivel con el algoritmo procesando las imágenes que recibe de un servidor de cuadros y enviando los resultados a otros clientes que se conecten.

La segunda aplicación del nivel es simplemente un visor, se implementa como un algoritmo que se conecta con un servidor de cuadros y que no hace ningún procesado sobre la imagen. El nivel de interacción con el algoritmo (que se ejecuta en la primera aplicación del nivel) es mínimo por lo que no se podrán modificar los parámetros ni iniciarlo o detenerlo desde la aplicación del visor. Una vez programado un visor puede ser común para más de un algoritmo. Entre las clases y ventanas proporcionadas como plantillas se encuentra un visor de algoritmos genérico ya implementado que podrá usarse directamente.

Si se decide crear un visor nuevo éstos serán los aspectos a tener en cuenta en su diseño. En la ventana principal solo se podrán distinguir tres zonas, una en la que se muestre la imagen resultado, otra con los botones para iniciar y detener la conexión con el servidor de cuadros y una zona para los logos. La única funcionalidad añadida es la de poder conectar con las cámaras, para ella se puede utilizar la plantilla `ventana_camaras` (sección 4.2.5). También contará con los menús de "Ayuda" y "Acerca de..." (sección 4.2.7 y 4.2.8). El visor al igual que la aplicación de cliente tendrá el modo compacto en el que se ocultarán todos los botones, iconos, etc. para dejar solo la imagen resultado en la ventana.

Bajo la interfaz gráfica habrá un algoritmo vacío, es decir, se creará una clase

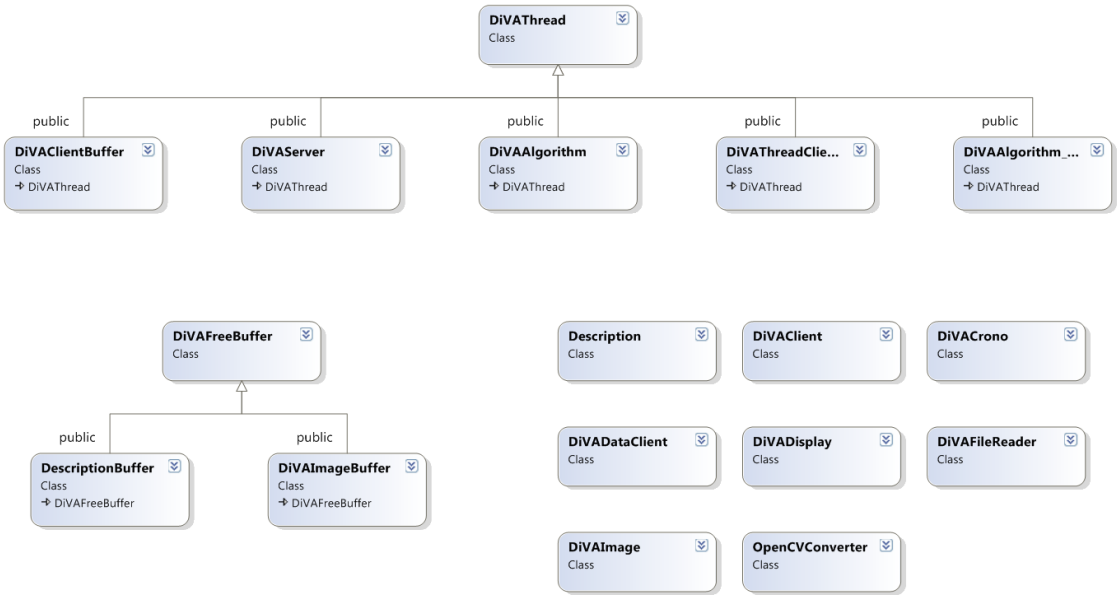

Figura 4.3: Diagrama de clases de libDiVABasic.

derivada de `DiVAAlgorithm` como se hacía en el nivel 2 (sección 4.1.2) y en el método `processFrame` simplemente se guardará la imagen que se recibe para luego en la función `sendFrameQt` emitir la señal con la imagen guardado. Una vez creada esta clase se añadirá la interfaz gráfica como se hizo en el nivel 3.1 (sección 4.1.3.2).

4.2. Librerías y plantillas proporcionadas.

4.2.1. libDiVABasic.

libDiVABasic es la librería principal de la plataforma DiVA. En ella se implementan las clases básicas de las que luego heredarán el resto de clases. En la figura 4.3 se puede observar un diagrama con las clases de la librería DiVA y su jerarquía. A continuación se dará una breve descripción de las clases más importantes.

`DiVAThread` esta clase implementa las funciones necesarias para que las clases que hereden de ella se ejecuten en un hilo independiente, para ello utiliza MFC (Microsoft Fundation Classes). Esta clase tiene tres métodos virtuales que serán implementados en las clases que hereden de ella. `initResources()` que será la función encargada de iniciar los recursos necesarios para la ejecución del hilo, `process()` función que se ejecutará en bucle mientras esté activo

el hilo y `releaseResources()` que liberará los recursos del sistema. Para iniciar el funcionamiento del hilo habrá que invocar al método `start()` y para detener el hilo al método `stop()`. Las clases que heredan de esta son `DiVAClientBuffer`, `DiVAServer`, `DiVAThreadClient`, `DiVAAlgorithm` y `DiVAAlgorithm_multicam`.

DiVAServer esta clase implementa la parte del servidor en la arquitectura cliente-servidor en el sistema DiVA. Hereda de `DiVAThread` por lo que se ejecutará en un hilo independiente. Para atender las peticiones de cada cliente, el servidor creará un nuevo hilo.

DiVAThreadClientAttention esta clase implementa un hilo independiente encargado de atender las peticiones de cada cliente que haya conectado a un servidor.

DiVAAlgorithm esta clase es la encargada de encapsular los algoritmos para introducirlos en la plataforma DiVA. Permite ejecutar los algoritmos en un hilo independiente al del programa principal. Cuando está en funcionamiento se encarga de conectar con los servidores de cuadros y de datos, de solicitar los datos, ejecutar el método encargado de procesar las imágenes resultado y mostrarlas por pantalla o guardarlas en archivos.

DiVAAlgorithm_multicam esta clase y `DiVAAlgorithm` cumplen la misma función pero con más de una cámara. Se encarga de recibir las imágenes secuencialmente de cada uno de los servidores para luego procesarlas y mostrar los resultados.

DiVAFreeBuffer esta clase define un *buffer* genérico de forma abstracta y los métodos que deberán implementarse en los diferentes *buffers* que se incluyan en el sistema. Los *buffers* se pueden utilizar para el almacenamiento de los datos del sistema, como pueden ser las imágenes capturadas por el módulo capturador. Dos clases heredan de esta.

DescriptionBuffer esta clase adapta al formato de descriptores de DiVA los métodos definidos en `DiVAFreeBuffer` .

DiVAImageBuffer esta clase adapta a imágenes en formato `DiVAImage` las funciones definidas en `DiVAFreeBuffer`.

Description esta clase define los descriptores utilizados en DiVA

DiVAClient esta clase implementa el cliente que se conectará a los servidores de la plataforma. A diferencia de `DiVAThreadClient` este se ejecutará en el mismo hilo del programa principal.

DiVACrono esta clase implementa el mecanismo necesario para medir tiempos dentro del sistema.

DiVADisplay esta clase permite mostrar por pantalla los resultados de ejecutar el algoritmo en una cuadricula con títulos.

DiVAFileReader esta clase implementa un lector de archivos que permite leer archivos de texto con el formato adecuado. Para que un objeto de esta clase pueda sacar la información del archivo este debe escribirse teniendo en cuenta lo siguiente:

- La sintaxis para las líneas de información es: `NOMBRE=valor`, dónde `NOMBRE` es el nombre identificativo del parámetro y `VALOR` es su valor.
- Las líneas en blanco no son tenidas en cuenta.
- Los comentarios se escriben en líneas que comienzan por dos barras (`//`) y que no son tenidas en cuenta.

DiVAImage es la clase que implementa el formato de imágenes del sistema. Está construida en torno al formato de imagen definido en OpenCV: `IplImage`. Cuenta con métodos que permiten operar con la imagen (cambiar el valor de un pixel, reescalar, ...). Cada imagen tendrá un identificador `idImage` que será asignado por la fuente de captura de la imagen y una marca de tiempo `timeStamp` indicando el momento de captura.

OpenCVConverter esta clase implementa un conversor de formato entre OpenCV y DiVA. Permite convertir una imagen en formato `IplImage` en `DiVAImage` y viceversa.

4.2.2. DiVA_QtSignals.

Esta clase implementa un objeto que contiene las diferentes `signals` necesarias para establecer la comunicación entre DiVA y Qt. La clase deriva de `QObject` porque para definir `signals` es necesario que se haga dentro de un objeto de Qt, de esta forma el precompilador de Qt las podrá identificar. Dentro de la clase solo se encuentra la declaración de las `signals` porque no necesitan ser definidas.

En el archivo que se proporciona con la librería ya existen algunas `signals`. Si ninguna de ellas encajara con las necesarias para la ejecución del algoritmo se podrá declarar una nueva que si lo haga. La forma de hacerlo es la siguiente.

```
signals:
        void signal_name(...);
```

Las `signals` se pueden utilizar para enviar variables, solo habría que añadirlas como argumentos en la declaración.

4.2.3. DiVA_QtGridDisplay.

Esta clase implementa una cuadrícula de imágenes. Hereda de la clase `QWidget` existente en la librería Qt. Se puede añadir directamente en el diseño de la interfaz gráfica como cualquier otro objeto de Qt. Al utilizar esta clase se consigue tener todas las imágenes dispuestas en una cuadrícula que ajusta el tamaño de las imágenes para ocupar el máximo espacio disponible y las reescala con cambios en el tamaño de la ventana. Cada imagen de la cuadrícula tendrá un título que la identifique.

Los métodos disponibles en la clase (además de los heredados) son los siguientes:

addDisplay sirve para añadir un nuevo *display* a la lista de los ya existentes. Recibe como argumento el título que tendrá el *display*. Devuelve la posición en la que se ha añadido.

deleteDisplay elimina de la lista el *display* cuyo título coincide con la cadena que recibe como argumento.

setTitle modifica el título del *display* en la posición que se le indica.

setDisPos función interna que coloca los *displays* y sus títulos en el espacio disponible. Recibe como argumento el tamaño que tendrá cada *display*.

setPreferredSize se utiliza para indicar a la clase cual es el tamaño ideal para mostrar las imágenes. Recibe el ancho y el alto deseado.

calcFrameSize función interna que calcula el tamaño y el número de columnas y filas necesarios para mostrar todas las imágenes utilizando como criterio maximizar el tamaño de las mismas. Tiene en cuenta el tamaño disponible y el tamaño de imagen deseado que se haya indicado con la función `setPreferredSize`.

paintFrame recibe como argumento una imagen y su título para pintarla en el *display* que tiene ese título.

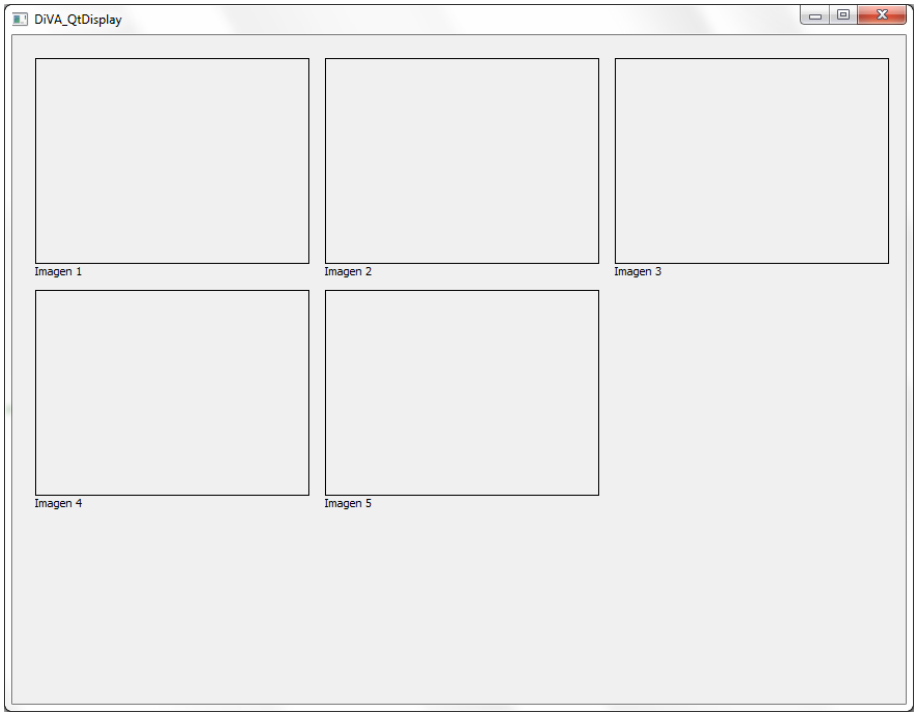

Figura 4.4: Imagen ejemplo `DiVA_QtGridDisplay`.

`resizeEvent` método heredado de `QWidget` extendido para que al cambiar de tamaño se llame a los métodos `calcFrameSize` y `setDisPos`.

4.2.4. Visor de algoritmos.

La aplicación propuesta en este apartado corresponde con la aplicación definida en el nivel 3.3. Concretamente con la que implementa la interfaz gráfica que conecta con el servidor de cuadros del algoritmo. Este ejemplo puede usarse directamente o utilizarse como plantilla para la creación de un nuevo visor.

Se puede observar (figura 5.16) que cumple con las características definidas en la sección 4.1.3.3. Tiene una zona en la que se mostrará la imagen y otra con los botones necesarios para iniciar, detener y abrir la ventana de selección de cámara. A estas funciones se puede acceder también desde los iconos de la barra de herramientas o desde los menús. Desde la barra de menús también se puede abrir la ventana de "Ayuda" y la de "Acerca de...".

Figura 4.5: Visor algoritmos.

4.2.5. Ventana cámaras.

El menú de selección de cámaras está implementado en esta ventana. Se compone de dos clases diferentes, una para la interfaz gráfica (ventana_camaras) y otra para tener una lista de cámaras (`cam_list`). Para utilizar la ventana hay que incluir en el proyecto los archivos correspondientes a la interfaz: "`ventana_camaras.h`" y "`ventana_camaras.cpp`" y los archivos de la lista de cámaras: "`camaras.h`" y "`camaras.cpp`". Luego simplemente hay que crear un objeto `ventana_camaras` y llamar a la función `ventana_camaras::show` para ejecutar la ventana.

La clase `cam_list` almacena las diferentes cámaras definidas en un vector de elementos del tipo `camera`. Este tipo se define como una estructura con los siguientes elementos:

```
1 struct camera{
2         int port;
3         char ip[50];
4         char name[255];
5         int new;
```

port almacena el puerto de servicio.

ip almacena la dirección ip de la cámara en formato xxx.xxx.xxx.xxx.

name almacena un nombre descriptivo de la cámara que permita identificarla al usuario.

new es una variable que sirve para distinguir las cámaras que se han añadido nuevas desde la ultima actualización del archivo de cámaras.

La lista de cámaras existente se almacena en un archivo de configuración con el siguiente formato:

```
NAME=camera name
IP=camera ip address
PORT=camera port
//coments
```

Al construir un elemento de esta clase si se utiliza el constructor por defecto leerá la información de las cámaras ya definidas del archivo cam.conf que se encuentra en el directorio ./config. Si el nombre del archivo fuera diferente se utilizará el constructor que recibe como argumento el nombre de dicho archivo. Al destruir el objeto se actualizará la información del archivo de cámaras especificado.

Los métodos pertenecientes a esta clase son los siguientes:

updateFileCameras añade al archivo de cámaras la información de las cámaras marcadas como nuevas (variable `new` de la estructura `camera`).

readFileCameras lee el archivo de cámaras y guarda la información en elementos de tipo `camera`.

addCamera añade una cámara a la lista. Recibe como argumento la ip, el puerto y el nombre de la cámara.

delCamera elimina de la lista la cámara cuyo nombre coincide con el que ha recibido como argumento.

selectDefaultCam permite elegir cual es la cámara por defecto. La cámara por defecto es la primera de la lista de cámaras, esta función pone la primera en la lista la cámara seleccionada.

writeFileCameras reescribe el archivo de información de cámaras con todos los elementos de la lista. Esta función se utiliza al eliminar una cámara o al establecer una por defecto.

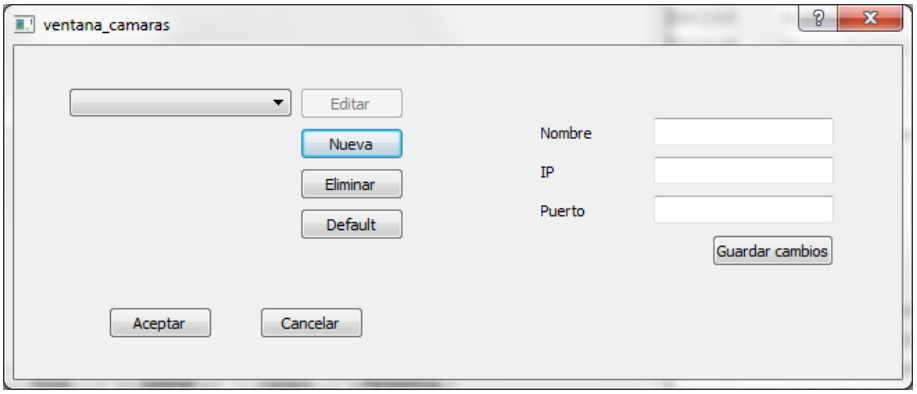

Figura 4.6: Imagen ventana cámaras.

La clase `ventana_camaras` hace de interfaz gráfica entre el usuario y la clase `cam_list`. Permite elegir cómodamente de una lista desplegable la cámara con la que queremos conectar, añadir una cámara nueva a la lista, modificar una ya existente o eliminarla y elegir cuál es la cámara por defecto. Una vez elegida la cámara y pulsado el botón de aceptar se emite una `signal` que contiene la información de la cámara.

En la imágen 4.6 se observa la disposición de los distintos elementos de la ventana. En primer lugar hay un menú desplegable con la lista de las cámaras. Hay un botón para añadir nueva cámara, otro para editar la cámara seleccionada, otro para eliminarla y otro para establecerla como cámara por defecto. Al pulsar el botón "Editar" o el botón "Nueva" se extiende la ventana y aparecen las opciones para añadir/editar la información de la cámara. Esta extensión cuenta con una línea de texto para escribir el nombre de la cámara, otra para la dirección IP, otra para el puerto y un botón para guardar los cambios realizados. Una vez pulsado el botón "Aceptar" la ventana se cierra y emite una `signal` con la información de la cámara seleccionada.

4.2.6. Ventana propiedades.

Esta ventana que se proporciona como ejemplo tiene la finalidad de mostrar cómo se utilizan los diferentes métodos de entrada de Qt. Implementa la configuración de parámetros del algoritmo Detección de Regiones Estáticas (ver sección 5.2).

Para nuevas implementaciones se podrá utilizar como plantilla y aprovechar su estructura pero habrá que cambiar los parámetros que se pueden configurar (que serán diferentes dependiendo del algoritmo). Hay que tener en cuenta que la ventana tiene un método llamado `setParam` que debe recibir los parámetros actuales del algoritmo para inicializar los métodos de entrada con el valor actual. También se debe modificar

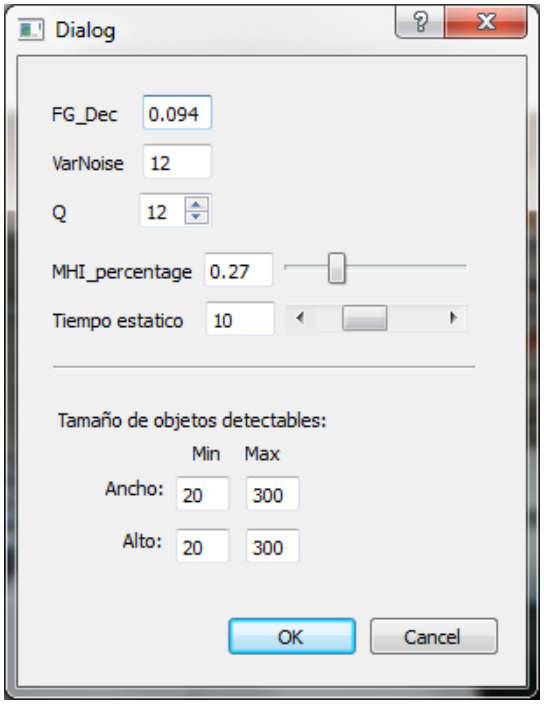

Figura 4.7: Ejemplo ventana de propiedades proporcionada.

la `signal` que se emite al pulsar el botón "Ok". En ella se envían todos los valores de los parámetros nuevos. Esta se conectará a un `slot` de la ventana principal que haga la configuración de los nuevos parámetros.

4.2.7. Ventana "Ayuda".

Esta ventana implementa el sistema de ayuda de la aplicación. Debe incluir una pequeña descripción de los parámetros que se pueden configurar en el algoritmo y una guía de cómo utilizar el algoritmo. En la figura 4.8 se muestra un ejemplo simple de cómo podría ser la ventana de ayuda de un algoritmo.

4.2.8. Ventana "Acerca de...".

La ventana acerca de es una ventana de información acerca del algoritmo. Contendrá información sobre el desarrollador del algoritmo, sobre el desarrollador de la aplicación, versión en la que se encuentra la aplicación, fecha de desarrollo e información de contacto.

La figura 4.9 muestra un ejemplo de implementación de dicha ventana.

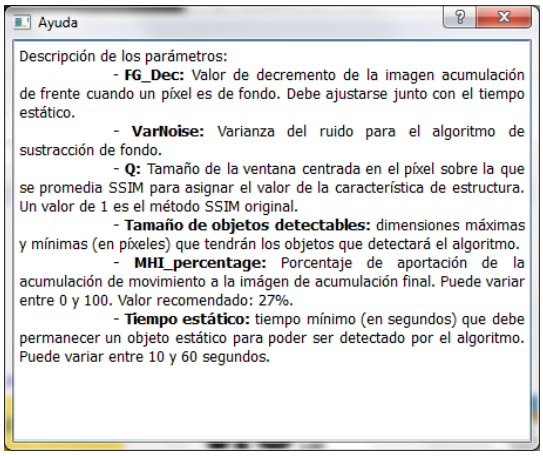

Figura 4.8: Ejemplo ventana "Ayuda" proporcionada.

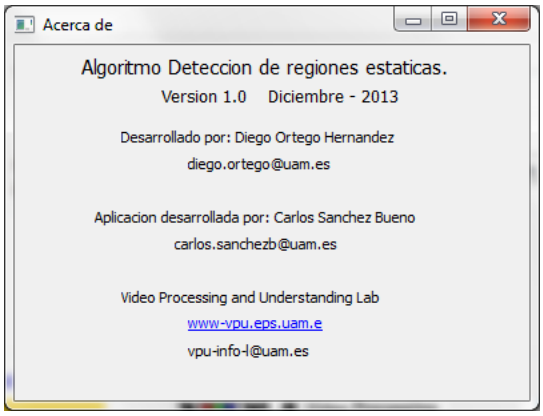

Figura 4.9: Ejemplo ventana "Acerca de..." proporcionada.

Capítulo 5

Aplicaciones desarrolladas y resultados.

Durante la realización del proyecto se han desarrollado dos aplicaciones que integran un algoritmo cada una en la plataforma DiVA. Cada aplicación se ha integrado siguiendo metodología de tres niveles definida en el capítulo 3.3 y desarrollada en el capítulo 4.

El proceso de desarrollo de las aplicaciones y la creación de la metodología ha sido un proceso simultáneo en el que primero se plantearon los niveles, y después durante el desarrollo de las aplicaciones se fueron encontrando fallos en la definición de los mismos que se fueron corrigiendo conforme se avanzaba con la programación hasta completar la metodología de integración descrita en este documento.

La primera aplicación desarrollada integra el algoritmo de substracción de fondo *Gaussian Mixture Model Background Subtraction* [5, 6] disponible en la librería OpenCV. Esta aplicación fue el punto de partida en el desarrollo de la metodología propuesta y la primera toma de contacto con la integración de algoritmos en la plataforma DiVA y la creación de interfaces gráficas utilizando Qt.

La segunda aplicación integra el algoritmo Detección de Regiones Estáticas [7] desarrollado en el VPULab por Diego Ortego.

5.1. Extracción de fondo.

La extracción del fondo en una secuencia de vídeo es comúnmente una de las primeras etapas presentes en el procesado. Es por ello que se han desarrollado innumerables técnicas para lograrlo. Para el desarrollo de la primera aplicación se utilizado el algoritmo *Gaussian Mixture Model Background Subtraction* [5, 6].

Este algoritmo de substracción de fondo trabaja a nivel de píxel y modela el conjunto de valores que puede tomar cada píxel de la imagen mediante una mezcla de M funciones de densidad de probabilidad gausianas. El número de gausianas utilizadas es decidido automáticamente por el algoritmo adaptándose al tipo de escena sobre el que se ejecuta. El algoritmo almacena estas gausianas en una ventana temporal, pudiendo adaptar el modelo de fondo a nuevos elementos presentes en él o a cambios severos de iluminación.

5.1.1. Nivel 1.

La aplicación desarrollada en este nivel permite analizar un archivo de vídeo o capturar imágenes de una cámara conectada al equipo que ejecuta el algoritmo. La salida del algoritmo es tanto por pantalla como en un archivo de vídeo. Por pantalla se muestra la máscara de frente obtenida en una ventana y en otra ventana, independiente de la primera, se muestra la imagen de entrada con la máscara superpuesta en color azul. En el archivo de vídeo generado se almacena únicamente el contenido de la segunda ventana (imagen de entrada con la máscara superpuesta).

Como resultado de este desarrollo se ha obtenido la clase "`BKGS_MOG2`" que implementa el algoritmo de substracción de fondo *Gaussian Mixture Model Background Subtraction* y que cumple con las especificaciones necesarias para su posterior integración en la plataforma DiVA. Esta clase no implementa la captura de cuadros, ya que esta no será necesaria al integrar el algoritmo en DiVA.

5.1.2. Nivel 2.

En este nivel se integra la clase desarrollada en el nivel anterior dentro de la plataforma DiVA, obteniendo una aplicación capaz de comunicarse con los distintos módulos disponibles en la plataforma. Ahora la fuente de imágenes será un servidor de cuadros de la plataforma, es decir, se podrán obtener imágenes de cualquier medio que esté integrado en la plataforma (cámaras y vídeo). El programa tiene la misma salida que en el nivel anterior, un vídeo en el que se ve la imagen de entrada con la máscara de frente superpuesta y salida por pantalla con esa imagen en una ventana y otra ventana con la máscara en blanco y negro (ver figura 5.1).

Los datos del servidor de cuadros (dirección IP y puerto) se extraen de un archivo de configuración. En este archivo también se puede elegir si queremos que los resultados de procesar el algoritmo se muestren por pantalla, se guarden en un fichero de vídeo o ambas. El archivo de configuración tiene el siguiente formato:

1. Cada parámetro va en una línea con la siguiente sintaxis: `NOMBRE=valor`

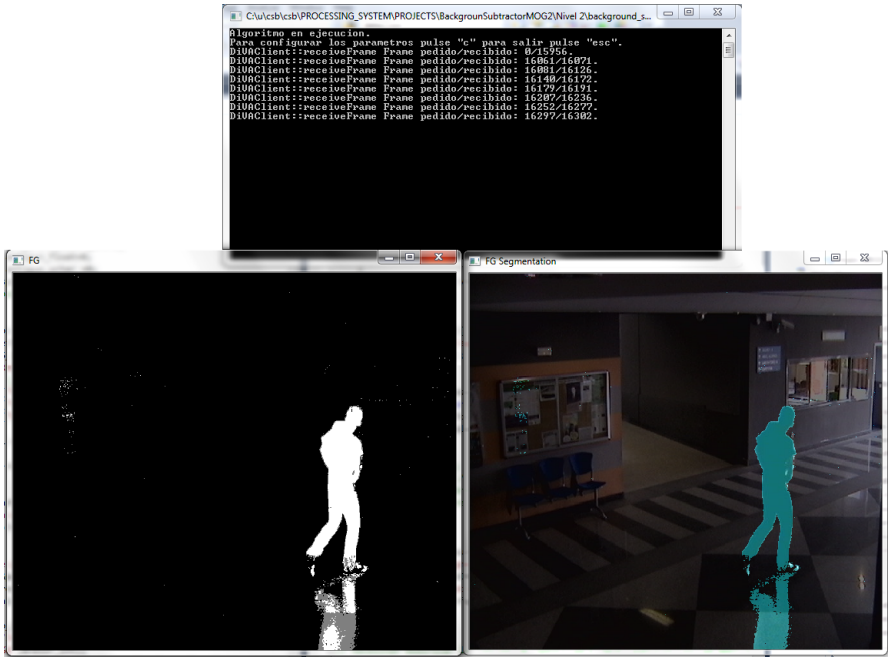

Figura 5.1: Imagen salida.

2. Las líneas en blanco son ignoradas.

3. Las líneas que comienzan con // se toman como comentarios y se ignoran.

Para el desarrollo del programa se ha encapsulado la clase "`BKGS_MOG2`" creada en el nivel anterior dentro de la clase "`DIVA_BKGS_MOG2`" que hereda de la clase encapsuladora disponible en la plataforma DiVA: `DiVAAlgorithm`.

La principal diferencia entre este nivel y el anterior es la intercomunicación con los diferentes módulos de la plataforma DiVA (principalmente servidores de cuadros). Esto permite que la fuente de imágenes sea cualquier cámara de las disponibles en el sistema o cualquier vídeo que sea capaz de leer la clase capturadora de DiVA diseñada para archivos de vídeo.

Aprovechando que la plataforma DiVA ejecuta la clase `DiVAAlgorithm` en un hilo diferente al hilo principal del programa, se ha creado una pequeña consola de comandos que permite cambiar, en tiempo de ejecución, los diferentes parámetros del algoritmo. Para ello basta con pulsar la tecla "c" introducir el número correspondiente al parámetro que se quiere configurar e introducir el nuevo valor (ver figura 5.2). También permite finalizar el algoritmo pulsando la tecla "Esc".

Como resultado de este nivel se ha obtenido una clase perteneciente a la plataforma DiVA que integra el algoritmo. Esta clase se utilizará en niveles posteriores para dotar de interfaz gráfica a la aplicación.

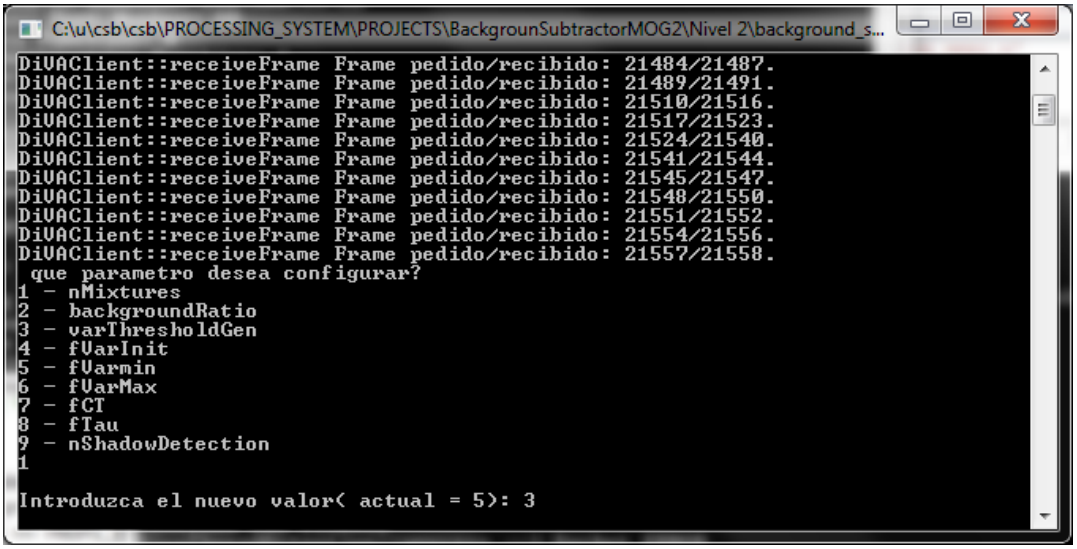

Figura 5.2: Imagen consola parámetros.

5.1.3. Nivel 3.1.

En este nivel se buscaba diseñar una interfaz gráfica para la aplicación destinada a desarrolladores. Siguiendo las pautas de diseño especificadas en los capítulos anteriores se ha obtenido una interfaz gráfica con las siguientes características.

Se muestran en la misma ventana los resultados de procesado del algoritmo, la información de conexión con los servidores de cuadros y un grupo de botones para iniciar la conexión y detenerla, el conjunto de parámetros modificables con diferentes métodos de entrada y un botón para aplicar los cambios.

Al iniciar la aplicación en la ventana principal (figura 5.3) aparece el hueco reservado para mostrar las imágenes en gris, los campos donde irán la dirección y el puerto del servidor de cuadros, un botón para iniciar el algoritmo, un botón para detenerlo, los métodos de entrada para los parámetros, un botón para aplicar los cambios en los parámetros y otro botón para hacer zoom en una zona de la imagen.

Una vez introducida la dirección IP y el puerto del servidor, al pulsar el botón "Iniciar" arranca el funcionamiento del algoritmo. Empiezan a mostrarse los resultados de procesar las imágenes que llegan desde el servidor, a la derecha se muestra la máscara de frente obtenida y a la izquierda se muestra la imagen de entrada con la máscara superpuesta en color azul. Se desactiva el botón "Iniciar", al no ser ya necesario, y se activa el botón "Deneter". Aparece también el valor actual de cada uno de los parámetros configurables y se activan los botones "Aplicar" y "Zoom selección". Para modificar el valor de uno o más parámetros bastará con introducir el nuevo valor y pulsar el botón "Aplicar". Para hacer zoom en alguna región de la imagen, se

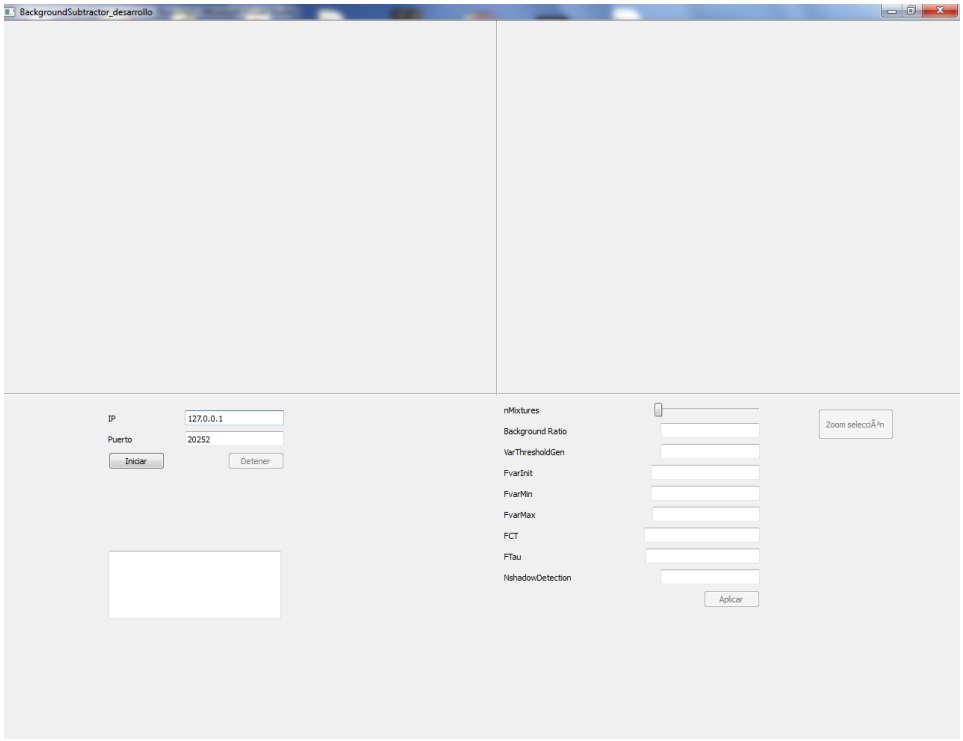

Figura 5.3: Ventana inicial.

debe seleccionar esta en una de las dos imágenes resultado mostradas en la ventana principal utilizando el método pincha y arrastra que irá dibujando un recuadro rojo con la zona seleccionada. Una vez seleccionada una región al pulsar el botón "Zoom selección" se abrirá una nueva ventana auxiliar con la zona seleccionada al doble de tamaño, se muestran la máscara como la imagen con la máscara superpuesta. Para eliminar el recuadro hay que pulsar el botón izquierdo del ratón sobre una de las dos imágenes resultado en la ventana principal.

Al pulsar el botón de "Detener", la aplicación deja de ejecutar el algoritmo y vuelve al estado inicial con la diferencia que no se borran los datos de los parámetros ni las últimas imágenes mostradas en la ventana principal. De este modo es posible volver a iniciar el algoritmo con la misma cámara o con otra.

5.2. Detección de regiones estáticas.

Para la segunda aplicación se ha elegido el algoritmo de Detección de Regiones Estáticas[7] desarrollado dentro del VPULab. Este algoritmo es capaz de extraer en una secuencia de vídeo las regiones que permanecen estáticas durante un periodo de tiempo preestablecido. Para lograrlo el algoritmo mezcla diferentes características que

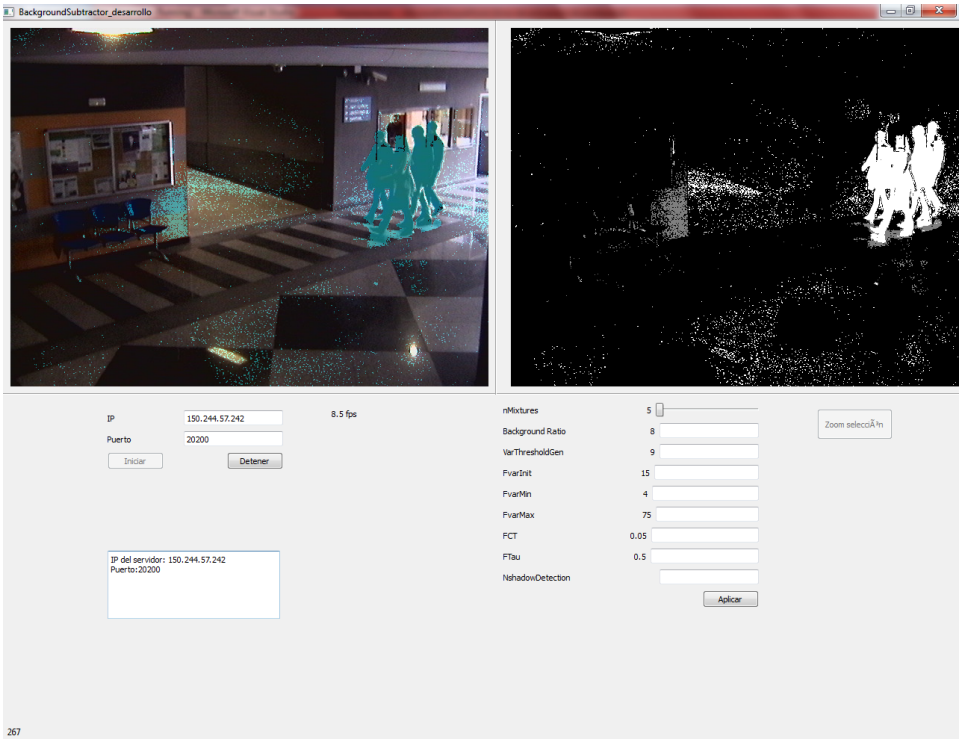

Figura 5.4: Imagen algoritmo funcionando.

lo hacen más robusto frente a efectos no deseados como pueden ser pequeños cambios de iluminación, oclusiones,...

El algoritmo utiliza tres características: similaridad estructural, substracción de fondo y movimiento. La característica de movimiento se obtiene utilizando filtros de medianas, eliminando de esta forma los movimientos constantes. La característica de similaridad estructural permite filtrar los cambios de iluminación. Por último se realiza una sustracción de fondo utilizando el algoritmo Gamma. Con estas características se generan tres imágenes de puntuación que finalmente se combinan para obtener una imagen en escala de grises que tras umbralizarla nos da como resultado la imagen de regiones estáticas.

5.2.1. Nivel 1.

Para este nivel, se ha desarrollado una aplicación de consola que ejecuta el algoritmo de detección de regiones estáticas. Recibe como argumento el nombre del archivo con la fuente de vídeo, el cual será leído y procesado. Como resultado de procesar las imágenes, el programa muestra por pantalla y almacena en un fichero de vídeo una cuadrícula que contiene nueve elementos: la imagen de entrada con el número

de cuadro procesado y ocho imágenes de resultados (máscaras de frente, imágenes de puntuación, máscara de regiones estáticas, ...). En la cuadrícula cada imagen se muestra con un título que permite identificarlas. En la consola, donde se ejecuta la aplicación, también se muestra el número de imagen y el tiempo que se ha tardado en procesarla.

En esta aplicación no es posible modificar los parámetros, estos tendrán unos valores establecidos por defecto.

Como resultado de este nivel se ha adaptado la clase ya existente que implementaba el algoritmo de detección de regiones estáticas al formato necesario para la integración en DiVA cumpliendo con las especificaciones del capítulo 3. Al igual que con el otro algoritmo, esta clase será utilizada en posteriores niveles de integración.

5.2.2. Nivel 2.

En este nivel partimos de la clase creada en el nivel anterior. Esta es encapsulada en la clase `DiVA_StaticMaskExtractor_HistoryImagesVPU` que hereda de `DiVAAlgorithm`. Con esto conseguimos nuevamente que el algoritmo quede integrado dentro de la plataforma pudiendo conectarse a los diferentes módulos presentes en él.

La fuente de vídeo de la aplicación será un servidor de cuadros de los disponibles en la plataforma, pudiendo proceder estos de archivos de vídeo o de cámaras. El vídeo resultado que produce el algoritmo vuelve a ser la misma cuadrícula de imágenes del nivel anterior (Figura 5.5). Es posible seleccionar si queremos mostrar el resultado en pantalla o si lo queremos almacenar en un archivo de vídeo o ambas cosas. La configuración de estas opciones, así como los datos de conexión al servidor son leídos de un archivo de configuración. El formato de este archivo es igual que en el ejemplo anterior.

Para esta aplicación también se ha implementado una consola que se ejecuta en el hilo del programa principal en paralelo al hilo de ejecución del algoritmo. Esta consola permite modificar algunos de los parámetros del algoritmo tras pulsar la tecla “c” y elegir el parámetro que queremos modificar. Al pulsar “Esc” el programa desconecta del servidor de cuadros y termina la ejecución.

La clase `DiVA_StaticMaskExtractor_HistoryImagesVPU` creada en este nivel será aprovechada en los siguientes niveles de integración.

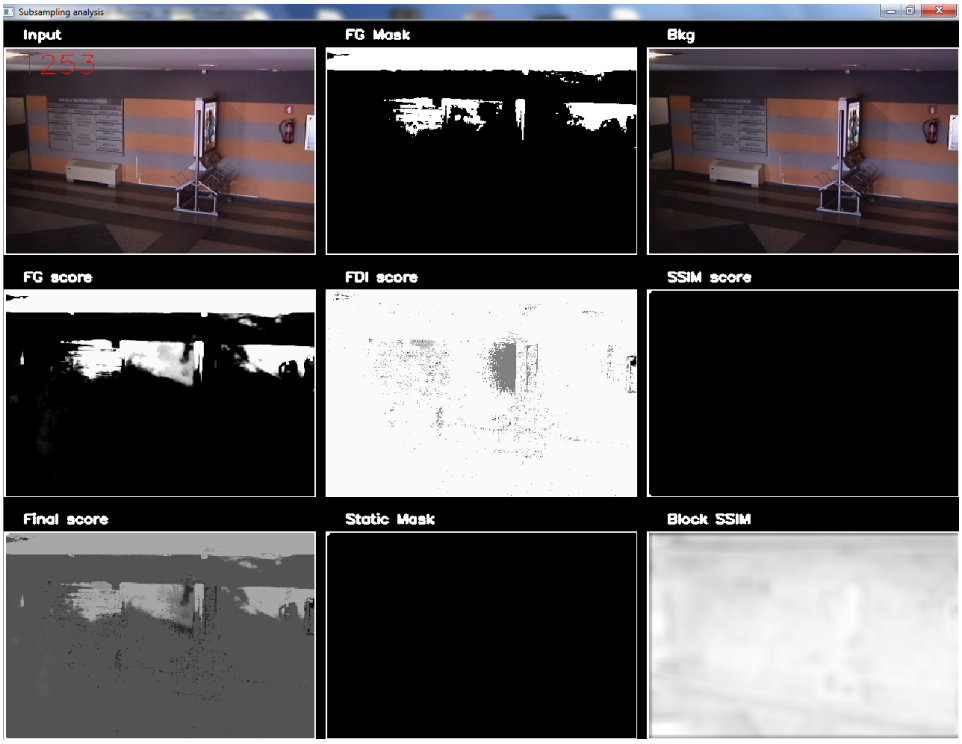

Figura 5.5: Cuadrícula de resultados nivel 2.

5.2.3. Nivel 3.1.

En este nivel se ha desarrollado la aplicación en su versión dirigida a desarrolladores, es decir, con posibilidad de visualizar resultados intermedios y con todos los parámetros modificables a la vista.

La disposición de elementos en la ventana se ha hecho siguiendo el esquema de la figura 4.1. A diferencia de la aplicación del nivel 3.1 del algoritmo anterior (sección 5.1.3), en esta la zona para introducir los datos del servidor de cuadros desaparece pues estos se introducirán utilizando otro método.

En la barra de menús hay dos menús, "Herramientas" y "Ayuda". El primero contiene únicamente la opción "Selección de cámara". En el segundo hay dos opciones, "Ayuda" y "Acerca de...". En la barra de herramientas situada bajo la barra de menús aparecen cuatro iconos. El primer icono, que representa una cámara, tiene la función de abrir la ventana de selección de cámara. El segundo, un cuadrado rojo, es para detener la ejecución del algoritmo. El tercero, un triángulo verde, sirve para iniciar el algoritmo. Por último, el icono de la interrogación abrirá la ventana de ayuda.

En la zona de botones nos encontramos con tres botones. La función del botón "Iniciar" es arrancar el algoritmo una vez está seleccionada la cámara. La del botón

"Detener" es parar la ejecución del algoritmo si ya estaba iniciado. Y el tercer botón, "Camara", sirve para abrir la ventana de selección de cámaras (servidores de imágenes). Junto a los botones se encuentran los diferentes parámetros que se pueden modificar y el botón "Aplicar" con el que se guardarán los cambios hechos en los parámetros. A la derecha de los parámetros están las casillas de selección con las cuales se puede elegir qué imágenes resultado queremos ver por pantalla. Las tres zonas anteriores se encuentran sobre la zona destinada a mostrar los resultados. Esta se diferencia por estar recuadrada y con el nombre de la cámara seleccionada en un cuadro de texto en la parte superior izquierda. Inicialmente se encuentra vacía, pero conforme se vayan marcando las casillas irán apareciendo las imágenes de resultado con el nombre impreso bajo ellas. En esta zona se hace uso de `DiVA_QtGridDisplay` (sección 4.2.3), por lo tanto se dispondrán automáticamente todas las ventanas de forma que las imágenes mostradas tengan el tamaño máximo posible. También se modificará el tamaño de los resultados si la ventana es reescalada. Por último en la parte inferior de la pantalla se hace referencia a las entidades colaboradoras en el proyecto incluyendo el logotipo de todas ellas.

La aplicación cuenta también con tres ventanas adicionales que complementan a la ventana principal. La primera es la ventana para seleccionar el servidor al que se conectará el algoritmo. Existen tres formas diferentes de abrir esta ventana: utilizando la opción "Selección de cámara" dentro del menú "Herramientas", pulsando el icono con forma de cámara en la barra de herramientas, o por último utilizando el botón "Camara". La ventana de selección de cámaras cuenta con una lista desplegable en la que se puede seleccionar entre las cámaras previamente guardadas, también hay botones que permiten añadir una cámara nueva, modificar una existente o eliminarla o establecer por defecto la cámara seleccionada.

La segunda ventana adicional es la de "Ayuda" (figura 5.6). En esta ventana aparece una pequeña descripción de cada uno de los parámetros que se pueden modificar y el valor que se recomienda que tomen. A esta ventana se accede desde la opción "Ayuda" del menú "Ayuda".

Por último, la tercera ventana es "Acerca de..." (figura 5.7). En esta ventana se incluye la información correspondiente al desarrollo tanto de la aplicación como del algoritmo. Aparece la versión de la aplicación, su fecha de realización, información de contacto del desarrollador de la aplicación y del algoritmo y los datos de contacto del laboratorio. Para abrir esta ventana habrá que seleccionar la opción "Acerca de..." del menú "Ayuda".

La secuencia de funcionamiento normal de la aplicación es la siguiente: una vez iniciada (figura 5.8), se debe seleccionar el servidor al que se conectará el algoritmo.

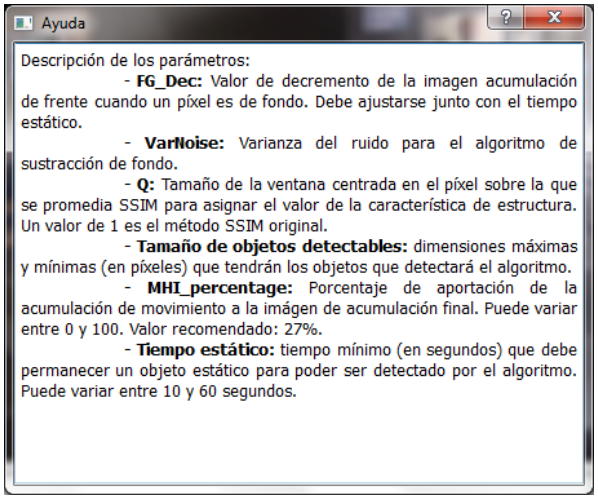

Figura 5.6: Ventana "Ayuda".

Figura 5.7: Ventana "Acerca de...".

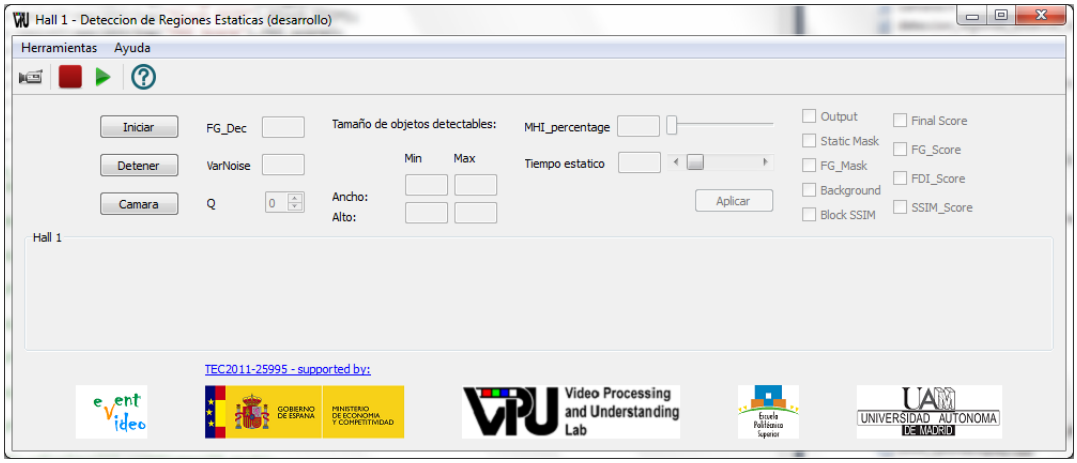

Figura 5.8: Estado inicial aplicación.

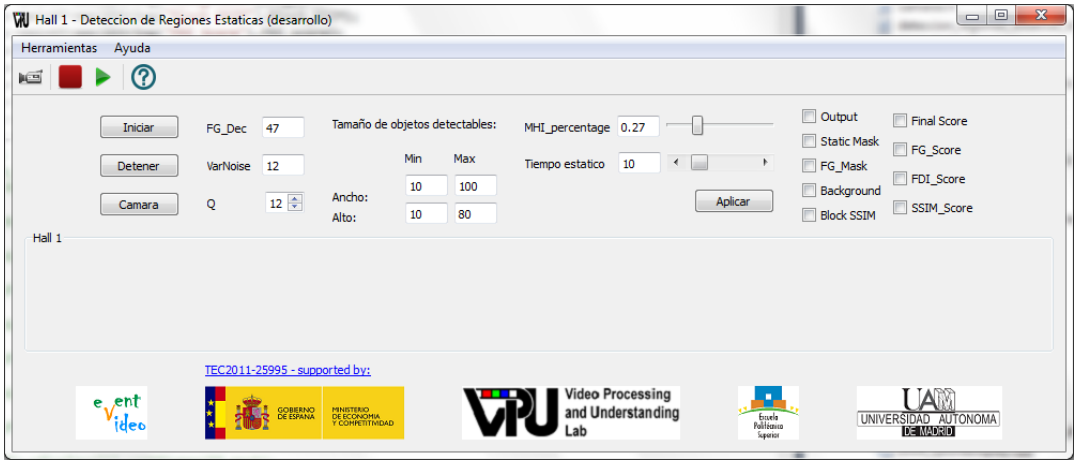

Figura 5.9: Aplicación funcionando y sin mostrar resultados.

Tras pulsar el botón "Iniciar" el algoritmo comenzará a funcionar y se habilitará la zona de configuración de parámetros (figura 5.9). En un primer momento no se muestra ningún resultado, estos deberán ser seleccionados entre los disponibles seleccionando la casilla correspondiente (figura 5.10). Durante la ejecución se podrá modificar el valor de los parámetros o cambiar de servidor deteniendo el algoritmo, seleccionando otro e iniciando de nuevo el algoritmo. Para finalizar el programa habrá que detener el algoritmo y pulsar el botón de cierre de la ventana.

5.2.4. Nivel 3.2.

La aplicación desarrollada en este nivel es similar en funcionamiento a la desarrollada en el nivel 3.1, pero cambiando la interfaz gráfica para hacerla más sencilla al

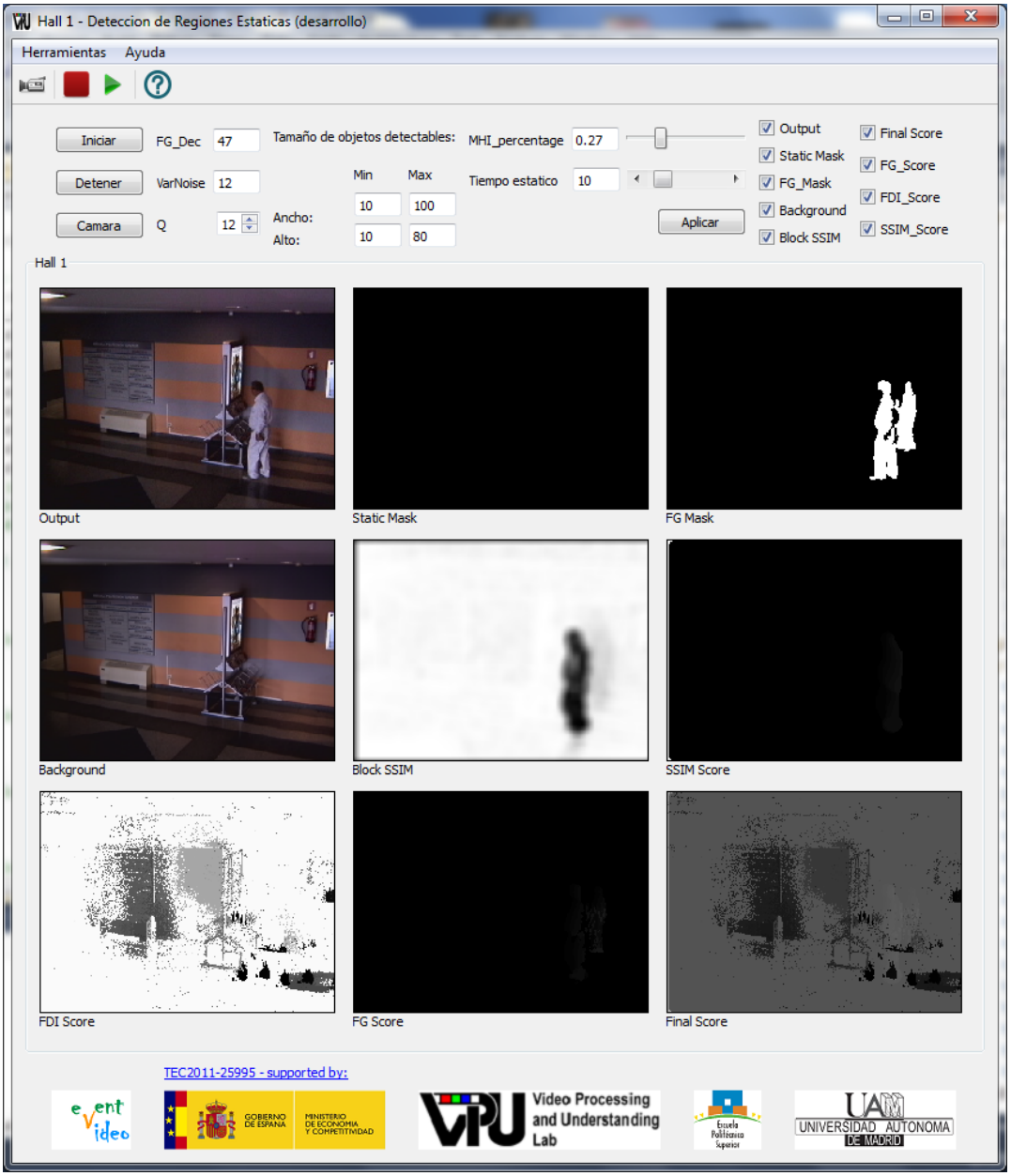

Figura 5.10: Cuadrícula de resultados en la aplicación.

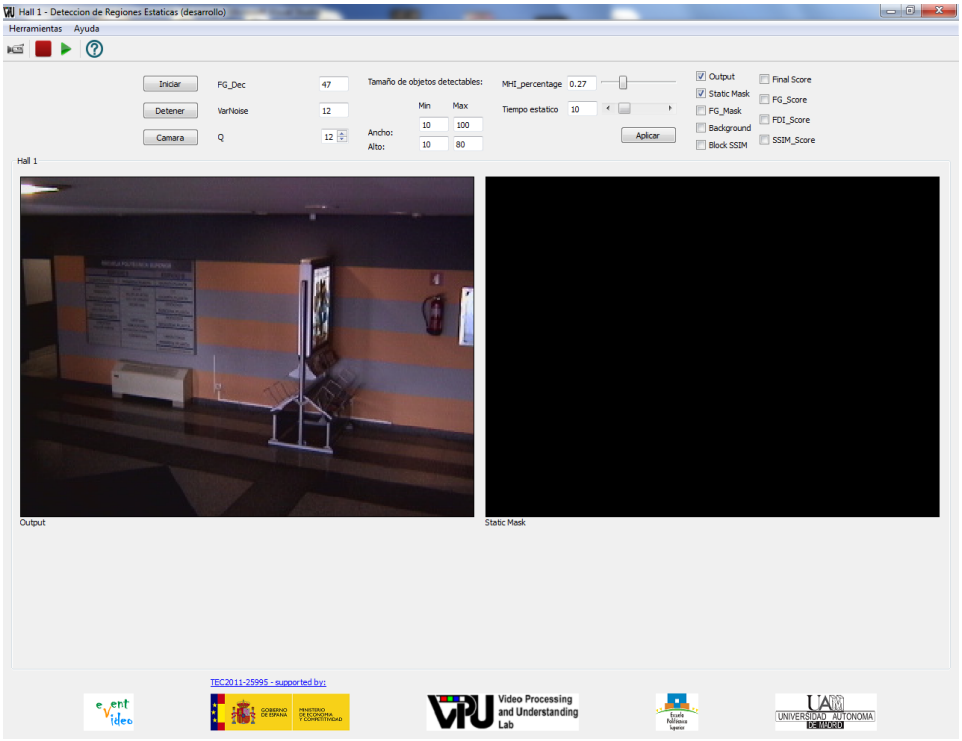

Figura 5.11: Ejemplo de reescalado según espacio disponible.

estar dirigida a un público menos técnico.

Partiendo de la clase `DiVA_StaticMaskExtractor_HistoryImagesVPU` del nivel 2, se ha diseñado la interfaz en torno a la imagen resultado final como elemento central y más importante de la aplicación. Bajo este elemento se encuentran los botones "Iniciar", "Detener", "Cámara" y "Parámetros". "Iniciar" pone en funcionamiento el algoritmo, "Detener" para la ejecución del algoritmo, "Cámara" abre la ventana de selección de cámaras y "Parámetros" abre una ventana independiente con los diferentes parámetros que se pueden configurar (ver figura 5.12). Debajo de los botones se sitúan los logos correspondientes a las entidades colaboradoras en el proyecto. En la parte superior se encuentran la barra de menús y la barra de herramientas. A las opciones presentes en los menús de la aplicación del nivel 3.1 se han añadido dos nuevas: "Modo compacto" y "Configurar parametros". La primera sirve para reducir la interfaz gráfica de modo que lo único que aparezca en la ventana sea la imagen de resultado, es decir, desaparecen los botones, logos y espacios que hubiera. Con esto se resalta la importancia de la imagen resultado sobre el resto de elementos. Con la otra opción, "Configurar parámetros", se abre la ventana que permite la configuración de los diferentes parámetros del algoritmo. En la barra de herramientas encontramos las mismas opciones y se añade una nueva con el icono de un engranaje para abrir la

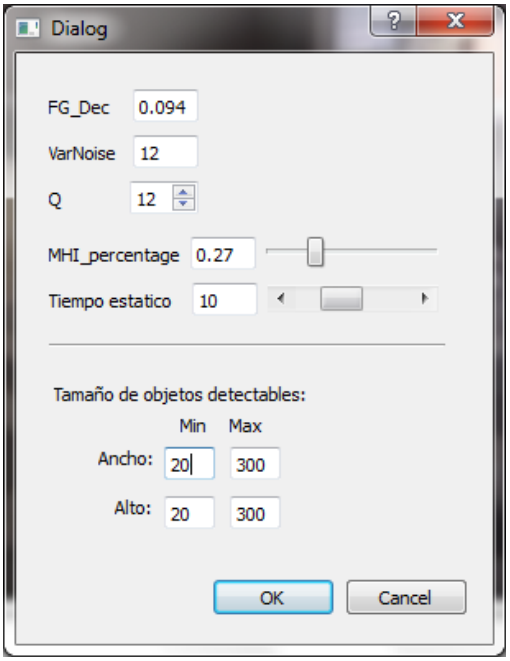

Figura 5.12: Ventana de configuración de parámetros.

ventana de configuración de parámetros.

La secuencia de funcionamiento normal para el programa es similar a las presentes en anteriores aplicaciones. Al iniciar aparece la ventana con un hueco en el centro donde irá la imagen una vez iniciado el algoritmo y los botones correspondientes a configurar parámetros y detener el algoritmo no realizan ninguna acción. Lo primero será elegir el servidor de cuadros con el que conectaremos e iniciar el algoritmo. Una vez está en funcionamiento se empieza a mostrar en la ventana la imagen resultado y ya se pueden utilizar los botones para detener el algoritmo o configurar los parámetros. Durante la ejecución puede activarse o desactivarse el modo compacto. Finalmente, para terminar la ejecución del programa hay que detener el algoritmo y cerrar la aplicación.

5.2.5. Nivel 3.3.

En este nivel se buscaba separar el módulo de procesado del algoritmo y el módulo de presentación (interfaz gráfica de usuario). Para conseguirlo, se ha aprovechado la arquitectura cliente-servidor utilizada en el resto de la plataforma, se ha añadido un servidor de cuadros a la clase `DiVA_StaticMaskExtractor_HistoryImagesVPU` al que se conectará la interfaz gráfica. Se ha eliminado también en este módulo la parte correspondiente a la presentación de resultados o su almacenamiento en fiche-

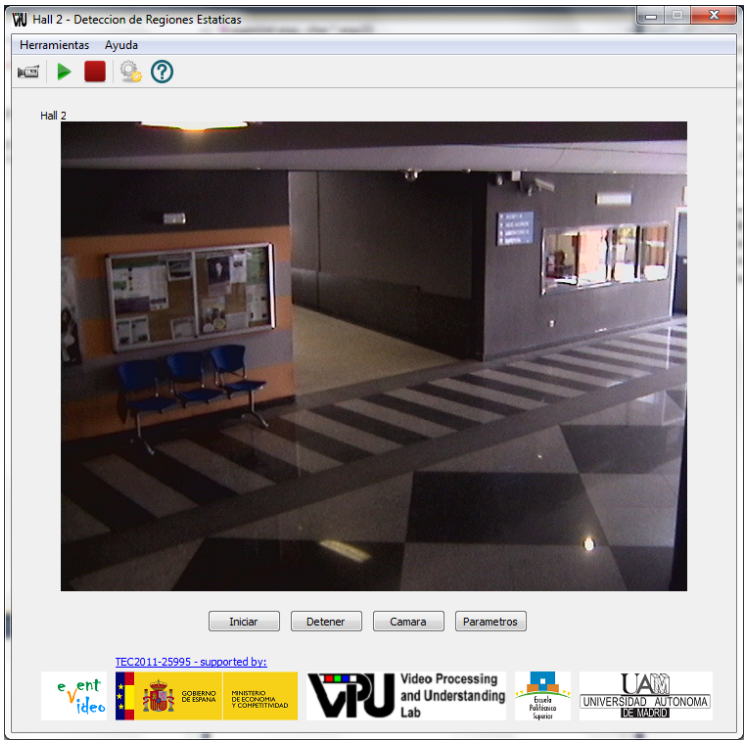

Figura 5.13: Aplicación nivel 3.2 en funcionamiento.

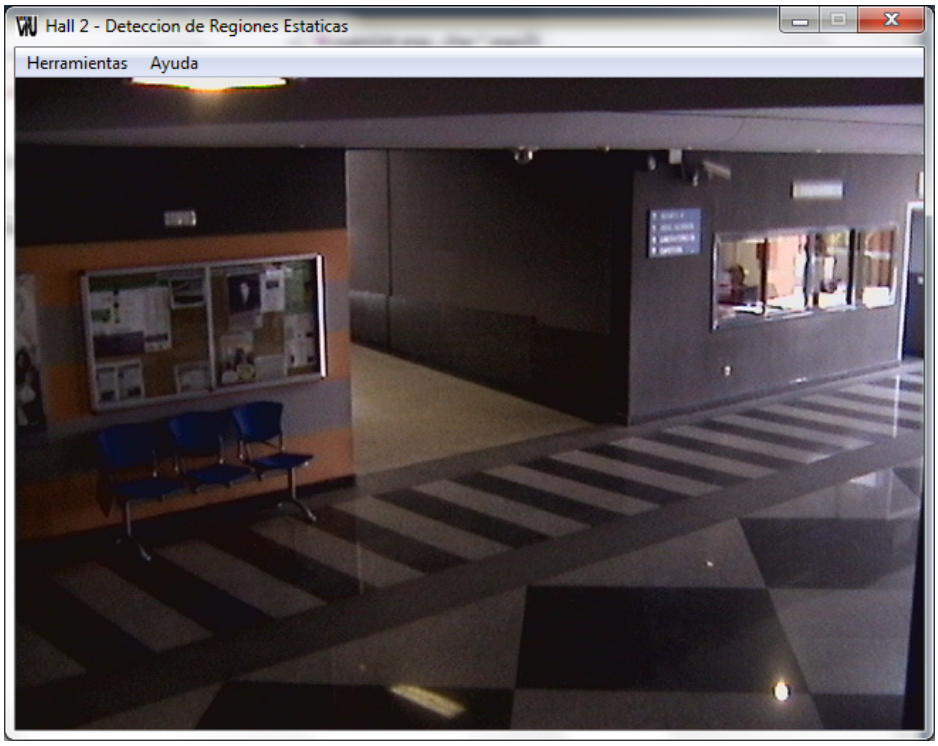

Figura 5.14: Modo compacto.

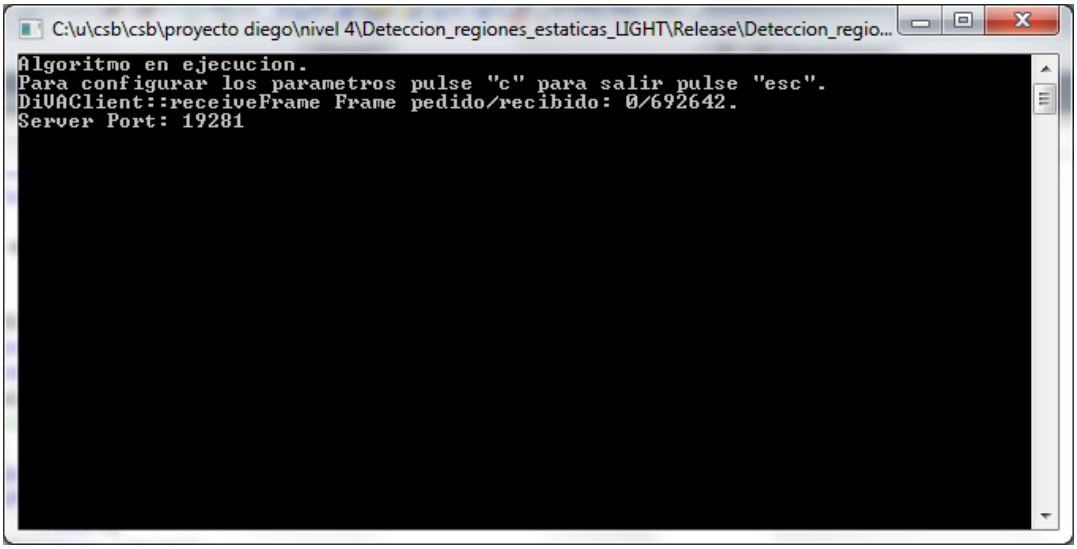

Figura 5.15: Algoritmo con servidor de imágenes.

ros de video, al no ser ya necesarios. Con estos cambios, se obtiene una aplicación de consola que se conecta con un servidor de cuadros, procesa las imágenes que recibe y crea un servidor de cuadros para enviar las imágenes procesadas a otro módulo de la plataforma. Esta aplicación de consola sigue permitiendo modificar los parámetros de ejecución del algoritmo porque no se ha suprimido esa funcionalidad, sin embargo, estos parámetros solo podrán modificarse en el equipo que ejecute el algoritmo y no en el que utilice la interfaz del módulo de presentación. En la figura 5.15 se puede observar como la aplicación que ejecuta el servidor al inicio de su funcionamiento nos indica en que puerto estará escuchando en busca de clientes a los que enviar las imágenes procesadas.

Por otro lado se tenía que diseñar una aplicación capaz de conectarse con el servidor de cuadros y mostrar las imágenes que recibía. Para ello se ha creado una aplicación como la del nivel 3.2 pero que implementaba un algoritmo vacío, es decir, que en la fase de procesado de las imágenes no las modifica. Con esto se consigue de una forma sencilla la comunicación con el servidor de cuadros creado en el módulo de procesado y se aprovecha el trabajo realizado en el diseño de la interfaz gráfica del nivel 3.2. En la figura se muestra la aplicación que implementa la interfaz gráfica funcionando, aunque aparentemente no difiere de la aplicación del nivel 3.2 (sección 3.3.3.2) esta aplicación no está procesando las imágenes que recibe.

Adicionalmente se ha eliminado de la aplicación la parte relacionada con la modificación de parámetros al no ser configurables desde la interfaz gráfica en este nivel. Gracias a no tener los parámetros, se ha obtenido una aplicación genérica para vi-

Figura 5.16: Visor de algoritmos en funcionamiento.

sualizar el resultado de las aplicaciones desarrolladas en el nivel 3.3. Es decir, esta aplicación no sirve únicamente para este algoritmo, si no que sirve para cualquier otro que implemente un servidor de cuadros como comunicación con la interfaz gráfica. Además, también se puede conectar con los servidores de cuadros de las cámaras, permitiendo ver que escenas se están capturando. Por estas razones, esta aplicación se proporciona como plantilla junto a la librería para ser utilizada directamente sin necesidad de volver a ser desarrollada.

Capítulo 6

Conclusiones y trabajo futuro.

6.1. Conclusiones.

El objetivo principal de este proyecto era crear una metodología de integración de algoritmos en el entorno de procesado de vídeo distribuido (DiVA) desarrollado en el Video Processing and Understanding Lab (VPULab) de la Universidad Autónoma de Madrid. Este se ha logrado de forma satisfactoria concluyendo con este documento que servirá de guía de integración de nuevas aplicaciones.

Como se ha visto a lo largo del proyecto, se ha conseguido simplificar la integración de nuevos algoritmos utilizando el principio "Divide y vencerás". La metodología propuesta divide la problemática del desarrollo de las aplicaciones en tres niveles. El primer nivel se centra en el desarrollo del algoritmo y en su codificación. El objetivo es preparar el algoritmo para centrarse en la integración en el siguiente nivel. El segundo nivel parte de un algoritmo funcional y tiene como objetivo la integración del mismo en la plataforma DiVA. Es por ello que todas las tareas de programación en este punto irán orientadas en ese sentido y no en modificar el algoritmo. Por último, el tercer nivel está orientado al desarrollo de la parte visual del algoritmo, la interfaz gráfica de usuario (GUI). Este nivel incluye la integración del algoritmo en la interfaz desarrollada. Al llegar al nivel final de integración se han establecido tres subniveles según a qué tipo de público están dirigidas las aplicaciones. Si la aplicación tiene como objetivo un desarrollador de algoritmos se ha definido un modelo aplicación que muestra tanto parámetros ajustables del algoritmo como resultados intermedios del procesado de las imágenes. La aplicación dirigida a clientes presenta una interfaz más sencilla en la que se muestra únicamente el resultado final. El último subnivel definido separa la interfaz gráfica del módulo de procesado, esta separación estaría destinada a entornos muy distribuidos en los que los núcleos de procesado se encuentren separados

de los puntos de visualización.

Como resultado de la elaboración de la metodología se han integrado dos algoritmos diferentes en la plataforma DiVA obteniendo hasta cuatro aplicaciones con interfaz gráfica. Pudiendo comprobar las grandes ventajas de un sistema distribuido frente a uno centralizado. Gracias a la metodología desarrollada se abre la puerta a la integración de nuevos algoritmos con la comodidad de no necesitar acceso a las cámaras para poder ponerlos en funcionamiento. Simplemente es necesario que el ordenador al que se conectan las cámaras establezca un servidor de cuadros de los disponibles en la plataforma y ya el resto de usuarios podrá recibir imágenes de esa cámara.

Para probar la metodología propuesta en el desarrollo del proyecto nueve algoritmos desarrollados en el laboratorio han sido integrados por cinco personas diferentes como trabajo fin de grado. Había algoritmos enmarcados en las diferentes ramas de la visión por computador. En el ámbito de la segmentación frente-fondo, además del algoritmo de substracción de fondo propuesto en la sección 5.1, han sido integrados un algoritmo de segmentación frente-fondo en entornos multimodales basado en [8, 9, 10] por Alejandro Blanco y una implementación del algoritmo *Gamma Background Subtractor* [11] para cámaras PTZ por Alberto Palero. En el campo de la detección de personas, Patricia Marín ha integrado cuatro algoritmos: primero el detector *Fusion* [12], luego el detector *Edge* [13], después el detector HOG [14] y finalmente el detector Latent SVM [15] . En seguimiento, Jorge San Juan ha integrado un algoritmo de seguimiento basado en *template matching* [16] y otro basado en filtros de partículas [17]. Finalmente en detección de eventos, junto al algoritmo de detección de regiones estáticas para robo/abandono (sección 5.2), Ana Huélamo ha integrado el algoritmo [18] de deteción de anomalías. Para todos estos algoritmos se ha realizado la implementación siguiendo los niveles y utilizando las plantillas y ejemplos proporcionados.

De forma paralela a la elaboración de la metodología se introdujeron otras mejoras en la plataforma. La principal fue añadir la posibilidad de codificar los cuadros antes de transmitirlos. Debido al carácter distribuido de DiVA y a que las imágenes se transmiten por la red, conforme aumente el número de elementos en la plataforma el volumen de tráfico generado en la red irá en crecimiento pudiendo llegar a ocupar todo el ancho de banda disponible en la red. Al transmitir las imágenes codificadas conseguimos reducir la cantidad de datos enviados por la red y no sobrecargarla.

También se cambió la capturadora utilizada para cámaras Firewire (IEEE1394) que antes dependía de una librería propietaria MIL desarrollada por Matrox para utilizar en la actualidad la librería gratuita y de código abierto VideoInput. Con esta nueva capturadora desarrollada se ha conseguido además añadir soporte para

cámaras que utilizan el estandar DirectShow de Microsoft como las cámaras web que se conectan al equipo por USB.

Finalmente se introdujo el soporte para algoritmos multicámara, permitiendo que el mismo algoritmo pueda recibir hasta imágenes de tres servidores de cuadros diferentes.

6.2. Trabajo futuro.

Tras la realización de este proyecto aparecen tres líneas de trabajo diferentes a seguir en el futuro. Estas serían: introducir mejoras en la plataforma DiVA, mejorar las aplicaciones propuestas (tanto a nivel de algoritmo como a nivel de aplicación) e integrar nuevos algoritmos en la plataforma.

Entre las mejoras a realizar sobre la plataforma DiVA estaría la creación de nuevos módulos de capturadoras para diferentes cámaras que aún no estén disponibles. Con estos nuevos módulos aumentaría la cantidad de cámaras compatibles con el sistema. Otro aspecto que serviría para aumentar las posibilidades de integración de la plataforma sería extender el soporte para algoritmos que necesitan imágenes de varias cámaras eliminando el límite actual de tres posibles fuentes de vídeo y permitiendo un número cualquiera de cámaras con las que conectar.

En cuanto a mejorar las aplicaciones y algoritmos realizados en el proyecto se proponen los siguientes cambios. El algoritmo de substracción de fondo (sección 5.1) se podría utilizar como etapa de preprocesado para otros algoritmos, habría que añadir nuevos módulos de procesado aumentando la funcionalidad y creando un sistema más complejo. Un ejemplo de módulo a añadir al algoritmo sería un detector de personas. El segundo algoritmo integrado en el desarrollo del proyecto (sección 5.2), ya es de por si un algoritmo completo y por lo tanto no sería necesario incluir nuevos módulos de procesado. Para este algoritmo las mejoras que se proponen van orientadas a mejorar el rendimiento para que pueda funcionar en tiempo real con imágenes de mayor resolución. Esto podría conseguirse optimizando el código, paralelizando tareas o aprovechando la arquitectura distribuida de la plataforma separando en diferentes módulos las diferentes etapas de proceso del algoritmo. También se propone como mejora a nivel de aplicación añadir un sistema de alarmas o notificaciones que avise al supervisor cuando se produzca algún evento. Otra posible mejora para las aplicaciones sería traducir las aplicaciones a diferentes idiomas para dar soporte multilingüe.

Por último, se podrían integrar los diferentes algoritmos desarrollados en el laboratorio para que pasen a formar parte de la plataforma. Con esto se conseguiría probar su funcionamiento en situaciones reales y no solo con videos de evaluación.

Bibliografía

[1] M. Valera and S. Velastin, "Intelligent distributed surveillance systems: a review," in *Vision, Image and Signal Processing, IEE Proceedings-*, vol. 152, pp. 192–204, IET, 2005.

[2] J. San Miguel, J. Bescos, J. Martinez, and A. Garcia, "Diva: A distributed video analysis framework applied to video-surveillance systems," in *Image Analysis for Multimedia Interactive Services, 2008. WIAMIS '08. Ninth International Workshop on*, pp. 207–210, May 2008.

[3] Y. Ivanov, C. Stauffer, A. Bobick, and W. Grimson, "Video surveillance of interactions," in *Visual Surveillance, 1999. Second IEEE Workshop on,(VS'99)*, pp. 82–89, IEEE, 1999.

[4] C. Nwagboso, "User focused surveillance systems integration for intelligent transport systems," in *Advanced Video-Based Surveillance Systems*, pp. 8–17, Springer, 1999.

[5] Z. Zivkovic, "Improved adaptive gaussian mixture model for background subtraction," in *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, vol. 2, pp. 28–31 Vol.2, Aug 2004.

[6] Z. Zivkovic and F. van der Heijden, "Efficient adaptive density estimation per image pixel for the task of background subtraction," *Pattern recognition letters*, vol. 27, no. 7, pp. 773–780, 2006.

[7] D. Ortego and J. SanMiguel, "Stationary foreground detection for video-surveillance based on foreground and motion history images," in *Advanced Video and Signal Based Surveillance (AVSS), 2013 10th IEEE International Conference on*, pp. 75–80, Aug 2013.

[8] A. Colmenarejo, M. Escudero-Vinolo, and J. Bescos, "Class-driven bayesian background modelling for video object segmentation," *Electronics letters*, vol. 47, no. 18, pp. 1023–1024, 2011.

[9] R. H. Evangelio, "Background subtraction for the detection of moving and static objects in video surveillance,"

[10] N. Goyette, P.-M. Jodoin, F. Porikli, J. Konrad, and P. Ishwar, "Changedetection. net: A new change detection benchmark dataset," in *Computer Vision and Pattern Recognition Workshops (CVPRW), 2012 IEEE Computer Society Conference on*, pp. 1–8, IEEE, 2012.

[11] A. Cavallaro, O. Steiger, and T. Ebrahimi, "Semantic video analysis for adaptive content delivery and automatic description," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 15, pp. 1200–1209, Oct 2005.

[12] V. Fernández-Carbajales, M. Á. García, and J. M. Martínez, "Robust people detection by fusion of evidence from multiple methods.," in *WIAMIS*, vol. 8, pp. 55–58, 2008.

[13] A. Garcia-Martin and J. M. Martinez, "Robust real time moving people detection in surveillance scenarios," in *Advanced Video and Signal Based Surveillance (AVSS), 2010 Seventh IEEE International Conference on*, pp. 241–247, IEEE, 2010.

[14] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1, pp. 886–893, IEEE, 2005.

[15] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part-based models," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 32, no. 9, pp. 1627–1645, 2010.

[16] R. Brunelli, *Template matching techniques in computer vision: theory and practice*. John Wiley & Sons, 2009.

[17] A. González Huete *et al.*, "Seguimiento y producción automática mediante cámaras ptz en entornos de red," 2013.

[18] P. Jodoin, V. Saligrama, and J. Konrad, "Behavior subtraction," *Image Processing, IEEE Transactions on*, vol. 21, no. 9, pp. 4244–4255, 2012.

Apéndice A

Definiciones de clases.

Este apéndice contiene la definición nueva de las clases de DiVA que han sido creadas o modificadas.

A.1. `DiVAImage.`

```
// Class definition
class DiVAImage
{
// Indicate private attributes
private:
        ///image identification
        long idImage;
        ///image timestamp
        long timestamp;
        ///image
        IplImage *image;
// Show public method
public:
        ///Default constructor
        DiVAImage();
        ///Constructor with basic parameters
        DiVAImage(long sizeX, long sizeY, int nChannel ,int Type);
        ///Constructor with a image in disk
        DiVAImage(char *filename,int iscolor=1);
        ///Class's Destructor
        virtual ~DiVAImage();
        ///Method to access to image pixels
```

```
23          DiVASCALAR getPixel(int x, int y);
24          ///Method to set one image pixel to a value
25          void setPixel(int x, int y, DiVASCALAR valor);
26          ///Set all pixels in the image to a reference value
27          void setPixels(DiVASCALAR value);
28          ///Set all pixels in the image to a reference matrix
29          void setPixels(void *value);
30          ///Gives direct access to pixels in the image
31          void *getPixels();
32          ///Set identification number
33          void setId(long id);
34          ///Set time stamp
35          void setTimeStamp( long timeStamp);
36          ///Set origin of pixels data
37          void setVAlign(int value);
38          ///Get the indentification number
39          long getId();
40          ///Get the time stamp
41          int getTimeStamp();
42          ///Get image width in pixels
43          int getWidth();
44          ///Get image height in pixels
45          int getHeight();
46          ///Get number of channels
47          int getNChannels();
48          ///Get type of image data
49          int getDepth();
50          ///Get image color model (RGB,HSV,...)
51          char *getColorModel();
52          ///Get origin of pixels data
53          int getVAlign();
54          ///Get Image data size in bytes
55          int getDataSize();
56          ///Flips image
57          void flip(int flip_mode);
58          ///Method to save a image in disk
59          int saveImage(char *filename);
60          ///Method to load a image from disk
61          int loadImage(char *filename,int iscolor=1);
62          ///Method to obtain a equal copy of the actual image
63          DiVAImage *clone();
```

```
64          ///Method to set the image pixels with the pixels content
               for the input image
65          int copy(DiVAImage *image);
66          ///Converts the image from RGB to Gray
67          int RGB2gray();
68          /// Method to encode the image
69          int encode(const char* ext,std::vector<int> parameters,std
               ::vector<uchar> &output);
70          /// Method to decode the image
71          int decode(std::vector<uchar> input,int flags);
72  };
```

A.2. DiVAServer.

```
1   class DiVAServer:public DiVAThread
2   {
3   public:
4           DiVAServer(DiVAImageBuffer* source, DescriptionBuffer*
               descSource,int portNumber);
5           ~DiVAServer();
6           //Threading control
7           int stop();
8           int process();
9           int initResources();
10          int releaseResources();
11          int processCommands(char *str,void* parametros);
12          int recvCommand(void* pclient,char*buffer);
13          int releaseAllClients();
14          int getNClients(){return _nclients;};
15          unsigned long int getNFramesServed(){return _framesServed
               ;};
16          void setNFramesServed(unsigned long int frames) { this->
               _framesServed = frames; return;};
17          int getFallos() { return this->_fallos;};
18          void setFallos(int fallos) {this->_fallos = fallos; return
               ;};
19          void sendEndAllClients();
20          void sendResetAllClients();
21          int sendEndClient(void *s);
22          int sendResetClient(void *s);
```

```
23          void setTypeCodification(int cod);
24  private:
25          DiVAImageBuffer* _source;
26          DescriptionBuffer* _descSource;
27          int _portNumber;
28          void* _pserver;
29          int _end;
30          void* _clientTable;
31          DiVAImage* getFrame(long idFrame, long idConsumidor, void*
                source=NULL);
32          Description* getDescription(long idFrame,long idConsumidor,
                void* source=NULL);
33          int getId();
34          int releaseId(long idClient,void* source=NULL);
35          void* setServerSocket(void* pserver);
36          void* getServerSocket(){return _pserver;};
37          int addClient(void* pclient);
38          int releaseClient(int _idclient);
39          int activaFlagDesconexion(int _idclient);
40          int _fallos;
41          int maxfd;
42          //int printClientsInfo();
43          int _nclients;
44          unsigned long int _framesServed;
45          int _TypeCod; // tipo de codificacion
46  public:
47          static int testFrameServer(char* filename);
48          int printClientsInfo();
49          int getPortNumber(){return _portNumber;};
50  };
```

A.3. DiVAClient.

```
1   class DiVAClient{
2   public:
3           DiVAClient(char* servername, int port=20248);
4           ~DiVAClient();
5   protected:
6           int SendCommand(char* command);
7           int WaitForImage(void**);
```

```
8           int WaitForDescription(void**);
9   private:
10          ///IP server name
11          char* _servername;
12          ///Connetion port
13          void* _pconn;
14          ///TCP/IP settings
15          struct hostent *_hp;
16          int _port;
17          int _idClient;
18          char* _sourceId;
19          int _isEnd;
20          int WaitForACK();
21          int WaitForId();
22          int DiVArecv(char*buffer ,int length);
23  public:
24          ///Method to connet with server
25          int Connect();
26          ///Method to known if task is end.
27          int isEnd(){return _isEnd;};
28          ///Method to set END status.
29          int setEnd();
30          ///Method to receive a frame from server
31          int receiveFrame(void** pbuffer,long idFrame);
32          ///Method to receive a description from server
33          int receiveDescription(void** pbuffer,long idFrame,char*
                descriptorName=(char*)NULL);
34          ///Method to set client ID
35          int setId(int idClient){_idClient=idClient;return _idClient
                ;};
36          ///Method to get client ID
37          int getId(){return _idClient;};
38          ///Get additional information about frameSource
39          char* getSourceId(){return _sourceId;};
40          ///Set additional information about frameSource
41          char* setSourceId(char* sourceId);
42          //Method to test basic operation
43          static int testFrameClient(char* servername);
44          void *getSocket(){ return this->_pconn; };
45          ///Method for checking server's activity
46          int DiVAClient::ReceiveKeepAlive();};
```

A.4. DiVACaptureFirewire.

```
class DiVACaptureFirewire : public DiVACapture { public:
        /// Default Constructor
        DiVACaptureFirewire();
        /// Constructor with specified format
        DiVACaptureFirewire(int resolutionX, int resolutionY, char
            *mode, char *fps, int camera);
        ///Default destructor
        virtual ~DiVACaptureFirewire();
        // DiVACapture public interface
        int init(int resolutionX=640, int resolutionY=480, char *
            mode="Y", char *fps="10", int camera =0);
        DiVAImage *getSampleFrame();
        ///Get current frame from the source
        int getCurrentFrame(DiVAImage *curFrame);
        ///Get the current frame from source and return it
        int getNextFrame(DiVAImage *nextFrame);
        ///Capture current frame from source (not return)
        int captureNextFrame();
        /// Get number of frames in file
        long getNumFrames();
        /// Get path from video source
        char *getFileName();
        ///Get status of Video capture
        int getStatus();
        // Returns number of systems (cameras) conected to capturer
        int getNumSystems();
// PRIVATE ATTRIBUTES
private:
        ///Images stored by the capture
        IplImage imagencv;
        videoInput VI;
        ///Capture Status
        int status;
        int *idConsumer;
        int numSystems;
        long frameId;
        unsigned char* buffer ;
        //To manage which camera is being used
        int id_camera;};
```

A.5. DiVAAlgorithm_multicam.

```
1  class DiVAAlgorithm_multiCam:public DiVAThread
2  {
3  public:
4          DiVAAlgorithm_multiCam(char* frameServername1,int
               portnumber1,
5          char* frameServername2,int portnumber2,
6          char* frameServername3,int portnumber3,
7          BOOL fileDumping=TRUE,
8          BOOL display=TRUE,
9          char* dataServerName=NULL,
10         char* contentServerName=NULL,
11         BOOL dataServerDumping=FALSE,
12         BOOL contentServerDumping=FALSE,
13         int captureMode=CAPTURE_BUF);
14
15         ~DiVAAlgorithm_multiCam();
16         char* getAlgorithmName(){return _algorithmName;};
17         //Initialization
18         virtual int init();
19         //Execution Control
20         int setEnd();
21         int process();
22         //Stop algorithm
23         int stop();
24         //Connections
25         int connect();
26         int getFrame(DiVAImage** pImage1,DiVAImage** pImage2,
               DiVAImage **pImage3,long idFrame1,long idFrame2,long
               idFrame3);
27         int receiveFrame(DiVAImage** pImage1,DiVAImage** pImage2,
               DiVAImage** pImage3,long idFrame1,long idFrame2,long
               idFrame3);
28         /************************************************/
29         /*To be overwriten ******************************/
30         //Data bases
31         virtual int receiveData(void** data){return 0;};
32         virtual int receiveContentData(void** data){return 0;};
33         virtual int releaseData(void* data){return 0;};
34         virtual int releaseContentData(void* data){return 0;};
```

```
35          virtual int dumpData(){return 0;};
36          virtual int dumpContentData(){return 0;};
37          virtual int dumpResultsInFiles(){return 0;};
38          //Image Processing
39          virtual int processFrame(DiVAImage* pImage1,DiVAImage*
               pImage2,DiVAImage* pImage3,void* pdata=NULL,void*
               pContentData=NULL);
40          //Displaying
41          virtual int refreshDisplay(){return 0;};
42          /***********************************************/
43  private:
44          BOOL _fileDumping;
45          char* _dataServerName;
46          char* _contentServerName;
47          BOOL _dataServerDumping;
48          BOOL _contentServerDumping;
49          BOOL _display;
50          long _lastId1;
51          long _lastId2;
52          long _lastId3;
53          int _captureMode;
54          //semaforo para sincronizacion entre los metodos stop() y
               process().
55          HANDLE hMutex;
56  public:
57          int getCaptureMode(){return _captureMode;};
58          BOOL getfileDumping(){return _fileDumping;};
59          char* getdataServerName(){return _dataServerName;};
60          char* getcontentServerName(){return _contentServerName;};
61          BOOL getdataServerDumping(){return _dataServerDumping;};
62          BOOL getcontentServerDumping(){return _contentServerDumping
               ;};
63          BOOL getdisplay(){return _display;};
64          DiVACrono* getCrono(){return _pCrono;};
65          long getLastId1(){return _lastId1;};
66          long getLastId2(){return _lastId2;};
67          long getLastId3(){return _lastId3;};
68          long setLastId1(long lastId){_lastId1 = lastId;return
               _lastId1;};
69          long setLastId2(long lastId){_lastId2 = lastId;return
               _lastId2;};
```

```
70          long setLastId3(long lastId){_lastId3 = lastId;return
               _lastId3;};
71          char* getDumpingpathRoot(){return _dumpingpathRoot;};
72          void setDisplay(bool display);
73          protected:
74          char* setAlgorithmName(char* name);
75          char* getSourceId();
76          char* getDumpingpath(){return _dumpingpath;};
77          char* _dumpingpath;
78          char* _dumpingpathRoot;
79          char* _frameServername1;
80          char* _frameServername2;
81          char* _frameServername3;
82          int _portnumber1;
83          int _portnumber2;
84          int _portnumber3;
85          char* _algorithmName;
86          DiVAClientBuffer* _pFrameBuffer1;
87          DiVAClientBuffer* _pFrameBuffer2;
88          DiVAClientBuffer* _pFrameBuffer3;
89          DiVAClient* _pFrameClient1;
90          DiVAClient* _pFrameClient2;
91          DiVAClient* _pFrameClient3;
92          DiVADisplay* _pDisplay;
93          DiVACrono* _pCrono;
94          DiVADataClient* _pDataClient;
95          /*En un futuro
96          DiVAdataClient* _pContentClient;
97          */
98          BOOL createDisplay(char* title,int rows,int columns,int h,
               int w);
99  //Connections
100         int connect2FrameServer();
101         int connect2DataServer(){return 0;};
102         int connect2ContentServer(){return 0;};
103 };
```

Apéndice B

Pruebas de velocidad de transmisión.

Para comprobar si los cambios realizados para transmitir las imágenes codificadas mejoran las prestaciones del sistema sin codificar se ha realizado la siguiente prueba, para tres videos de diferentes resoluciones (320x240, 720x576 y 1920x1080) se ha medido el tiempo que se emplea en la codificación de cada frame, la transmisión por la red y la decodificación. Se ha utilizado los siguientes esquemas de codificación: JPEG con diferentes calidades (100/100, 90/100 y 50/100) y PNG con diferentes valores de compresión (2/9 y 8/9). También se ha medido el tiempo empleado en la transmisión sin codificar.

Los equipos utilizados son:

- Servidor encargado de la codificación: Pentium 4, 3GHz y 1GB RAM.
- Cliente encargado de la decodificación: Intel Core 2 Duo E7500, 2,93GHz y 4GB RAM (3,46 útiles).

Los resultados obtenidos son los siguientes:

A la vista de los resultados (tablas B.1, B.2 y B.3), se observa que utilizando codificación JPEG se consigue reducir el tiempo de transmisión lo suficiente para que compense el tiempo que se tarda en codificar la imagen. Con PNG no sucede lo mismo, el tiempo que se empleado en codificar la imagen y decodificarla es mayor que el tiempo que se gana en la transmisión.

La ganancia obtenida para las distintas resoluciones es similar en cada formato utilizado, si bien el tiempo absoluto que se tarda se reduce más con videos de mayor resolución.

320x240			
Tipo codificación	Tiempo medio total (cod+Tx+decod)	% respecto a no codificado	fps
SIN CODIFICAR	41,49 ms		24,10
JPEG 100/100	30,03 ms	72,37%	33,30
JPEG 90/100	12,82 ms	30,90%	77,99
JPEG 50/100	7,86 ms	18,95%	127,19
PNG 2/9	51,71 ms	124,64%	19,34
PNG 8/9	44,32 ms	106,81%	22,56

Tabla B.1: Tiempos medios para un tamaño de imágen de 320x240 píxeles.

720x576			
Tipo codificación	Tiempo medio total (cod+Tx+decod)	% respecto a no codificado	fps
SIN CODIFICAR	157,15 ms		6,36
JPEG 100/100	103,31 ms	65,74%	9,68
JPEG 90/100	55,72 ms	35,46%	17,95
JPEG 50/100	42,79 ms	27,23%	23,37
PNG 2/9	192,85 ms	122,72%	5,19
PNG 8/9	187,48 ms	119,30%	5,33

Tabla B.2: Tiempos medios para un tamaño de imágen de 720x576 píxeles.

1920x1080			
Tipo codificación	Tiempo medio total (cod+Tx+decod)	% respecto a no codificado	fps
SIN CODIFICAR	541,68 ms		1,85
JPEG 100/100	384,30 ms	70,95%	2,60
JPEG 90/100	210,81 ms	38,92%	4,74
JPEG 50/100	169,69 ms	31,33%	5,89
PNG 2/9	634,46 ms	117,13%	1,58
PNG 8/9	652,17 ms	120,40%	1,53

Tabla B.3: Tiempos medios para un tamaño de imágen de 1920x1080 píxeles.

El valor que se proporciona de fps (*frames* por segundo) es el valor máximo que podría alcanzar de media el sistema, es decir, si tuviera un suministro de imágenes suficientemente rápido el servidor y el cliente también los procesara más rápido de lo que le llegan, el sistema sería capaz de alcanzar esa tasa de cuadros por segundo.

En las tablas B.4, B.5 y B.6 se muestran los resultados desglosados para codificación, transmisión y decodificación.

En ellas se observa que el elemento más lento del sistema es por lo general el codificador y es el que se podría mejorar. Esta posibilidad de mejora se debe a que el ordenador utilizado no tenía demasiada memoria RAM y por lo tanto la codificación se hacía más lenta. Si observamos los resultados del decodificador (con más RAM) observamos que el tiempo puede ser menor. Si aumentáramos la velocidad del codificador, los resultados obtenidos para JPEG serían aún mejores y los obtenidos con PNG serían mejores que sin codificar, permitiendo así utilizar un formato sin pérdidas y reduciendo el tiempo de procesado a la vez.

Otra cosa que se observa es que mientras que los tiempos de codificación y decodificación tienen poca desviación. No ocurre lo mismo con el tiempo de transmisión que según el estado de la red puede variar más que los otros. Por lo tanto otra ventaja de la transmisión codificada es que el frame rate será más constante al hacer menos uso de la red.

	320X240											
	Codificación				Transmisión				Decodificación			
	MAX	MIN	MEDIA	DESVIACION	MAX	MIN	MEDIA	DESVIACION	MAX	MIN	MEDIA	DESVIACION
SIN CODIFICAR	0 ms	0 ms	0 ms	0 ms	98,42 ms	22,93 ms	41,49 ms	7,90 ms	0 ms	0 ms	0 ms	0 ms
JPEG 100/100	14,29 ms	9,80 ms	10,11 ms	0,56 ms	931,55 ms	6,57 ms	16,45 ms	79,09 ms	4,01 ms	3,32 ms	3,47 ms	0,10 ms
JPEG 90/100	8,50 ms	5,18 ms	5,68 ms	0,47 ms	204,06 ms	1,39 ms	4,94 ms	17,17 ms	2,46 ms	1,99 ms	2,20 ms	0,11 ms
JPEG 50/100	6,93 ms	4,07 ms	4,50 ms	0,43 ms	4,12 ms	0,00 ms	1,58 ms	0,48 ms	4,80 ms	1,56 ms	1,79 ms	0,30 ms
PNG 2/9	28,38 ms	18,08 ms	19,56 ms	2,60 ms	633,49 ms	15,48 ms	26,91 ms	54,89 ms	7,02 ms	4,82 ms	5,24 ms	0,28 ms
PNG 8/9	26,37 ms	18,00 ms	18,80 ms	0,93 ms	49,40 ms	13,05 ms	20,24 ms	5,14 ms	8,32 ms	4,79 ms	5,28 ms	0,36 ms

Tabla B.4: Desglose de las medidas de tiempos para imágenes de 320x240 píxeles.

	720x576											
	Codificación				Transmisión				Decodificación			
	MAX	MIN	MEDIA	DESVIACION	MAX	MIN	MEDIA	DESVIACION	MAX	MIN	MEDIA	DESVIACION
SIN CODIFICAR	0 ms	0 ms	0 ms	0 ms	756,55 ms	125,50 ms	157,15 ms	77,85 ms	0 ms	0 ms	0 ms	0 ms
JPEG 100/100	57,75 ms	49,72 ms	51,69 ms	1,44 ms	404,56 ms	22,16 ms	33,71 ms	39,32 ms	20,73 ms	17,27 ms	17,91 ms	0,53 ms
JPEG 90/100	35,86 ms	28,42 ms	29,82 ms	1,06 ms	365,41 ms	6,35 ms	14,38 ms	34,19 ms	13,95 ms	10,97 ms	11,52 ms	0,44 ms
JPEG 50/100	35,60 ms	22,10 ms	23,28 ms	1,29 ms	587,60 ms	2,47 ms	10,20 ms	53,12 ms	12,38 ms	8,60 ms	9,31 ms	0,63 ms
PNG 2/9	100,83 ms	81,18 ms	92,04 ms	3,90 ms	582,95 ms	42,59 ms	73,97 ms	76,99 ms	37,74 ms	24,85 ms	26,84 ms	1,35 ms
PNG 8/9	138,85 ms	81,68 ms	93,32 ms	8,13 ms	959,70 ms	41,28 ms	67,37 ms	96,20 ms	34,85 ms	24,93 ms	26,79 ms	1,00 ms

Tabla B.5: Desglose de las medidas de tiempos para imágenes de 720x576 píxeles.

	1920x1080											
	Codificación				Transmisión				Decodificación			
	MAX	MIN	MEDIA	DESVIACION	MAX	MIN	MEDIA	DESVIACION	MAX	MIN	MEDIA	DESVIACION
SIN CODIFICAR	0 ms	0 ms	0 ms	0 ms	620,93 ms	532,20 ms	541,68 ms	9,91 ms	0 ms	0 ms	0 ms	0 ms
JPEG 100/100	279,11 ms	158,28 ms	204,87 ms	21,06 ms	597,95 ms	57,78 ms	105,40 ms	85,76 ms	97,96 ms	63,09 ms	74,03 ms	5,67 ms
JPEG 90/100	197,07 ms	108,75 ms	131,26 ms	16,12 ms	306,58 ms	19,42 ms	27,02 ms	25,30 ms	74,63 ms	48,11 ms	52,53 ms	3,20 ms
JPEG 50/100	173,38 ms	95,00 ms	110,93 ms	14,60 ms	246,32 ms	7,53 ms	13,17 ms	20,55 ms	73,78 ms	43,28 ms	45,60 ms	3,35 ms
PNG 2/9	485,86 ms	318,31 ms	375,21 ms	24,76 ms	230,44 ms	114,98 ms	147,18 ms	16,03 ms	119,76 ms	102,54 ms	112,07 ms	4,06 ms
PNG 8/9	525,48 ms	328,78 ms	377,55 ms	26,95 ms	397,78 ms	122,42 ms	162,51 ms	38,73 ms	121,08 ms	102,42 ms	112,11 ms	4,11 ms

Tabla B.6: Desglose de las medidas de tiempos para imágenes de 1920x1080 píxeles.

Apéndice C

Creación de interfaces gráficas con Qt.

En este apéndice se incluye una guía rápida de cómo crear una aplicación utilizando el *framework* Qt. Se explicarán los pasos a seguir para crear un proyecto nuevo y se hará una introducción a la herramienta QtDesigner. También dará una idea general sobre los diferentes elementos que se ha considerado que pueden ser útiles en el desarrollo de aplicaciones similares a las propuestas en el proyecto y de cómo utilizarlos. Estos elementos se separarán en diferentes secciones según la función que cumplan. Se explicarán los diferentes métodos de entrada de datos que están disponibles en Qt y los elementos para mostrar información o elementos de salida. Luego se tratarán los diferentes botones que pueden utilizarse y se explicará el sistema de "*Signals*" y "*Slots*" de Qt que nos servirá para comunicar los diferentes elementos de la interfaz.

C.1. Crear un proyecto Qt.

En este apéndice se explicará como crear un proyecto de Qt desde el entorno de desarrollo Microsoft Visual Studio 2010. Qt dispone de un plug-in necesario para poder crear y editar proyectos de Qt en este IDE.

Para crear un proyecto de Qt en Visual Studio, una vez instalado el *addin*, hay que seguir los siguientes pasos.

En primer lugar hay que abrir el menú crear nuevo proyecto siguiendo la ruta Archivo>Nuevo>Proyecto... (*File*>*New*>*Project*...) como se ve en la figura C.1.

Entre los diferentes proyectos que nos permite crear Visual Studio 2010 hay que seleccionar Qt Application dentro de la plantilla Qt5Projects. Esta plantilla nos creará un proyecto de una aplicación con interfaz gráfica con una ventana. En esta ventana

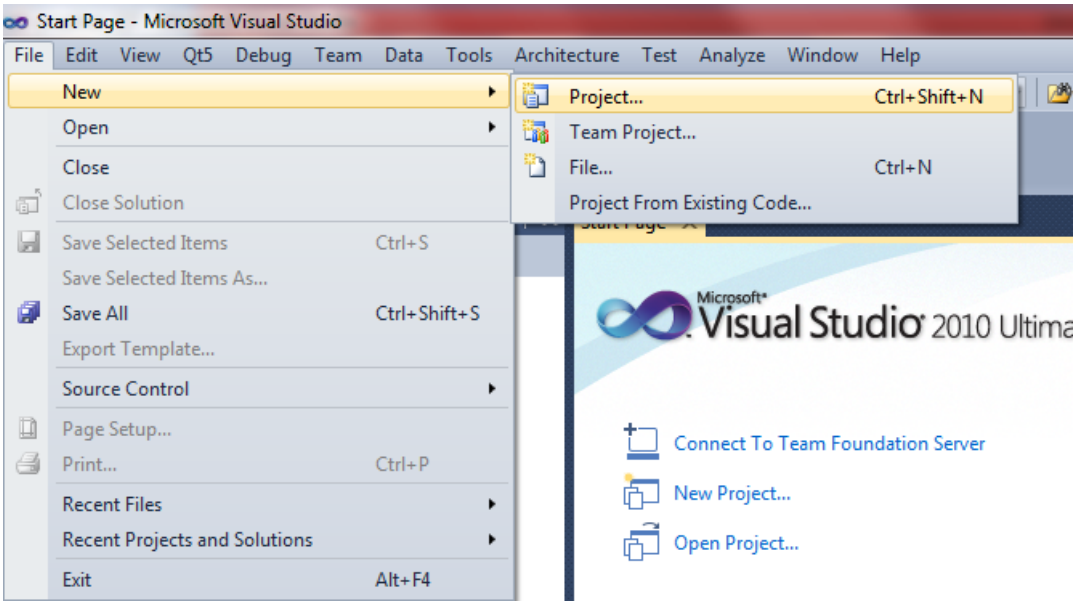

Figura C.1: Menú crear proyecto.

también podremos elegir el nombre del proyecto y de la solución así como su ubicación en el disco duro.

Una vez creado el proyecto se iniciará un asistente de Qt para configurar las diferentes opciones propias de Qt (figura C.3). En este asistente podremos elegir el nombre para la clase que implementará la ventana y los nombres de los archivos necesarios para ello. También es posible elegir la clase base sobre la que se basará nuestra ventana, las posibles opciones son:

1. `QMainWindow`: esta clase implementa una ventana típica en la que reserva hueco para una barra de menús, una barra de herramientas y una barra de estado.

2. `QWidget`: esta clase es la más simple de todas, implementa una ventana vacía

3. `QDialog`: es una plantilla orientada a crear un cuadro de diálogo con los botones habituales en este tipo de ventana: “Aceptar” y “Cancelar”.

Una vez finalizado el asistente de creación de proyectos de Qt ya se habrán creado todos los archivos necesarios para el proyecto. En la figura C.4 se muestra el aspecto que tendrá la solución. En ella se puede observar que además de los clásicos archivos de cabecera y de código fuente (*.h y *.cpp) aparecen otros tipos diferentes de archivos. En primer lugar aparece un archivo con la extensión .ui, este tipo de archivo tiene sintaxis de etiquetas xml y contiene la disposición gráfica de los elementos que

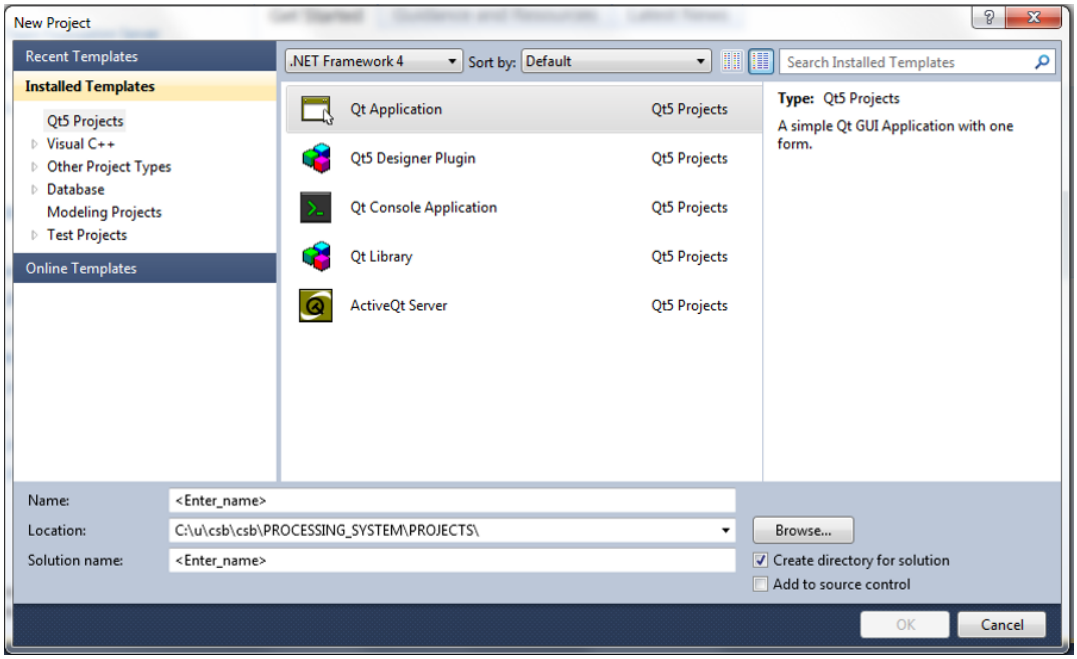

Figura C.2: Selección de tipo de proyecto.

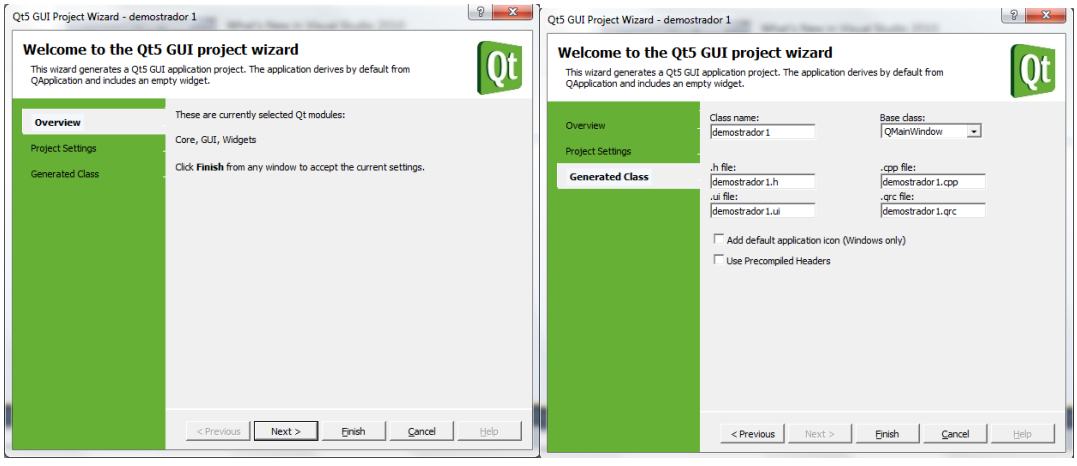

Figura C.3: Asistente de creación de proyectos Qt.

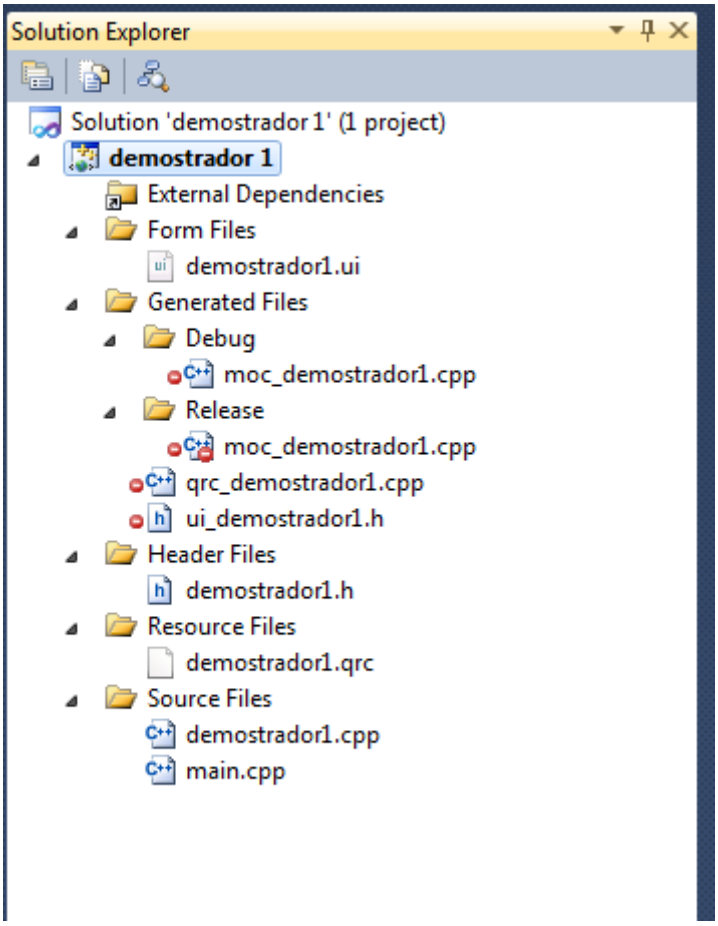

Figura C.4: Solución de Visual Studio 2010 con un proyecto de Qt.

componen la ventana (botones, cuadros de texto, etc.). El siguiente conjunto de archivos que aparecen dentro de la carpeta "Generated Files" son archivos que se generan automáticamente por el precompilador de Qt. Estos archivos en su gran mayoría son la traducción de los archivos .ui y .qrc a código C++ interpretable por el compilador de visual estudio. Estos archivos se eliminan y se vuelven a crear en cada compilación del proyecto si ha habido algún cambio en los otros archivos, por ello tiene utilidad alguna modificarlos.

C.2. Insertar elementos en la interfaz gráfica.

Una vez está creado el proyecto ya se puede empezar a diseñar la interfaz gráfica. En este apartado se explicarán algunos aspectos básicos de cómo hacerlo utilizando la herramienta QtDesigner que nos permite hacerlo gráficamente. También es posible

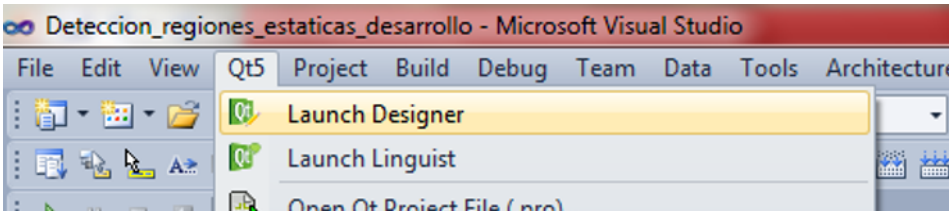

Figura C.5: Iniciar QtDesigner.

hacerlo mediante código C++ en el archivo que implementa la clase de la ventana o modificando el código del archivo xml con extensión .ui.

La herramienta QtDesigner podemos encontrarla como una de las opciones del menú Qt que aparece tras instalar el *add-in* en Visual Studio (figura C.5).

Una vez se abre la aplicación aparece una ventana en la que se nos da la posibilidad de crear una ventana nueva o abrir uno ya existente. Podemos utilizar el archivo que se generó al crear el proyecto de Qt abriéndolo desde esta ventana. Una vez cargado ese archivo aparece la ventana principal de QtDesigner (figuraC.6). En la zona central de esta ventana encontramos el aspecto que tendrá nuestra ventana y el lugar donde tendremos que ir añadiendo los elementos. A su izquierda se encuentra la lista con todos los elementos que podemos añadir a la ventana separados en categorías y con la posibilidad de filtrarlos. En la parte derecha hay cinco elementos. Arriba está el inspector de objetos "Object Inspector", en él podremos ver cada elemento que añadimos al diseño. En la zona central está el editor de propiedades donde podremos modificar las diferentes propiedades de cada objeto. El conjunto de propiedades modificables varía según que el elemento que esté seleccionado. Bajo este se encuentran agrupados en tres pestañas el editor de recursos, editor de acciones y editor de *signals* y *slots*. En el editor de recursos se añadirá lo referente a imágenes e iconos necesarios para el programa. El editor de acciones permite crear los diferentes botones que irán en la barra de herramientas y definir que se ejecutará al pulsar las opciones de los menús. El editor de *signals* y *slots* sirve para definirlas directamente en esta ventana en lugar de hacerlo en el código, este tema se tratará en secciones posteriores. En la parte superior de la ventana están la barra de herramientas y los menús.

Insertar elementos en nuestra ventana es tan simple como buscar cual queremos introducir en la lista hacer clic sobre él y arrastrarlo hasta la posición en la que queremos que aparezca. De esta forma podremos ir colocando todos los botones, cuadros de texto, etc. de la aplicación.

A cada elemento Qt le asigna un nombre de forma automática formado por el tipo de elemento y un número en el caso que ya hubiera más elementos del mismo tipo. Para cambiar el nombre habrá que seleccionar el elemento que queremos modificar

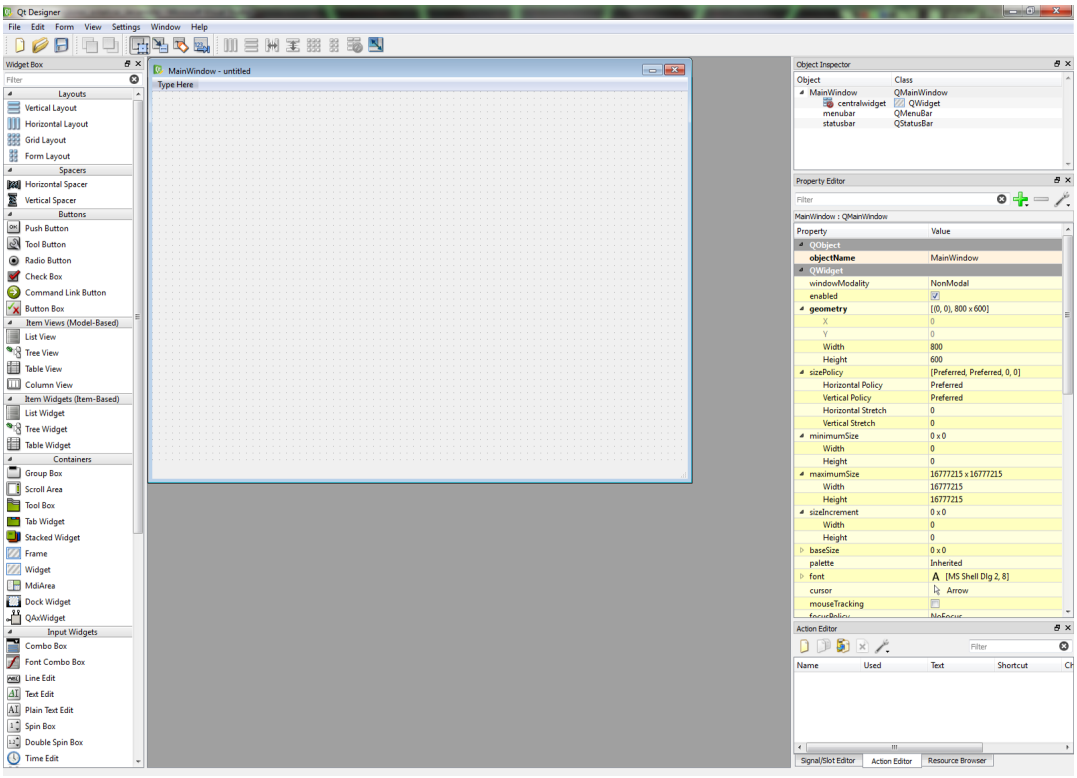

Figura C.6: Ventana principal QtDesigner.

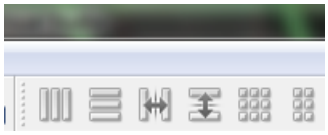

Figura C.7: Tipos de *layout*.

y en el panel de configuración de propiedades escribir el nuevo nombre. También podremos cambiar en este panel otras propiedades como tamaño, estado (activado o desactivado), texto, posición, etc.

Una vez insertados los elementos podemos acceder a ellos desde el código a través del objeto `ui` creado en la clase que implementa la ventana principal. Este objeto ui contiene dentro todos los elementos que añadimos desde QtDesigner.

Para ver el resultado final además de compilar y ejecutar el programa, QtDesigner nos da la posibilidad de ejecutar la ventana para ver la disposición de los elementos. Para iniciar esta vista hay que utilizar una de las opciones que hay dentro del menú *Window>Preview in*. De esta forma conseguiremos una previsualización de la ventana tal y como está diseñada en QtDesigner, los posibles cambios realizados en el código sobre los elementos no serán visibles.

C.2.1. *Layouts.*

A la hora de colocar los elementos en nuestra ventana podemos hacerlo manualmente o podemos establecer que se coloque automáticamente siguiendo unas reglas preestablecidas. Estas reglas de diseño en Qt se llaman "*Layouts*" y no son más que formas de agrupar elementos para que se coloquen en la ventana siempre de forma similar. Es decir, podemos definir una regla que obligue a que un botón y un cuadro de entrada de texto siempre estén alineados y uno junto al otro. De esta forma conseguiremos que al cambiar el tamaño de la ventana (por ejemplo al maximizarla) los elementos se recoloquen pero mantengan el mismo orden.

Para aplicar un *layout* en primer lugar hay que seleccionar los elementos sobre los que se aplicará y luego hay que elegir el tipo en la barra de herramientas (figura C.7).

Existen seis tipos diferentes de *layouts*:

1. *Layout* vertical: Se utiliza para colocar los elementos formando una columna, uno encima de otro.

2. *Layout* horizontal: Se utiliza para colocar los elementos uno a continuación de otro, formando una fila.

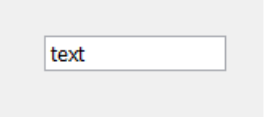

Figura C.8: `QLineEdit`.

3. *Layout in splitter* vertical: Similar al *layout* vertical pero introduce un separador entre los elementos que nos permite cambiar el espacio asignado a cada uno pudiendo modificar su tamaño.

4. *Layout in splitter* horizontal: Similar al *layout* horizontal pero introduce un separador entre los elementos que nos permite cambiar el espacio asignado a cada uno pudiendo modificar su tamaño.

5. *Layout* en cuadrícula: Dispone todos los elementos en una cuadrícula.

6. *Layout* en formulario: coloca los elementos en dos columnas.

C.3. Métodos de entrada.

En este apartado se describirán los principales métodos de entrada de datos que nos ofrece Qt.

C.3.1. `QLineEdit`.

Este elemento nos permite escribir y editar una línea de texto. El texto escrito podremos recuperarlo desde el programa utilizando el método `QLineEdit::text()` que devolverá una cadena de caracteres de Qt (`QString`).

Puede utilizarse para leer cadenas de caracteres que solo contengan números y luego convertir la cadena en un número con los métodos destinados para tal fin en la clase `QString`.

C.3.2. *Spinboxes.*

Un *spinbox* es un método de entrada de números formado por un cuadro de texto y dos flechas para modificar el valor del número. Este valor se puede cambiar también escribiendo el número directamente en el cuadro de texto. Qt cuenta con dos tipos diferentes, uno para números enteros (`QSpinBox`) y otro para números racionales (`QDoubleSpinBox`). Para acceder al valor desde el programa se utiliza el método `QSpinBox::value()` que devolverá el número que contiene.

Figura C.9: `QSpinBox` y `QDoubleSpinBox`.

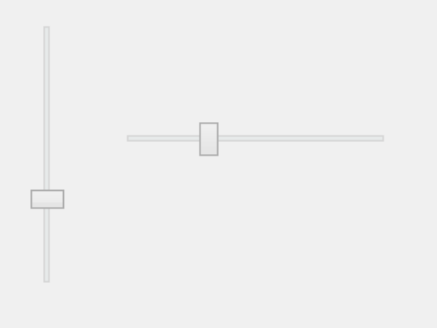

Figura C.10: *Slider* vertical y horizontal.

Es posible establecer un rango numérico entre los que pueda tomar valor el *spinbox* eligiendo el máximo y el mínimo. También el posible elegir el tamaño de los incrementos o decrementos al pulsar las flechas.

C.3.3. *Sliders.*

Los *sliders* están formados por una línea y un elemento móvil que se desplaza a lo largo de la línea. Hay dos tipos: vertical y horizontal. Este medio de entrada sirve para introducir números enteros. Se puede configurar los valores máximo y mínimo que pueden tomar y el intervalo entre las diferentes posiciones del deslizador. Para acceder al valor se utiliza el método `QSlider::value()` que devolverá el número entero correspondiente a la posición del elemento móvil.

Aunque directamente no es posible utilizar estos elementos para números decimales, podemos conseguir ese efecto mapeando el rango de valores objetivo al mismo rango de números enteros multiplicando por una potencia de 10 y luego deshacer la multiplicación al leer los valores. Es decir, si queremos un slider que oscile en el rango [0,1] en intervalos de 0.1 podemos crear un *slider* que se mueva entre [0,10] y al leer su valor dividirlo entre 10. Cuantas más cifras decimales queramos mayor será la potencia de 10 por la que habrá que multiplicar.

Figura C.11: *Scrollbar* vertical y horizontal.

C.3.4. *Scrollbars.*

Una *scrollbar* está formado por una barra móvil y dos flechas en los extremos que nos permiten mover la barra. Existen dos tipos de *scrollbar*: vertical y horizontal. Al igual que los *sliders* las *scrollbar* son métodos de entrada de números enteros, aunque como se ha visto en la sección anterior es posible utilizarlos para introducir valores no enteros. Para leer el valor de la *scrollbar* hay que utilizar el método `QScrollBar::value()` que devolverá el valor correspondiente a la posición. Para estos elementos también es posible definir los extremos del rango y el intervalo entre posiciones.

C.3.5. `QComboBox`.

La clase `QComboBox` implementa una lista desplegable. Se distingue por ser un cuadrado con texto y una flecha en la parte derecha que al pulsar sobre ella nos muestra una lista de opciones. Se puede rellenar la lista desde el propio QtDesigner haciendo doble clic sobre el elemento y también se puede hacer desde el código utilizando el método `QComboBox::insertItem` al que hay que indicarle la posición que ocupará en la lista el nuevo elemento y el nombre del elemento o utilizando el método `QComboBox::addItem` que solo recibirá como argumento el nombre y colocará el elemento al final de la lista. Es posible acceder al elemento seleccionado con dos métodos. El primero, `QComboBox::currentIndex`, nos devolverá el índice del elemento seleccionado mientras que el segundo, `QComboBox::currentText`, devolverá el texto de la selección. Para eliminar un elemento de la lista se utiliza el método `QComboBox::removeItem` que recibe como argumento el índice del elemento que hay que eliminar.

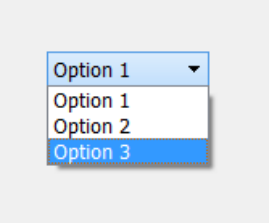

Figura C.12: `QComboBox`.

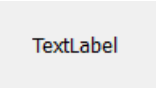

Figura C.13: `QLabel`.

C.4. Métodos de salida.

En este capítulo se describirán los principales métodos que podemos utilizar para que el programa comunique información al usuario.

C.4.1. `QLabel`.

El `QLabel` es un cuadro de texto no editable por el usuario. Es posible cambiar el texto que contiene tanto desde QtDesigner como desde el código del programa. En QtDesigner basta con hacer doble clic sobre el elemento y ya podremos cambiar el texto. Desde el código hay que utilizar el método `QLabel::setText` que recibe como argumento el nuevo texto.

Este elemento puede mostrar texto plano o texto enriquecido siguiendo el formato de etiquetas de HTML 4.

C.4.1.1. `QLabel` para mostrar imágenes.

Es posible utilizar el elemento `QLabel` para mostrar imágenes en la ventana. Para ello en primer lugar debemos eliminar el texto, y después en el panel de configuración de QtDesigner hay que marcar la opción "*Autofill Background*". Una vez hecho esto ya podemos mostrar imágenes en el cuadro de texto utilizando el método `QLabel::setPixmap` que recibirá como argumento un objeto de la clase `QPixMap` que contiene la imagen que queremos mostrar.

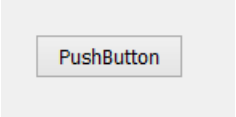

Figura C.14: `QPushButton`.

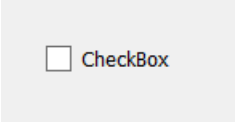

Figura C.15: `QCheckBox`.

C.5. Botones.

En esta sección se introducirá alguno de los diferentes tipos de botones disponibles en Qt.

C.5.1. `QPushButton`.

Esta clase implementa un botón que se puede pulsar para realizar una acción y que puede contener texto escrito sobre él. También es posible sustituir el texto por una imagen.

Para implementar la funcionalidad del botón habrá que definir un método en la clase con la siguiente sitaxis: `on_buttonName_clicked()`, donde `buttonName` es el nombre que tiene asignado el botón. De esta forma el precompilador de Qt conectará la señal que se produce al pulsar el botón con esta función.

C.5.2. `QCheckBox`.

Esta clase implementa una casilla seleccionable con un cuadro de texto. La casilla se puede definir con dos estados: seleccionada y no seleccionada, o con tres estados: seleccionada, no seleccionada y parcialmente seleccionada. Para obtener el estado de la casilla se puede utilizar el método `QCheckBox::checkState` que devolverá el estado de la casilla en una variable de tipo `Qt::CheckState`.

C.5.3. `QRadioButton`.

Esta clase implementa un botón circular seleccionable con una etiqueta de texto. Puede tener dos estados: seleccionado y no seleccionado. En un mismo grupo de `QRadioButton` solo puede haber uno seleccionado, si otro es marcado se borrará la selección anterior.

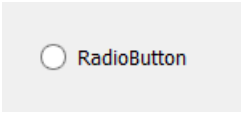

Figura C.16: `QRadioButton`.

C.6. *Signals* y *slots*.

Para establecer la comunicación entre los diferentes elementos de la interfaz en Qt hay un sistema implementado formado por *signals* y *slots*. Las *signals* son elementos que se emiten tras ocurrir un evento (clic en un botón, cambios en un cuadro de texto, seleccionar una casilla, etc.) y que servirán para iniciar un slot que es un método del objeto que se ejecutará cuando este reciba la signal que tiene asociada.

La mayoría de los elementos que hay definidos en Qt cuentan con su propio grupo de *signals*, por ejemplo, un botón tiene una *signal* para cuando se hace clic, cuando se pulsa, cuando se deja de pulsar,... Tambien es posible definir nuestras propias *signals* dentro de nuestra clase, para ello hay que añadir la directiva `signals:` antes de su declaración. Las *signals* no tienen implementación, es decir, solo hay que declararlas en el archivo de cabecera y ya se podrán utilizar.

Los elementos de Qt también tienen *slots* predefinidos, por ejemplo un objeto de tipo `QLineEdit` cuenta con *slots* para copiar el contenido, para pegar algo previamente copiado, para escribir texto, para deshacer,... De nuevo, podemos definir *slots* que se adapten a la funcionalidad que necesitemos. Los *slots* se definen igual que un método normal, pero antes de su declaración hay que incluir la directiva `slots:`.

Para establecer la comunicación entre *signals* y *slots* y hacer que al emitirse una *signal* se ejecute un *slot* hay que utilizar la función `connect` de Qt. La función recibirá como cuatro argumentos, en primer lugar recibirá el objeto que emite la *signal*, en segundo lugar la *signal* que se conectará, en tercer lugar el objeto que recibe la *signal* y por último el *slot* que se ejecutará cuando se reciba la *signal*.

Apéndice D

Presupuesto

1. **Ejecución Material**
   - Compra de ordenador personal (Software incluido) ..... 2.000€
   - Alquiler de impresora láser durante 6 meses ..... 260 €
   - Material de oficina ..... 150 €
   - Total de ejecución material ..... 2.400 €
2. **Gastos generales**
   - 16 % sobre Ejecución Material ..... 352 €
3. **Beneficio Industrial**
   - 6 % sobre Ejecución Material ..... 132 €
4. **Honorarios Proyecto**
   - 1800 horas a 15 € / hora ..... 27.000 €
5. **Material fungible**
   - Gastos de impresión ..... 280 €
   - Encuadernación ..... 200 €
6. **Subtotal del presupuesto**
   - Subtotal Presupuesto ..... 32.774 €

7. **I.V.A. aplicable**

   - 21 % Subtotal Presupuesto ..... 6.882,5 €

8. **Total presupuesto**

   - Total Presupuesto ..... 39.656,5 €

Madrid, FECHA

El Ingeniero Jefe de Proyecto

Fdo.: Carlos Sánchez Bueno

Ingeniero de Telecomunicación

Apéndice E

Pliego de condiciones

Este documento contiene las condiciones legales que guiarán la realización, en este proyecto, de un entorno de desarrollo de aplicaciones de de vídeo-seguridad. En lo que sigue, se supondrá que el proyecto ha sido encargado por una empresa cliente a una empresa consultora con la finalidad de realizar dicho entorno. Dicha empresa ha debido desarrollar una línea de investigación con objeto de elaborar el proyecto. Esta línea de investigación, junto con el posterior desarrollo de los programas está amparada por las condiciones particulares del siguiente pliego.
Supuesto que la utilización industrial de los métodos recogidos en el presente proyecto ha sido decidida por parte de la empresa cliente o de otras, la obra a realizar se regulará por las siguientes:

Condiciones generales

1. La modalidad de contratación será el concurso. La adjudicación se hará, por tanto, a la proposición más favorable sin atender exclusivamente al valor económico, dependiendo de las mayores garantías ofrecidas. La empresa que somete el proyecto a concurso se reserva el derecho a declararlo desierto.

2. El montaje y mecanización completa de los equipos que intervengan será realizado totalmente por la empresa licitadora.

3. En la oferta, se hará constar el precio total por el que se compromete a realizar la obra y el tanto por ciento de baja que supone este precio en relación con un importe límite si este se hubiera fijado.

4. La obra se realizará bajo la dirección técnica de un Ingeniero de Telecomunicación, auxiliado por el número de Ingenieros Técnicos y Programadores que se estime preciso para el desarrollo de la misma.

5. Aparte del Ingeniero Director, el contratista tendrá derecho a contratar al resto del personal, pudiendo ceder esta prerrogativa a favor del Ingeniero Director, quien no estará obligado a aceptarla.

6. El contratista tiene derecho a sacar copias a su costa de los planos, pliego de condiciones y presupuestos. El Ingeniero autor del proyecto autorizará con su firma las copias solicitadas por el contratista después de confrontarlas.

7. Se abonará al contratista la obra que realmente ejecute con sujeción al proyecto que sirvió de base para la contratación, a las modificaciones autorizadas por la superioridad o a las órdenes que con arreglo a sus facultades le hayan comunicado por escrito al Ingeniero Director de obras siempre que dicha obra se haya ajustado a los preceptos de los pliegos de condiciones, con arreglo a los cuales, se harán las modificaciones y la valoración de las diversas unidades sin que el importe total pueda exceder de los presupuestos aprobados. Por consiguiente, el número de unidades que se consignan en el proyecto o en el presupuesto, no podrá servirle de fundamento para entablar reclamaciones de ninguna clase, salvo en los casos de rescisión.

8. Tanto en las certificaciones de obras como en la liquidación final, se abonarán los trabajos realizados por el contratista a los precios de ejecución material que figuran en el presupuesto para cada unidad de la obra.

9. Si excepcionalmente se hubiera ejecutado algún trabajo que no se ajustase a las condiciones de la contrata pero que sin embargo es admisible a juicio del Ingeniero Director de obras, se dará conocimiento a la Dirección, proponiendo a la vez la rebaja de precios que el Ingeniero estime justa y si la Dirección resolviera aceptar la obra, quedará el contratista obligado a conformarse con la rebaja acordada.

10. Cuando se juzgue necesario emplear materiales o ejecutar obras que no figuren en el presupuesto de la contrata, se evaluará su importe a los precios asignados a otras obras o materiales análogos si los hubiere y cuando no, se discutirán entre el Ingeniero Director y el contratista, sometiéndolos a la aprobación de la Dirección. Los nuevos precios convenidos por uno u otro procedimiento, se sujetarán siempre al establecido en el punto anterior.

11. Cuando el contratista, con autorización del Ingeniero Director de obras, emplee materiales de calidad más elevada o de mayores dimensiones de lo estipulado

en el proyecto, o sustituya una clase de fabricación por otra que tenga asignado mayor precio o ejecute con mayores dimensiones cualquier otra parte de las obras, o en general, introduzca en ellas cualquier modificación que sea beneficiosa a juicio del Ingeniero Director de obras, no tendrá derecho sin embargo, sino a lo que le correspondería si hubiera realizado la obra con estricta sujeción a lo proyectado y contratado.

12. Las cantidades calculadas para obras accesorias, aunque figuren por partida alzada en el presupuesto final (general), no serán abonadas sino a los precios de la contrata, según las condiciones de la misma y los proyectos particulares que para ellas se formen, o en su defecto, por lo que resulte de su medición final.

13. El contratista queda obligado a abonar al Ingeniero autor del proyecto y director de obras así como a los Ingenieros Técnicos, el importe de sus respectivos honorarios facultativos por formación del proyecto, dirección técnica y administración en su caso, con arreglo a las tarifas y honorarios vigentes.

14. Concluida la ejecución de la obra, será reconocida por el Ingeniero Director que a tal efecto designe la empresa.

15. La garantía definitiva será del 4 % del presupuesto y la provisional del 2 %.

16. La forma de pago será por certificaciones mensuales de la obra ejecutada, de acuerdo con los precios del presupuesto, deducida la baja si la hubiera.

17. La fecha de comienzo de las obras será a partir de los 15 días naturales del replanteo oficial de las mismas y la definitiva, al año de haber ejecutado la provisional, procediéndose si no existe reclamación alguna, a la reclamación de la fianza.

18. Si el contratista al efectuar el replanteo, observase algún error en el proyecto, deberá comunicarlo en el plazo de quince días al Ingeniero Director de obras, pues transcurrido ese plazo será responsable de la exactitud del proyecto.

19. El contratista está obligado a designar una persona responsable que se entenderá con el Ingeniero Director de obras, o con el delegado que éste designe, para todo relacionado con ella. Al ser el Ingeniero Director de obras el que interpreta el proyecto, el contratista deberá consultarle cualquier duda que surja en su realización.

20. Durante la realización de la obra, se girarán visitas de inspección por personal facultativo de la empresa cliente, para hacer las comprobaciones que se crean

oportunas. Es obligación del contratista, la conservación de la obra ya ejecutada hasta la recepción de la misma, por lo que el deterioro parcial o total de ella, aunque sea por agentes atmosféricos u otras causas, deberá ser reparado o reconstruido por su cuenta.

21. El contratista, deberá realizar la obra en el plazo mencionado a partir de la fecha del contrato, incurriendo en multa, por retraso de la ejecución siempre que éste no sea debido a causas de fuerza mayor. A la terminación de la obra, se hará una recepción provisional previo reconocimiento y examen por la dirección técnica, el depositario de efectos, el interventor y el jefe de servicio o un representante, estampando su conformidad el contratista.

22. Hecha la recepción provisional, se certificará al contratista el resto de la obra, reservándose la administración el importe de los gastos de conservación de la misma hasta su recepción definitiva y la fianza durante el tiempo señalado como plazo de garantía. La recepción definitiva se hará en las mismas condiciones que la provisional, extendiéndose el acta correspondiente. El Director Técnico propondrá a la Junta Económica la devolución de la fianza al contratista de acuerdo con las condiciones económicas legales establecidas.

23. Las tarifas para la determinación de honorarios, reguladas por orden de la Presidencia del Gobierno el 19 de Octubre de 1961, se aplicarán sobre el denominado en la actualidad "Presupuesto de Ejecución de Contrata" y anteriormente llamado "Presupuesto de Ejecución Material" que hoy designa otro concepto.

Condiciones particulares

La empresa consultora, que ha desarrollado el presente proyecto, lo entregará a la empresa cliente bajo las condiciones generales ya formuladas, debiendo añadirse las siguientes condiciones particulares:

1. La propiedad intelectual de los procesos descritos y analizados en el presente trabajo, pertenece por entero a la empresa consultora representada por el Ingeniero Director del Proyecto.

2. La empresa consultora se reserva el derecho a la utilización total o parcial de los resultados de la investigación realizada para desarrollar el siguiente proyecto, bien para su publicación o bien para su uso en trabajos o proyectos posteriores, para la misma empresa cliente o para otra.

3. Cualquier tipo de reproducción aparte de las reseñadas en las condiciones generales, bien sea para uso particular de la empresa cliente, o para cualquier otra aplicación, contará con autorización expresa y por escrito del Ingeniero Director del Proyecto, que actuará en representación de la empresa consultora.

4. En la autorización se ha de hacer constar la aplicación a que se destinan sus reproducciones así como su cantidad.

5. En todas las reproducciones se indicará su procedencia, explicitando el nombre del proyecto, nombre del Ingeniero Director y de la empresa consultora.

6. Si el proyecto pasa la etapa de desarrollo, cualquier modificación que se realice sobre él, deberá ser notificada al Ingeniero Director del Proyecto y a criterio de éste, la empresa consultora decidirá aceptar o no la modificación propuesta.

7. Si la modificación se acepta, la empresa consultora se hará responsable al mismo nivel que el proyecto inicial del que resulta el añadirla.

8. Si la modificación no es aceptada, por el contrario, la empresa consultora declinará toda responsabilidad que se derive de la aplicación o influencia de la misma.

9. Si la empresa cliente decide desarrollar industrialmente uno o varios productos en los que resulte parcial o totalmente aplicable el estudio de este proyecto, deberá comunicarlo a la empresa consultora.

10. La empresa consultora no se responsabiliza de los efectos laterales que se puedan producir en el momento en que se utilice la herramienta objeto del presente proyecto para la realización de otras aplicaciones.

11. La empresa consultora tendrá prioridad respecto a otras en la elaboración de los proyectos auxiliares que fuese necesario desarrollar para dicha aplicación industrial, siempre que no haga explícita renuncia a este hecho. En este caso, deberá autorizar expresamente los proyectos presentados por otros.

12. El Ingeniero Director del presente proyecto, será el responsable de la dirección de la aplicación industrial siempre que la empresa consultora lo estime oportuno. En caso contrario, la persona designada deberá contar con la autorización del mismo, quien delegará en él las responsabilidades que ostente.