Universidad de Alcalá

Escuela Politécnica Superior

Ingeniero Técnico Industrial

Proyecto Fin de Carrera

Incorporación de algoritmos de procesado de imagen y vídeo al programa Forevid.

Autor: Jorge Irogbeniachi Onuoha Cid

Director: Pedro Gil Jiménez.

TRIBUNAL:

Presidente: .....

Vocal 1º: .....

Vocal 2º: .....

CALIFICACIÓN:..... FECHA:.....

Incorporación de algoritmos de procesado de imagen y vídeo al programa Forevid.

Jorge Irogbeniachi Onuoha Cid.

21 de junio de 2013

"Todo debe hacerse de la forma más sencilla posible pero no de la más fácil." Albert Einstein

Resumen

Forevid es un programa gratuito de código abierto, que ha sido diseñado principalmente para estudiar y manipular vídeos y crear informes sobre el estudio. Con Forevid se pueden aplicar diferentes algoritmos de filtrado a los vídeos sin modificar el contenido original. En este proyecto, se realiza una investigación, a través del código fuente, sobre los distintos caminos posibles para mejorara el programa mediante la incorporación de nuevos algoritmos de procesado de imagen a Forevid. Se obtienen dos métodos para la inclusión: crear un script de AviSynth para ejecutar el proyecto y añadir el algoritmo a la funcionalidad del programa en la cadena de filtros existente.

Summary

Forevid is a free open source program, which is designed primarily to study and manipulate videos and create reports about the study. With Forevid you can apply different filtering algorithms to videos without modifying the original content. This project, researchs through the source code, the different ways to incorporate new image processing algorithms in Forevid. Finally two methods have been developed for inclusion: Creating an AviSynth script to run the project and add the algorithm to the Forevid functionality in the existing filter chain.

Resumen extendido

Introducción

Forevid es un programa de software libre diseñado principalmente para la creación, edición y análisis de archivos de vídeo. Incorpora una serie de funciones por medio de las cuales se pueden editar las imágenes, cortar trozos del vídeo innecesarios, pegarlos, aplicarles una amplia gama de filtros, etc, y, sin modificar el vídeo original. Una vez hechas las correspondientes modificaciones pueden ser guardadas en varios formatos. Por otra parte incorpora una herramienta que facilita las anotaciones sobre las imágenes y permite crear documentos en pdf para presentación y elaboración de informes. Forevid no trabaja con el audio de los archivos, solo con las imágenes.

Objetivos del proyecto

El objetivo del proyecto está centrado en el análisis de la estructura interna, y el código fuente del programa Forevid, ya que los desarrolladores de la aplicación no han hecho pública la documentación relativa al desarrollo del código. La principal intención es la de poder estar en condiciones de añadir nuevos algoritmos de procesado de imagen para mejorar y ampliar su funcionalidad, y facilitar la comprensión del mismo a futuros desarrolladores que necesiten ampliar el programa, para que pueda ser modificado con mayor facilidad y rapidez.
Se crea también un manual en castellano de fácil manejo, para mejorar aún más la comprensión para que pueda ser utilizado por un mayor número de usuarios, sin necesidad de una formación específica.

Desarrollo

En primer lugar, en el capítulo 1 se desarrolla un manual, con una explicación completa del funcionamiento del programa. Se muestran imágenes de la interfaz gráfica para una mejor comprensión.

En los capítulo 2 y 3 se explica como funcionan los lenguajes AviSynth y Qt respectivamente, en orden de procurar unos conocimientos básicos sobre ambos lenguajes para mejorar la posterior comprensión del capitulo 4. Ambos capítulos se desarrollan utilizando sencillos ejemplos para hacer mas fácil y rápida la comprensión.

El capítulo 2, que introduce al lenguaje AviSynth, termina con un script de código, en el que se crea un algoritmo de filtrado. El algoritmo creado invierte los colores del vídeo al que es aplicado, y más adelante, se utiliza el mismo script como ejemplo para ser incluido a Forevid en el capítulo 5.

Se realiza un estudio profundo de la estructura del programa (capítulo 4), orientado a la incorporacion del algoritmo a Forevid. Se explica el funcionamiento interno usando el esquema que utilizaron los desarrolladores.

Finalmente, en el capítulo 5, tras un intenso estudio sobre las distintas vías que se pueden utilizar para incluir algoritmos en Forevid, se obtienen como solución al problema dos opciones:

- La primera, trata de añadir el algoritmo a través de un script de AviSynth. Este método es especialmente útil cuando se dispone de poco tiempo y solo se tiene disponible el código del algoritmo. La gran ventaja reside en que no hay necesidad de descargar, ni el código fuente de Forevid, ni un compilador (junto con los plugins necesarios) para modificar el programa. Simplemente con el archivo ejecutable de Forevid y el código del algoritmo desarrollado, se puede crear dicho scrpit y ejecutarlo con Forevid.
- La otra solución que se desarrolla, es mas compleja. Modificando el código fuente, se logra incorporar un nuevo comando al programa para ejecutar el algoritmo, además, se explica la mejor manera de modificar el código para incluirlo de forma apropiada. La ventaja que tiene es que el algoritmo forma parte de Forevid de manera permanente, y permite al usuario manejar de forma interactiva los parámetros del filtro, del mismo modo que los demás.

Campo de aplicación

El campo de aplicación de la herramienta es muy extenso, puede ser utilizado desde la persona que desea modificar un vídeo para filtrarlo o cambiarle el formato, hasta departamentos de investigación forense, que estudien detalladamente un vídeo obtenido de una cámara de vigilancia. En definitiva, su uso se extiende a cualquier tipo de usuario que trabaje con vídeos en distintos formatos, y le interese el estudio y la elaboración de informes sobre los mismos. El uso más práctico a nivel profesional es en el campo de la investigación forense.

Índice general

Lista de figuras

Capítulo 1

Descripción del programa Forevid

1.1. Introducción

Forevid es un programa de código abierto para análisis forense de vídeos de vigilancia [5]. Forevid ofrece un amplio conjunto de características requeridas en los trabajos de casos forenses, incluyendo por ejemplo, diversas opciones para la documentación de la reproducción de vídeo, el procesamiento y el resultado. Pero solo vale para interactuar con las imágenes de los archivos de vídeo, no trabaja con el audio.

Se puede utilizar Forevid para diferentes propósitos: como una herramienta intuitiva y simple para previsualizar vídeos de vigilancia, como herramienta avanzada para la mejora de vídeos de baja calidad o para analizar los acontecimientos de los vídeos y documentar los resultados obtenidos en sus análisis.

1.2. Manejo del proyecto

Forevid es utilizado para manejar casos bajo investigación como proyectos. Cada vez que se investigue un nuevo caso se abrirá un nuevo proyecto. El objetivo es que los trabajos se vean de forma organizada. Cuando se inicia el programa, aparece en pantalla la ventana de gestión de proyectos.

1.2.1. Empezar un nuevo proyecto

En la ventana de gestión de proyectos se crean nuevos proyectos. Para cada nuevo proyecto se rellena el nombre, la descripción y el autor del proyecto. Más tarde, esta información será mostrada en la ventana de gestión de proyectos. Los proyectos son creados en una estructura de carpetas que se muestran en la carpeta raíz de la ubicación. Por esta razón, el nombre del proyecto sólo puede contener caracteres permitidos en los nombres

del sistema de archivos de Windows. Se puede cambiar la ubicación de la carpeta raíz del proyecto pulsando "Browse". Tras crear un nuevo proyecto o abrir uno nuevo, se abre la ventana principal de Forevid.

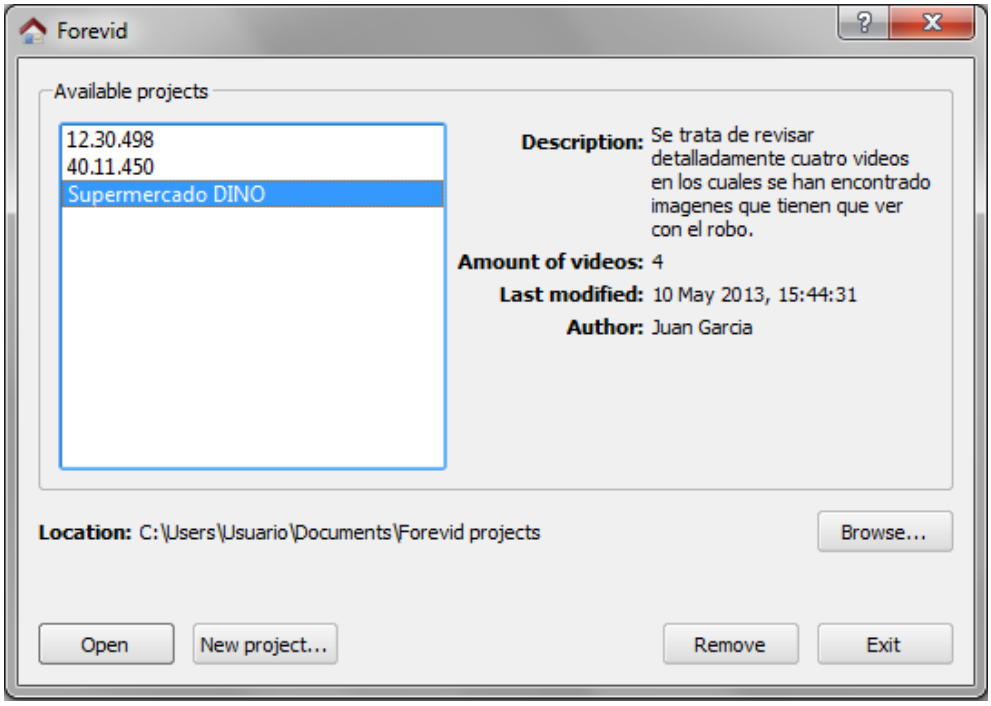


Figura 1.1: Ventana de gestión de proyectos

1.2.2. Acceso a proyectos existentes

La ventana de gestión de proyectos muestra la lista de todos los trabajos ubicados debajo de la carpeta raíz del proyecto. Además se muestra, la descripción del proyecto, la cantidad de vídeos, el tiempo de la última modificación y el autor de cada proyecto. Se puede abrir cada proyecto existente seleccionando el nombre del proyecto y pulsando "Open". Del mismo modo, se puede borrar el proyecto existente pulsando "Remove". Cuando un proyecto se elimina, la carpeta del proyecto y todo su contenido se eliminan de forma permanente.

1.3. Interfaz de usuario

Después de crear un nuevo proyecto o de abrir un proyecto que ya existe, se muestra la ventana principal de Forevid. Las partes de la interfaz de usuario son las siguientes: los vídeos importados se muestran en el lado izquierdo, todos los marcadores se muestran en el lado derecho, y la pantalla de vídeo y los controles en el centro de la pantalla.

Finalmente, la barra de menús contiene los menús comunes, y una barra de herramientas proporciona acceso a los comandos utilizados con mas frecuencia.

Figura 1.2: La interfaz de usuario principal para Forevid: (A) vídeos importados, (B) menús y barra de herramientas, (C) marcadores y (D) controles de vídeo.

1.4. Reproducción

1.4.1. Importar vídeos

Forevid proporciona una reproducción de los vídeos almacenados con una variedad de contenidos diferentes y códecs. Cuando se importa un vídeo nuevo (*File* → *Open* o botón de vídeo en la barra de herramientas) el usuario puede seleccionar uno de los métodos siguientes para importar:

- **FFmpeg** - Para importar vídeos se utiliza la biblioteca integrada *FFmpeg*. La biblioteca *FFmpeg* es compatible con una gran cantidad de contenidos de formatos diferentes y códecs.

- **DirectShow** - Se importan vídeos utilizando *Microsoft DirectShow*. Para importar correctamente el vídeo, el sistema operativo tiene que tener los splitters[1] *DirectShow* necesarios y los códecs instalados.

[1]Es un programa que demultiplexa el contenido multimedia y envía cada fracción a su respectivo decodificador para que se decodifique.

- **Vídeo para Windows** - Se importan los vídeos almacenados en formato AVI. Para importar correctamente el vídeo, el archivo debe estar en formato AVI, y el sistema operativo necesita tener instalado los códecs *VFW* necesarios.
- **Script de AviSynth** - Se importan como vídeo todos los archivos de script de aviSynth.

Además, se pueden importar los vídeos arrastrando los archivos de vídeo directamente desde el Explorador de *Windows* a la lista de los vídeos importados. En tal caso, se importa el vídeo utilizando el método de *FFmpeg*. Todos los vídeos importados se muestran como iconos en la parte de vídeos. De forma predeterminada, los nombres de los vídeos importados son definidos por los nombres de los archivos correspondientes, pero se pueden modificar seleccionando *File* → *Rename video*. Los cambios de nombre no cambian el nombre del archivo. Si se necesita acceder más tarde al archivo de vídeo actual, *File* → *Locate file* abre con el Explorador de Windows la carpeta donde se encuentra el archivo.

Si por alguna razón, Forevid no es capaz de importar el vídeo solicitado, aparecerá un cuadro de diálogo de error. Al seleccionar "Media info" desde el diálogo, se puede explorar la información detallada del archivo de vídeo, y por ejemplo, puede ser identificado el código *FourCC* del códec requerido.

1.4.2. Grabación desde pantalla

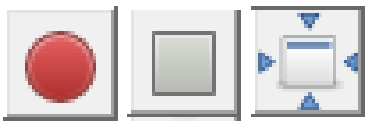
A veces se tratan vídeos de vigilancia almacenados en formatos propios, que sólo pueden ser reproducidos con reproductores registrados proporcionados por los fabricantes. En tal caso, el vídeo se puede importar a Forevid utilizando la funcionalidad de grabación de pantalla, que copia los cuadros que se muestran en la pantalla mientras se reproduce el vídeo con el reproductor privado. A partir de entonces, se genera un nuevo archivo de vídeo sin comprimir y se importa a Forevid.

La grabación de pantalla se inicia seleccionando *File* → *Record screen*. El usuario puede definir dos parámetros de grabación:

- **Nombre y ubicación del archivo generado**. De forma predeterminada, el vídeo grabado se almacena en la carpeta del proyecto.
- **Velocidad de grabación**. Define cuántos fotogramas se capturan en la pantalla por segundo. El valor adecuado para la tasa de grabación depende de la velocidad de fotogramas del vídeo original[2].

[2]Sin embargo, el rendimiento de grabación de la pantalla depende del rendimiento del equipo. Se debe verificar que toda la información relevante fue capturada.

Después de configurar los parámetros y pulsar el botón grabar, se muestra un rectángulo discontinuo, que se utiliza para definir el área de grabación y se ajusta con el ratón. La grabación se controla con los siguientes botones:



(a) **(b)** **(c)**

Figura 1.3: Botones que controlan la grabación

- **(a) Botón "Start recording"**. Empieza a grabar transcurridos tres segundos. Funciona como botón de pausa durante la grabación.
- **(b) Botón "Stop recording"**. Detiene la grabación de la pantalla y vuelve al cuadro de diálogo anterior. Después de cerrar el diálogo, el nuevo vídeo se importa en el proyecto actual.
- **(c) Botón "Select area"**. Selecciona el área de grabación basándose en la siguiente ventana. Seleccionando los recuadros blancos y manteniendo el botón izquierdo del ratón pulsado, se agranda, disminuye o desplaza el lugar sobre el que se desean grabar las imágenes.

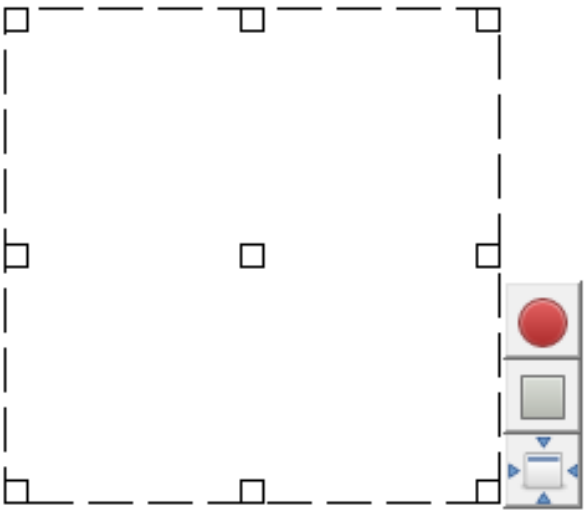


Figura 1.4: Ventana de selección de area

1.4.3. Reproducción

Existen diferentes opciones para la reproducción de vídeo. La reproducción normal del vídeo actual, se inicia con el botón de reproducción, y se muestra el vídeo con la velocidad de reproducción normal, desde la trama actual hasta el último cuadro del vídeo[3]. Un atajo de teclado para la operación de "Play / Pause" es la barra de espacio. La velocidad de reproducción se puede modificar, seleccionando con el botón derecho del ratón sobre "Play" escogiendo una nueva velocidad. Cuando seleccionamos *Video* → *Play all*, al terminar el vídeo actual, se inicia el siguiente vídeo automáticamente. Con las operaciones "Next frame" y "Previous frame", el usuario puede desplazarse hacia delante y hacia atrás cuadro por cuadro[4]. Los atajos de teclado para los comandos anteriores, corresponden a las flechas izquierda y derecha del cursor.

1.4.4. Búsqueda binaria

Es un método que se basa en la división sucesiva del espacio de tiempo del vídeo en sucesivas mitades, hasta encontrar el evento buscado. Si el evento que buscamos no se ha producido presionamos "shift + flecha derecha" (se pulsara "shift + flecha izquierda" si ya se ha producido), esto sitúa la reproducción en el punto intermedio del vídeo restante. Manteniendo presionada la tecla "shift" y usando las flechas del teclado, izquierda (para retroceder) y derecha (para avanzar), conseguiremos aproximarnos en un breve periodo de tiempo, al evento que se desee localizar en el vídeo mediante tramos de aproximación cada vez mas cortos (cada uno es exactamente la mitad del anterior). En caso de un vídeo largo, el método de búsqueda binaria puede ser significativamente mas rápido.

1.4.5. Otras acciones

En el vídeo mostrado se puede hacer zoom temporalmente con los botones "Zoom-in" y "Zoom-out" que están disponibles en la barra de herramientas. Los atajos de teclado correspondientes son "Z" y "X". Además, el tamaño original del vídeo se puede restaurar con la tecla "C".

La ventana "Media information" genera información sobre el archivo de vídeo. Por ejemplo, revela la identificación de los códecs que necesita el decodificador para reproducir el archivo.

[3]Sin embargo, dependiendo del nivel de rendimiento del equipo, la resolución del vídeo y la aplicación de filtros, se puede reproducir el vídeo a una tasa más lenta.

[4]Si el códec de vídeo utilizado no proporciona el indice preciso de los fotogramas, no sera posible desplazarse hacia atrás en el vídeo.

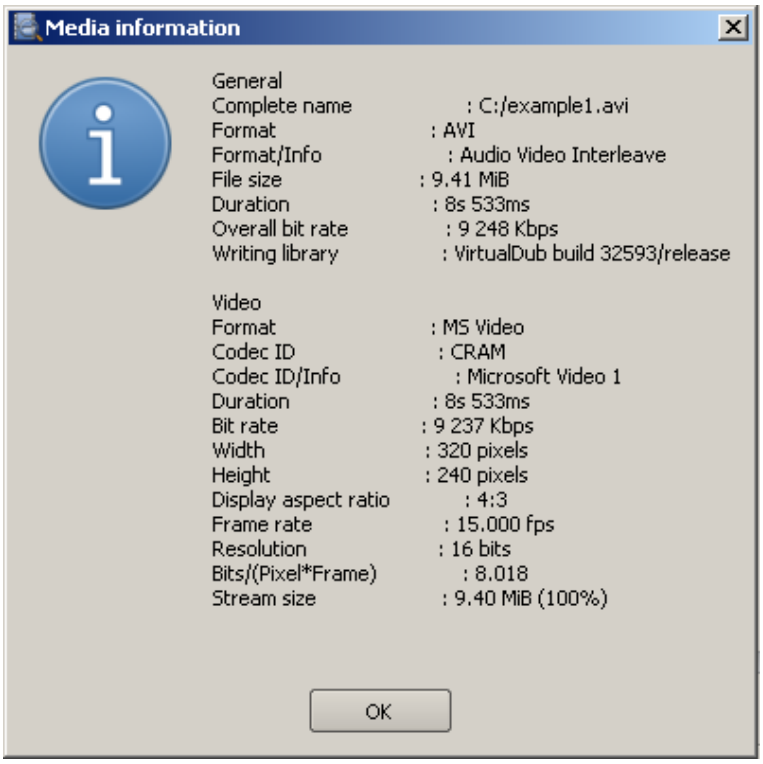


Figura 1.5: Ventana "Media information"

1.5. Procesamiento del vídeo

Para conseguir los resultados deseados en los vídeos analizados, se puede aplicar diversas operaciones de procesamiento de vídeo a los vídeos importados. Todo el procesamiento se realiza en la memoria del ordenador, dejando el vídeo original intacto. La configuración del vídeo genera una cadena de procesamiento que puede ser posteriormente modificada.

El procesamiento de vídeo se controla en la ventana de procesamiento de vídeo. Mediante la selección *Video* → *Filters* (o el botón en la barra de herramientas correspondiente), se abre una ventana donde el apartado de la izquierda muestra los filtros disponibles, y a la derecha los filtros que ya se han aplicado en el orden correspondiente. Cuando se añade un filtro, se abre una ventana donde se pueden modificar los parámetros del filtro. Al seleccionar "Vista preliminar", aparece una vista previa del vídeo procesado con el filtro dado. Cuando los parámetros se modifican, el botón "Update" actualiza la vista previa del vídeo. Cuando se pulsa "OK', se añade a la cadena de procesamiento el filtro con los parámetros dados. El último filtro de la cadena se puede eliminar con "Delete" o modificar sus parámetros de edición con el botón "Edit". En la ventana principal, seleccionando *View* → *Original video*, se puede visualizar el vídeo original y el modificado al mismo tiempo.

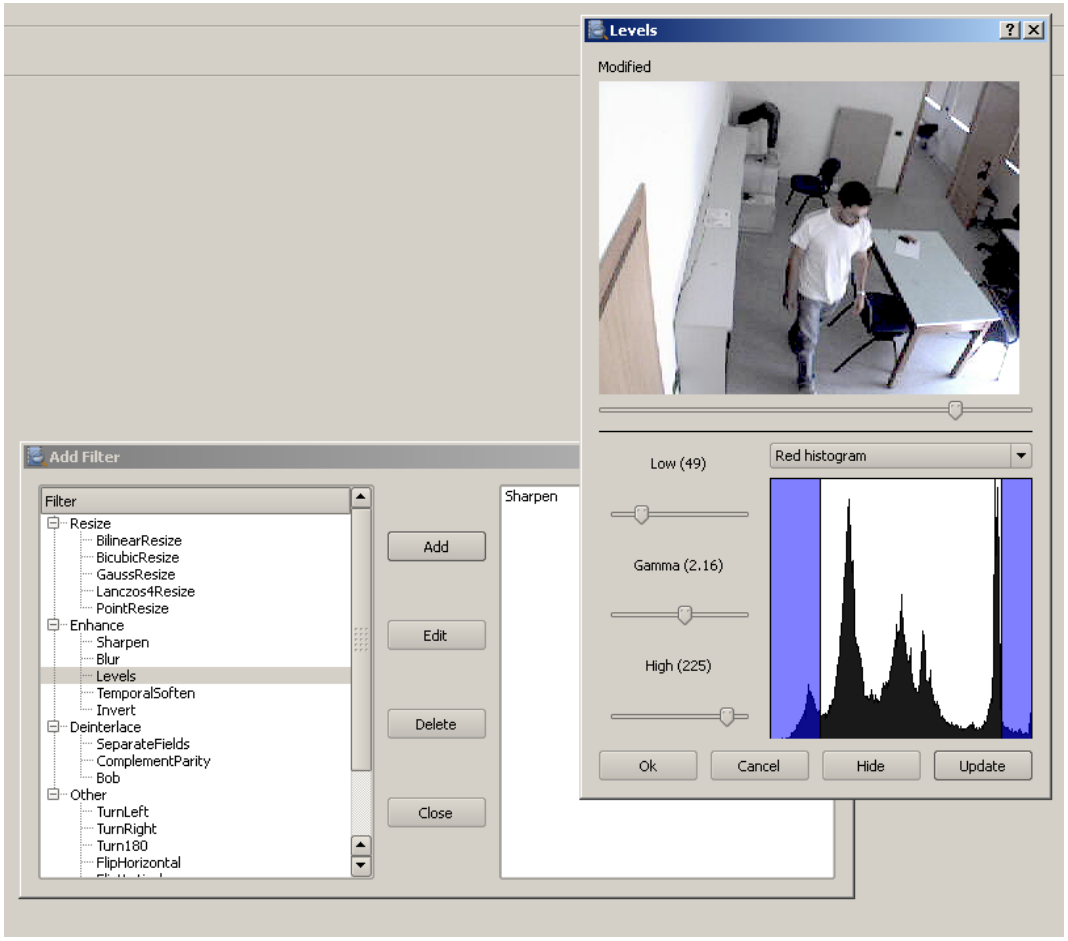


Figura 1.6: Ventana de procesamiento de vídeo

Los filtros se dividen en cuatro categorías, según su función: cambiar el tamaño, mejorar, desentrelazado, y otros. Actualmente están disponibles en Forevid los siguientes filtros[5].

Cambiar el tamaño:

- **PointResize** - Esta es la forma más sencilla posible de cambiar la resolución del vídeo. Se realiza el cambio de tamaño mediante el algoritmo de píxel más cercano, que por lo general da como resultado una imagen de peor calidad que con otros métodos.
- **BilinearResize** - Cambia el tamaño de los cuadros de entrada de vídeo a una nueva resolución arbitraria, mediante interpolación bilineal.
- **BicubicResize** - Similar a "BilinearResize", excepto que se utiliza la interpolación bicúbica. Los parámetros "b" y "c" se puede utilizar para ajustar las propiedades del

[5]Las descripciones de los filtros han sido traducidas de la pagina www.avisynth.com

cubo.

- **Lanczos4Resize** - Alternativa a "BicubicResize" que produce un fuerte enfoque. Por lo general, ofrece una mejor calidad (menos artefactos) y una imagen nítida.
- **Spline36Resize** - Es un método para modificar la resolución del vídeo, basado en ecuaciones con polímeros que utilizan 6 puntos de muestreo.

Mejorar:

- **Enfocar** - Trata de corregir el posible desenfoque del vídeo.
- **Blur** - Desenfoque de los fotogramas del vídeo. El parámetro controla la cantidad de desenfoque aplicada.
- **Deblock** - Desbloquear los fotogramas de vídeo. El parámetro "Quant" controla la fuerza de desbloqueo. El parámetro "AOffset", controla el detector de umbral de bloqueo. Cuanto mayor sea el valor, más bordes se desbloquean. "BOffset" es otro parámetro que se utiliza para controlar el bloque de detección y para controlar también la fuerza de desbloqueo.
- **SpatialSoften** - Eliminar el ruido de los fotogramas de vídeo de forma selectiva con la mezcla de píxeles. El filtro reemplaza cada muestra en un fotograma, con la media de todas las muestras cercanas que difieren de la muestra central por no más de un cierto valor de umbral. El tamaño de la ventana de filtrado está definido por el parámetro radio, y los umbrales se definen por separado para "luma_threshold" y "chroma_threshold".
- **TemporalSoften** - Funciona de manera similar a "SpatialSoften", a excepción de que los píxels se buscan en los fotogramas cercanos, en lugar de los píxels cercanos dentro del mismo fotograma.
- **Niveles** - Ajusta el brillo, contraste y gamma del vídeo. Los parámetros "low" y "high" determinan qué valores de píxel de entrada se tratan como negro puro y blanco puro. Para ayudar a seleccionar los valores correctos, el histograma de la imagen aparece junto a los parámetros. El parámetro *gamma* ajusta las proporciones relativas de las áreas brillantes y oscuras en las imágenes de vídeo.
- **Invertir** - Invierte los colores del vídeo.

Desentrelazar:

El desentrelazado sólo tiene sentido en vídeos analógicos con sistema de barrido entrelazado. No es aplicable para vídeos digitales con barrido progresivo.

- **SeparateFields** - Se toma un fotograma del clip y se divide en los campos que lo componen, produciendo un nuevo clip con el doble de velocidad de fotogramas.
- **ComplementParity** - Si el clip de entrada esta basado en campos, se cambian los campos superiores con los campos inferiores y viceversa.
- **Bob** - Se toma un clip y se desentrelaza. Esto significa que se amplía cada campo en su propio fotograma mediante la interpolación entre líneas. Los campos superior se empujan un poco más en comparación con los campos inferiores.

Otros:

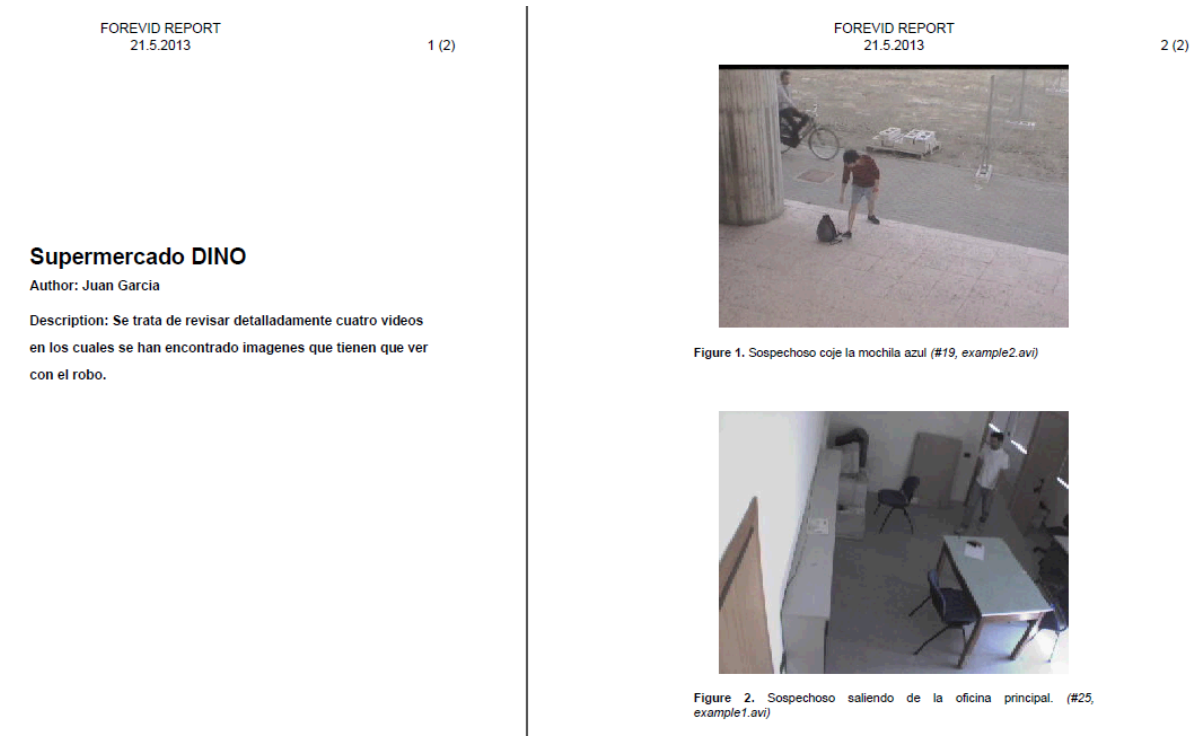
- **TurnLeft** -Se gira el vídeo 90 grados hacia la izquierda.
- **TurnRight** - Se gira el vídeo 90 grados hacia la derecha.
- **Turn180** - Se gira el vídeo 180 grados.
- **Reverse** - Hace que se reproduzca el vídeo en sentido inverso.
- **FlipVertical** - Invierte el vídeo en sentido vertical.
- **FlipHorizontal** - Da la vuelta al vídeo de izquierda a derecha.
- **ShowRed** - Muestra el canal rojo del vídeo.
- **ShowGreen** - Muestra el canal verde del vídeo.
- **ShowBlue** - Muestra el canal azul del vídeo.
- **AssumeFPS** - Cambia la velocidad de reproducción de vídeo por una velocidad dada por el usuario.
- **Loop** - Crea un bucle en el vídeo una cantidad determinada de veces.

1.6. Documentación

La documentación de resultados de análisis de cada vídeo analizado es una parte esencial. En Forevid, se pueden documentar los resultados mediante la exportación de imágenes y vídeos re-codificadas, o mediante la generación de marcadores e informes PDF. Cada proyecto tiene su propia carpeta de exportación donde se almacena todo el material de forma predeterminada. Se puede acceder a la carpeta de exportación seleccionando *File* → *Export folder* (o con el botón de barra de herramientas correspondiente).

1.6.1. Bookmarks (marcadores)

La ubicación de la mayoría de los fotogramas de interés se pueden almacenar con marcadores. Se crea un marcador seleccionando *Action* → *Add bookmark* (o con el botón de barra de herramientas), esto creará un marcador que apunta a la ubicación del fotograma actual, además, se puede añadir una descripción por escrito. Todos los marcadores se muestran en la lista de favoritos. Al seleccionar un marcador, el recuadro queda marcado en la pantalla[6]. Los marcadores y sus descripciones se pueden guardar como informe PDF, seleccionando *File* → *Export* → *Bookmarks as PDF*. Antes de generar el archivo PDF, los detalles del informe y la ubicación del archivo pueden ser modificados.


FOREVID REPORT
21.5.2013 1 (2)

Supermercado DINO

Author: Juan Garcia

Description: Se trata de revisar detalladamente cuatro videos en los cuales se han encontrado imagenes que tienen que ver con el robo.

FOREVID REPORT
21.5.2013 2 (2)

Figure 1. Sospechoso coje la mochila azul *(#19, example2.avi)*

Figure 2. Sospechoso saliendo de la oficina principal. *(#25, example1.avi)*

Figura 1.7: Resumen de un informe PDF de los marcadores de fotogramas

[6]Si el códec de vídeo utilizado no soporta indexación, el marcador podría no apuntar a la estructura correcta.

1.6.2. Imágenes fijas

Se pueden generar imágenes de diferentes maneras. En primer lugar, la forma más sencilla de almacenar un fotograma destacado es presionar el botón de grabación (en la ventana principal en la parte de controles de vídeo). Se trata de una operación de acceso directo que guarda automáticamente el fotograma actual, en la carpeta de exportación, en un formato predefinido. El formato de la imagen se puede modificar seleccionando *File* → *settings*. En segundo lugar, el fotograma actual se puede exportar al porta-papeles seleccionando *File* → *Export* → *Current frame to clipboard*. *Todos los fotogramas se pueden guardar como imágenes estáticas seleccionando File* → *Export* → *Bookmarked frames as images*. Por último, se pueden guardar una serie de imágenes consecutivas seleccionando un tramo en la línea de tiempo del vídeo con la secuencia *File* → *Export* → *Selected frames as images*.

1.6.3. Editar imágenes fijas

Antes de guardar el fotograma seleccionado como imagen, se puede editar. Esto se hace seleccionando *File* → *Export* → *Current frame for editing*. A partir de entonces, el fotograma actual se abre en el editor de imágenes como se muestra en la siguiente figura. El fotograma resultante puede ser guardado como una imagen o como una copia en el porta-papeles.

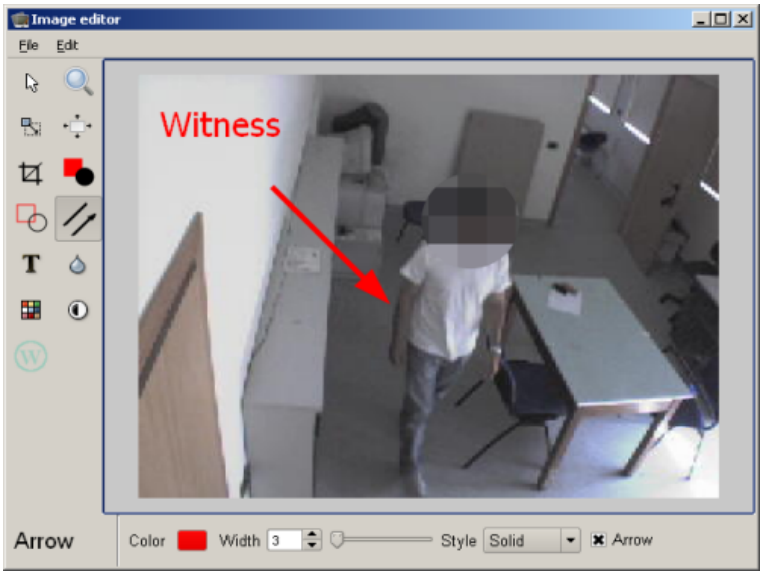


Figura 1.8: Editor de imágenes

Descripción de los comandos del editor de imágenes:

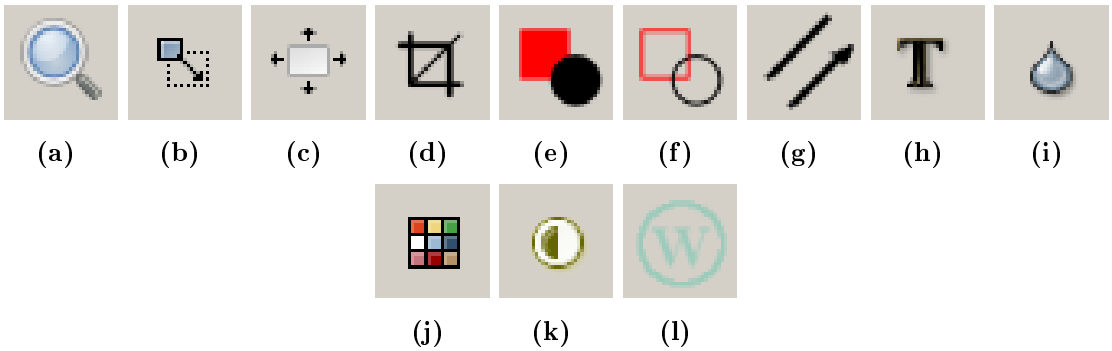


Figura 1.9: Comandos del editor de imágenes

- **(a) Ampliar imagen**: Acerca la imagen con el propósito de visualizarla. No cambia el tamaño de la imagen resultante.
- **(b) Escala de la imagen**: Escala la imagen con una cantidad dada. El escalado puede hacerse de forma independiente horizontal y verticalmente, o preservando la relación de aspecto.
- **(c) Modificar lienzo**: Cambia el tamaño y el color del lienzo de la imagen subyacente. Esta operación se puede utilizar para añadir bordes alrededor de la imagen original.
- **(d) Recortar la imagen**: Ajusta la imagen a un tamaño dado. El área de ajuste se selecciona cambiando el tamaño del área resaltada sobre la imagen.
- **(e) Agregar formas rellenas**: Agrega objetos rectangulares, cuadrados, elípticos o circulares a la imagen. Además de la forma se puede controlar, el color y la opacidad de los objetos.
- **(f) Agregar formas con bordes**: Similar a la operación anterior, excepto para objetos que contienen sólo bordes. Además se puede seleccionar el ancho del borde y el estilo.
- **(g) Añadir líneas o flechas**: Añade líneas con o sin punta de flecha a la imagen. Se puede modificar el color, ancho y estilo de las líneas.
- **(h) Añadir texto**: Añade un fragmento de texto con una fuente, un tamaño, un color y una opacidad predeterminados.
- **(i) Difuminar parte de la imagen**: Difumina la imagen de forma rectangular, cuadrada, elíptica o circular. La cantidad de desenfoque puede ser controlada.

- **(j) Pixeliza parte de la imagen**: Similar a la operación anterior, excepto que en lugar de difuminar, disminuye la resolución en el área seleccionada con el ratón.
- **(k) Actualizar parte de la imagen**: Similares a las dos anteriores, excepto que en lugar de difuminar o desenfocar, modifica el brillo de la zona seleccionada.
- **(l) Añadir marca de agua**: Agrega una marca de agua a la imagen. La marca de agua es un archivo de imagen cuya opacidad se puede modificar.

1.6.4. Vídeos

Se pueden exportar los vídeos procesados como vídeos re-codificados con la secuencia *File* → *Export* → *Encoded video*. La codificación se realiza usando el codificador *x264*. Los parámetros de codificación se definen mediante la selección de uno de los perfiles predefinidos para el parámetro de la lista. Los formatos disponibles para los vídeos codificados son "Matroska" (mkv), "MPEG-4 Part 14" (mp4) y "Flash" (FLV). Además, los vídeos codificados se pueden exportar dentro de un reproductor de vídeo de extracción automática (reproductor SFX), que reproduce automáticamente el vídeo incluido.

Figura 1.10: Ventana "Forevid SFX Player"

1.6.5. Resumen del proyecto

"Resumen del proyecto", ofrece un documento de registro para el proyecto actual. En él se enumeran los vídeos que forman parte del proyecto y las operaciones (y sus parámetros) que se aplican a cada vídeo. "Resumen del proyecto" es un archivo PDF que se genera seleccionando *File* → *Project summary*.

1.7. Otros

En esta sección se describen algunas de las características diversas de Forevid.

1.7.1. Editando vídeos

Aunque Forevid no está destinado para la edición de vídeo, ofrece algunas operaciones de edición simples. Por ejemplo, se pueden eliminar partes del vídeo, seleccionando el área que se desea eliminar con los botones "{" y "}" para designar el tramo y pulsar el botón "Cortar". Tras retirar la parte del vídeo, se puede cancelar la operación de la misma forma que se eliminan los filtros (*Video* → *Filters...* → *Delete*).

También se pueden fusionar los vídeos (*Video* → *Combine vídeos...*). La fusión se lleva a cabo mediante la adición de los vídeos en la lista "Source" de la línea de tiempo. Pero únicamente, sólo se pueden añadir los vídeos que comparten la misma resolución y velocidad de fotogramas.

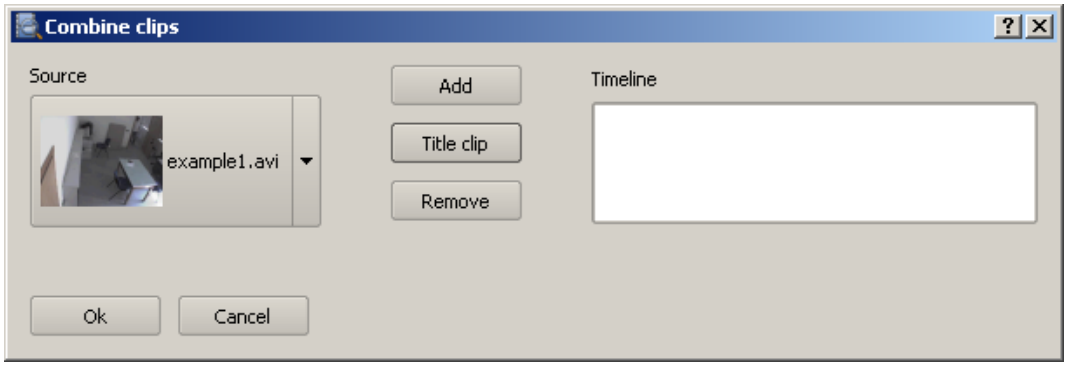


Figura 1.11: Ventana de edición "Combine clips"

Se pueden crear fácilmente vídeos titulados seleccionando "Title clip". Un vídeo titulado es un clip, en el que se puede definir el título (texto, color, tamaño), tipo (blanco, barras de colores) y la duración. Además, se necesita definir una plantilla para cada vídeo titulado. La resolución del vídeo titulado y la velocidad de fotogramas se basará en el formato del vídeo original.

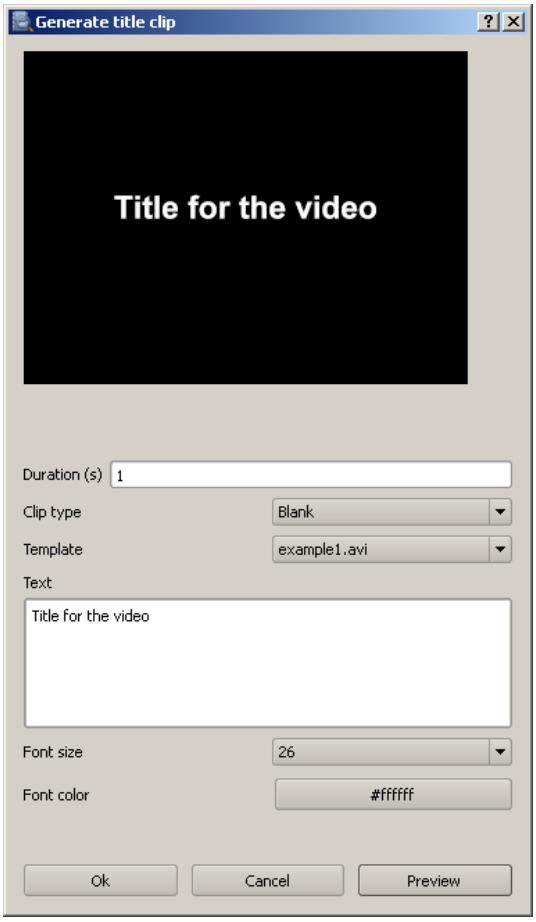


Figura 1.12: Ventana "Generate title clip"

1.7.2. Comparar el hash

Se puede utilizar el valor del hash para verificar que un archivo de vídeo no haya sido cambiado. Si se considera dicho archivo como un flujo de bits y le aplicamos un algoritmo de HASH, lo que se obtiene es otro conjunto de bits (de longitud fija y que depende del número de bits de salida del algoritmo o función que se utilice) que depende bit a bit del contenido del flujo original de bits que sirvió como entrada al algoritmo. Se puede calcular el hash de un archivo de vídeo determinado, seleccionando *Video* → *Hash...* A continuación, se pega el valor del anterior hash en el espacio de "Given value", se selecciona el algoritmo hash y se presiona "Compare". El campo Resultado dirá si los valores coinciden.

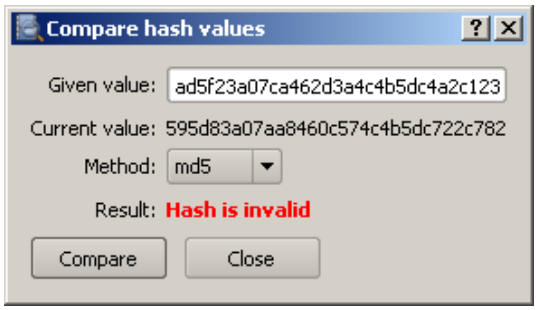


Figura 1.13: Ventana "Compare hash values"

Capítulo 2

Introducción a AviSynth

2.1. Introducción a AviSynth

El lenguaje AviSynth, fue creado para utilizarse como herramienta de *post producción* de vídeo, y proporciona una gran variedad de formas de edición y procesamiento de vídeos [1].

AviSynth es un programa libre y gratuito destinado a realizar *frameserver* y edición de vídeos. A diferencia de otros programas de edición de vídeo, AviSynth carece de interfaz gráfica desde donde ejecutar sus opciones y/o configurarlo. En su lugar, la configuración se programa enteramente sobre un archivo de texto sin formato, mediante el uso de un lenguaje propio de AviSynth. Aunque en un primer momento resulte complicado, es una herramienta extraordinariamente poderosa y muy buena para manejar proyectos de forma precisa y consistente. En cuanto a su ejecución, es en todo momento transparente para el usuario (salvo quizá, en casos de error en la configuración).

De forma muy básica y breve, el funcionamiento podría resumirse así: AviSynth toma un archivo de vídeo y/o audio descomprimido (la decodificación siempre es externa) y se ocupa de procesarlo en tiempo real mediante la aplicación de filtros (internos o externos) de muy variada índole, para luego devolverlo y que sea recibido por un programa compresor (encoder) o reproductor. AviSynth vendría a ser una especie de intermediario, y por lo tanto no solo requiere de algún programa anterior que le entregue el vídeo descomprimido (decodificadores y/o demultiplexores), sino también es necesario un programa posterior (encoder o reproductor) al cual se devuelve el vídeo filtrado.

Volviendo a su configuración, se dijo que se hacía a través de un archivo de texto usando el lenguaje propio de AviSynth. Es decir que se deben escribir las instrucciones indicando lo que queremos que haga. A este archivo de texto con las instrucciones se le llama "script", y puede ser creado con el bloc de notas de windows o similares, pero siem-

pre guardándolo sin formato ".txt". Para que AviSynth lo reconozca como un script, debe tener extension ".avs". También existen utilidades que permiten crear el script de forma mucho más cómoda e incluso interactiva, siendo lo más cercano a una interfaz gráfica que se puede encontrar, por ejemplo, así es como funciona el programa Forevid para la edición del procesado del vídeo analizado tal y como se verá más adelante. En el script se pueden indicar los filtros que se desean aplicar y su configuración, teniendo en cuenta que AviSynth respeta el orden del texto en su ejecución.

Este archivo ".avs", es el que luego debe ser cargado al compresor o reproductor, donde una vez abierto AviSynth, se encarga de que los mismos vean un archivo AVI con vídeo descomprimido, siendo ésta la función principal del programa, tomar un vídeo en algún formato cualquiera (como mínimo debe poder ser reproducido con los códecs instalados en el sistema) y entregarlo como vídeo AVI, función denominada *frameserver*.

El lenguaje de scripting es poderoso pero simple, y se pueden crear filtros complejos a partir de operaciones básicas para desarrollar una sofisticada paleta de efectos útiles y únicos.

2.2. Introducción al lenguaje

Como se mencionó anteriormente AviSynth se configura a partir de archivos de texto sin formato denominados "scripts". Aquí veremos las bases del lenguaje que se deben emplear en dichos scripts para poder indicar lo que se desea que haga AviSynth .

AviSynth lee scripts en orden, linea por linea de arriba a abajo, al igual que un libro. Esto se debe tener en cuenta cuando se crea un script, ya que los comandos y filtros serán ejecutados en este orden, y esto es de suma importancia como se comenta mas adelante. Es importante aclarar que AviSynth ignora espacios en blanco dejados entre comandos o filtros sucesivos, así como también mayúsculas y minúsculas.

Como crear un script en AviSynth

Se empezará con un script de AviSynth muy básico. Se abre cualquier editor de texto y se escribe lo siguiente (utilizando cualquier archivo AVI como por ejemplo "miclip.avi")

```
C:\folder\myclip.avi
```

A continuación se guarda como "miclip.avs". Ahora se tiene un script que puede ser abierto con la mayoría de reproductores de AVI como: *Windows Media Player 6.4* (conocido como "mplayer2"), *VirtualDubMod* o el *VirtualDub*.

2.3. Filtros

Se trata de comandos que indican a AviSynth que debe aplicar un determinado filtro en ese punto del script. Este comando está compuesto por el nombre del filtro seguido de dos paréntesis entre los que es posible dar valores personalizados a ciertos parámetros que regulan su configuración. Los tipos de filtros que se pueden encontrar en AviSynth son internos, externos y funciones. La sintaxis que utiliza AviSynth para llamar a los filtros es la siguiente:

```
NombreDelFiltro()
```

Al margen del orden en que AviSynth ejecuta el script, los filtros sucesivos pueden ir de forma completamente arbitraria, considerando que AviSynth ignora los espacios en blanco. Por ejemplo, el siguiente script:

```
AviSource("C:\video.avi")
Resize(1280,720)
FFT3DFilter()
```

Es exactamente lo mismo que:

```
AviSource("C:\video.avi")Resize(1280,720)FFT3DFilter()
```

Ambas formas son idénticas en funcionalidad, pero la primera resulta mucho mas ordenada para quien debe leer y trabajar con dicho script. Más adelante se verá un resumen más extenso de los filtros.

Parámetros:

La acción de un filtro se configura a partir de una serie de parámetros, que son valores propios de cada filtro en particular y utilizados internamente para cálculo y/o configuración. A estos parámetros se les debe asignar un cierto valor, según el efecto que dicho valor

tendrá en el funcionamiento del filtro, para ello, se debería consultar la ayuda del filtro en cuestión. Cabe aclarar que la enorme mayoría de los filtros los parámetros ya vienen con un valor asignado por defecto, en cuyo caso se verá que con solo aplicar el comando del filtro el script es funcional. Sin embargo, aunque existen filtros cuyos valores por defecto pueden ser aplicados a una gran variedad de situaciones, siempre es aconsejable ajustarlos al vídeo utilizado y así lograr los resultados óptimos. Dichos valores se pueden asignar de diferentes formas: en algunos filtros se debe conocer el orden en que se encuentran los parámetros dentro del paréntesis y los valores se les asignan en el mismo orden separados por comas.

```
STMedianFilter(a,b,c,d) -> STMedianFilter(8,15,4,7)
```

Esto puede aplicarse a todos los filtros. Sin embargo, en la gran mayoría se conoce el nombre de los parámetros y el filtro es capaz de reconocerlos, por lo que se les puede asignar su valor en cualquier orden mientras se nombren como se ve a continuación:

```
FFT3DFilter(sigma=2.0, bt=3, plane=0)
```

El ejemplo anterior es una forma mucho mas cómoda y versátil, ya que nos permite solo asignar valores a aquellos parámetros a los que se necesita modificar su valor por defecto.

Dentro de los paréntesis se pueden dejar todos los espacios que se deseen, pero si se quiere continuar en la siguiente linea se debe emplear el comando \, de la siguiente forma:

```
STMedianFilter(8,         15 \
,4,7)
```

Los valores que puede tomar un cierto parámetro dependen de su tipo, que puede ser:

- Integer - números enteros
- String - texto, que debe ser escrito entre comillas
- Real - números reales
- Boolean - puede ser verdadero (en cuyo caso se escribe True) o falso (en cuyo caso se escribe False), sin comillas

En el siguiente ejemplo se ve la aplicación de todos los tipos:

```
LSFmod(defaults="fast", strength=85, secure=false, ss_x=1.78)
```

AviSynth es capaz de trabajar con tres tipos diferentes de filtros:

- **Filtros Internos**: Son aquellos incluidos con AviSynth y pueden usarse directamente.
- **Filtros Externos o Plugins** [1]: Filtros no incluidos con AviSynth, también llamados plugins. Se implementa dentro de un archivo ".dll" aunque en algunos casos pueden requerir que otras librerías estén presentes en el sistema. Para poder invocar un plugin en el script (mediante su sintaxis correspondiente), primero debe cargarse el mismo utilizando la siguiente linea en el comienzo del script:

  ```
  LoadPlugin("C:\...\plugin.dll")
  ```

  o para que se cargue automáticamente, colocar el plugin en la carpeta por defecto:

  ```
  C:\Program Files\AviSynth 2.5\plugins
  ```

  En un mismo script se pueden utilizar tantos plugins como gusten, simplemente se debe usar una linea de código por cada uno para cargarlos, o ponerlos todos en la carpeta mencionada.
- **Funciones**: Para crear los scripts, AviSynth ofrece un lenguaje mucho mas complejo de lo que se ve aquí, donde solo nos limitamos a hacer llamadas a los filtros y el modo de aplicarlos y configurarlos. Es posible programar lo que se desee con unos determinados filtros, de una forma más compleja, con el objetivo de lograr mejores resultados propios. Estos scripts suelen contener la programación, en el lenguaje AviSynth, de lo que se acostumbra a llamar "función", es decir que en cada script de este tipo se define una función. La forma de trabajar con estas funciones es muy similar a como se trabaja con los plugins, hay que asegurarse de cargarlos en el script y luego se pueden invocar mediante su sintaxis como si de otro filtro se tratara. Es importante tener en cuenta que el script solo contiene una programación, pero no

[1] Este tipo de filtros es el que debe ser utilizado para crear un algoritmo e incluirlo como parte del programa Forevid. En la página 28 del capítulo 2.4 se implementará un filtro "Invertir".

puede filtrar por si mismo, para funcionar depende de los filtros internos o externos que se mencionan en las respectivas secciones del manual. Existen variedad de estos scripts, dedicados por ejemplo a aumentar la nitidez (efecto sharpen), filtrar ruido, filtrar bloques, etc. Como la función suele requerir de filtros externos se debe asegurar de cargarlos como corresponde o tenerlos en la carpeta por defecto. Respecto a la carga de la función misma, se realiza mediante la linea:

```
Import("C:\...\función.avs")
```

Para cargar automáticamente funciones, se pueden colocar en la carpeta por defecto junto al resto de los plugins, con la salvedad de poner extension .avs para que puedan ser reconocidas por AviSynth.

Orden de aplicación de los filtros

Es muy importante el orden en el que se aplican los diferentes filtros sobre el vídeo, pudiendo afectar de forma negativa a la calidad del vídeo en el caso de usarse en un orden incorrecto.

2.4. Ejemplo de como escribir un plugin para AviSynth en C++

En esta sección se explica como se escribe un plugin para AviSynth [6]. Para ello se utiliza el ejemplo del plugin "Invert" cuya función es invertir los canales de la imagen del vídeo que se reproduce.

Se requiere generar el archivo ".cpp" para crear el plugin a partir del algoritmo de programación del código del filtro. A continuación se muestra el código generado:

```
#include "avisynth.h"


class Invert : public GenericVideoFilter {
public:
    Invert(PClip _child) : GenericVideoFilter(_child) {}
    PVideoFrame __stdcall GetFrame(int n, IScriptEnvironment* env);
};
```

```
PVideoFrame __stdcall Invert::GetFrame(int n, IScriptEnvironment* env) {

    PVideoFrame src = child->GetFrame(n, env);
    PVideoFrame dst = env->NewVideoFrame(vi);

    const unsigned char* srcp = src->GetReadPtr();
    unsigned char* dstp = dst->GetWritePtr();

    const int src_pitch = src->GetPitch();
    const int dst_pitch = dst->GetPitch();
    const int row_size = dst->GetRowSize();
    const int height = dst->GetHeight();

    for (int y = 0; y < height; y++) {
        for (int x = 0; x < row_size; x++)
            dstp[x] = srcp[x] ^ 255;
        srcp += src_pitch;
        dstp += dst_pitch;
    }

    return dst;
}


AVSValue __cdecl Create_Invert(AVSValue args, void* user_data,
IScriptEnvironment* env) {
    return new Invert(args[0].AsClip());
}


extern "C" __declspec(dllexport) const char* __stdcall
AvisynthPluginInit(IScriptEnvironment* env) {
    env->AddFunction("Invertte", "c", Create_Invert, 0);
    return "‘Invert’ sample plugin";
}
```

Se compila el archivo en una dll a continuación y se le pone el nombre “Invert.dll”. Después se crea un script de AviSynth de la siguiente manera:

```
LoadPlugin(‘‘d:\path\Invert.dll’’)
clip = AVISource(‘‘d:\path2\video.avi’’)
return clip.Invert()
```

Como funciona

A continuación se procede a dar línea por línea una breve explicación de como funciona el código anterior:

```
#Include "avisynth.h"
```

Este encabezado declara toda la variedad de clases y constantes que se van a necesitar para escribir el plugin. Todos los plugins externos deben incluirlo.

Los plugins externos no se vinculan con avisynth.dll, por lo que no pueden acceder directamente a las funciones de AviSynth. Para solucionar esto, las funciones de AviSynth están definidas en línea o declaradas como virtuales.

```
class Invert : public GenericVideoFilter
```

Un filtro de AviSynth es simplemente una clase de C++ que implementa la interfaz *IClip*. *IClip*, tiene cuatro métodos virtuales puros: *GetVideoInfo*, *GetFrame*, *GetParity* y *GetAudio*.

La clase *GenericVideoFilter* hereda de *IClip* y es un simple filtro definido en avisynth.h que implementa vacíos los métodos virtuales puros de *IClip*. La mayoría de los filtros se pueden heredar de *GenericVideoFilter* en lugar de hacerlo directamente desde *IClip*, lo que evitaría tener que implementar métodos que no interesan, como *GetAudio*.

```
Invert(PClip _child) : GenericVideoFilter(_child)
```

PClip es una puntero inteligente a un objeto *IClip* que lleva la cuenta del número de punteros PClip que apuntan al objeto *IClip*, de manera que cuando el objeto deja de ser apuntado por un *PClip*, es eliminado. Siempre se debe utilizar un *PClip* para referenciar a los *IClip*.

Los filtros de AviSynth tienen un canal de salida normalizada por *IClip*, pero ningún canal de entrada estándar. Cada filtro es responsable de obtener su material de fuentes propias (mediante funciones como *GetFrame*) - generalmente de otro clip (como en este caso), pero a veces de varios clips diferentes, o de un archivo.

GenericVideoFilter tiene un solo constructor tomando un solo clip, que luego pasa simplemente a través de su salida. Vamos a reemplazar el método *GetFrame* para hacer algo más útil, dejando los otros tres métodos tal cual para saltarnos los aspectos del clip que no son necesarios cambiar.

```
PVideoFrame Invert::GetFrame(int n, IScriptEnvironment* env)
```

Este método es llamado para que el filtro reproduzca el fotograma "*n*" en su salida. El segundo argumento, "env", es para hacer una devolución de llamada (callback). En realidad, se implementa en AviSynth por una clase llamada *ScriptEnvironment*. Se crea una instancia de esta clase para cada script AVS abierto, por lo que a veces puede haber varias instancias activas a la vez. Es importante que las funciones de devolución de llamada se llamen a través de la instancia apropiada.

Este método devuelve un *PVideoFrame* y es un puntero inteligente como PClip a un objeto de la clase VideoFrame, en general, la imagen correspondiente una vez procesada por el filtro implementado.

```
PVideoFrame src = child->GetFrame(n, env);
```

El parámetro "child" es un miembro protegido de *GenericVideoFilter*, de tipo *PClip* y es normalmente el *PClip* que se paso en la iniciación del objeto. Contiene el clip que se ha pasado al constructor. Para producir el fotograma "*n*" es necesario el fotograma correspondiente de la entrada. Si se necesita un fotograma diferente en la entrada, todo lo que tenemos que hacer es pasar un número de fotograma distinto a $child \rightarrow GetFrame$.

```
PVideoFrame dst = env->NewVideoFrame(vi);
```

La devolución de llamada *NewVideoFrame* asigna espacio de memoria para un fotograma de vídeo de un tamaño determinado (este espacio de memoria contendrá la salida

de nuestro filtro). Este espacio de memoria no se inicializa excepto en la versión de depuración.

El parámetro "vi" es otro miembro protegido de *GenericVideoFilter*. Se trata de una estructura de tipo *VideoInfo*, que contiene información sobre el clip (como el tamaño del fotograma, formato de píxels, frecuencia de muestreo de audio, etc.) *NewVideoFrame* utiliza la información de "vi" para devolver un buffer de fotogramas de un tamaño apropiado.

Las memorias intermedias de fotogramas son reutilizadas una vez que todas las referencias *PVideoFrame* han desaparecido, así que, por lo general, *NewVideoFrame* en realidad no tendrá que reservar memoria de la pila.

```
const unsigned char* srcp = src->GetReadPtr()
unsigned char* dstp = dst->GetWritePtr();
```

Al igual que en *VirtualDub*, el "pitch" [2] de una memoria intermedia de fotogramas es el tamaño (en bytes) desde el comienzo de una línea de exploración hasta el comienzo de la siguiente. Los fotogramas de origen y de destino no tendrán el mismo "pitch" necesariamente. Los buffers creados por *NewVideoFrame* son siempre "quadword" (palabras cuádruples, 8-bytes) alineados y siempre tienen un paso que es un múltiplo de 8.

```
const int row_size = dst->GetRowSize();
```

El tamaño de fila es la longitud de cada fila en bytes (no en píxels). Desde nuestro origen y destino las imágenes tienen el mismo ancho y formato de píxel, y ademas siempre tendrán el mismo tamaño de la fila, así que sólo se necesita una variable "Row_Size", a la cual se podría haber llamado con *src* $\rightarrow$ *GetRowSize()*.

```
const int height = dst->GetHeight();
```

El "height" es la altura en píxels. Se debe tener en cuenta que es la misma para la fuente que para el destino, ya que, en este caso, la reserva de memoria para "dst" se hizo tomando como tamaño el de la imagen origen, que viene especificado en la variable "vi".

Las variables "RowSize" y "height" se emplean para controlar el bucle que recorre la imagen.

[2] Tamaño de memoria que ocupa una linea de un fotograma.

```
# El siguiente bucle invierte los colores hacendo una XOR bit a bit
    for (int y = 0; y < height; y++) {
        for (int x = 0; x < row_size; x++)
            dstp[x] = srcp[x] ^ 255;
        srcp += src_pitch;
        dstp += dst_pitch;
    }

    return dst;
}
```

Este es el código que realmente hace el trabajo.

GetFrame devuelve el nuevo fotograma creado. Nuestra referencia a este fotograma se irá con la variable "dst". Esta llamada se convertirá en la única propietaria del fotograma destino (en el cual todavía se podrá escribir), y el fotograma de origen se retendrá en la memoria cache y finalmente se reciclará, todo gracias a los punteros inteligentes de C++ que hemos empleado en el desarrollo del código.

```
AVSValue__cdecl Create_Invert(AVSValue args, void* user_data,
IScriptEnvironment* env)
```

Para utilizar el nuevo filtro se necesita una función de lenguaje de script que cree una instancia de la misma. La función mostrada arriba es la que se utiliza en este caso.

Las funciones de script escritas por C++ toman tres argumentos, "args" contiene todos los argumentos pasados por el script, "user_data" contiene el puntero vacío que se le pasa a *AddFunction* (normalmente no se necesita), y "Env" contiene el mismo puntero *IScriptEnviroment* que se pasa mas tarde a *GetFrame*.

AVSValue es un tipo de variante que puede contener cualquiera de los valores siguientes: un valor booleano (true o false), un número entero, un número de coma flotante, una cadena, un vídeo clip (PClip), un conjunto de *AVSValues*, o nada (undefined). Se puede comprobar cuál es con los métodos *IsBool()*, *IsInt()*, *IsFloat()*, *IsString()*, *IsClip()*, *IsArray()*, y *Defined()* (que devuelve true si el AVSValue no es "undefined"). Se puede obtener el valor con *AsBool()*, *AsInt()*, etc. Para las matrices, se puede utilizar *ArraySize()* para obtener el número de elementos, y mediante el operador de indexación [] para obtener los elementos mismos. Para mayor comodidad, *IsFloat()* y *AsFloat()* funcionarán también con números enteros, pero los valores booleanos no son tratados como numéricos.

El nombre de "Create_Invert" es arbitrario. Esta función realmente se conoce como "Invert" en los scripts, porque ese es el nombre que se pasa a *AddFunction* abajo.

```
return new Invert(args[0].AsClip());
```

El argumento "args" pasado a una función de script siempre será una matriz. El valor de retorno debe ser de cualquiera de los otros tipos (nunca una matriz).

Los tipos de los valores de la matriz "args" están garantizados para que coincidan con una de las firmas de función que se pasan a *AddFunction*, al igual que en *VirtualDub*. Por lo tanto, no hay necesidad de preocuparse por *IsClip* aquí.

Create_ Invert simplemente crea y devuelve una instancia del filtro, que es automáticamente convertida en un *AVSValue* a través del constructor *AVSValue (IClip *)*.

```
extern "C" __declspec(dllexport) const char* __stdcall
AvisynthPluginInit(IScriptEnvironment* env)
```

Esta es la única función que se exporta desde el DLL. Es llamada por la función del script *LoadPlugin* la primera vez que se carga este plugin en un script en particular. Si se encuentran varios scripts abiertos a la vez y más de uno de ellos carga este plugin, *AvisynthPluginInit* debe ser llamado más de una vez con *IScriptEnvironment* diferentes. Por lo tanto:

- No se puede salvar el parámetro "env" en una variable global.
- Si se necesita inicializar algún dato estático, se debe hacer en el *DLLmain* (principal), no en esta función. La función *DLLmain*, es la función que se ejecuta al cargar una dll.
- Mientras que *AvisynthPluginInit* se utiliza para crear el DLL (utilizando el *IScriptEnvironment* actual), *DLLmain* se utiliza para cargarlo.

El propósito principal de la función *AvisynthPluginInit* es añadir las funciones del filtro creado para que puedan ser accedidas desde el script a traves de:

```
env->AddFunction("Invertte", "c", Create_Invert, 0);
```

Según lo dicho, ahora se llama a *AddFunction* para hacer saber de la existencia de nuestro filtro a AviSynth. Esta función toma cuatro argumentos: el nombre de la función del nuevo script, la cadena del tipo de parámetros, la función C++ (que implementa la función de script), y la cookie user_data.

```
return " 'Invert' sample plugin";
```

El valor de retorno de *AvisynthPluginInit* es una cadena que puede contener cualquier mensaje que se desee, como una nota de la identificación de la versión y el autor del plugin. Esta cadena se convierte en el valor de retorno de *LoadPlugin*, y casi siempre sera ignorado. También se puede simplemente devolver 0 si se prefiere.

Se compila el proyecto, para crear un archivo dll (en el capítulo 5.1 se ilustra un ejemplo utilizando VisualStudio). El filtro ya puede ser llamado desde un script de AviSynth.

2.4.1. Añadir argumento a un filtro de un plugin AviSynth

Para explicar cómo se puede añadir un argumento a un filtro, se utiliza el código del filtro "Invert" y se cambia, incorporando un parámetro que se puede modificar cada vez que hagamos una llamada a la función que invoca el filtro desde cualquier script de AviSynth, a continuación se muestra el código:

```
#include "avisynth.h"

class Invert : public GenericVideoFilter {
private:
int nuevo_atributo;
public:
    Invert(PClip _child, int valor_del_slide) :
    GenericVideoFilter(_child)
    {
nuevo_atributo = valor_del_slide;
    }

    PVideoFrame __stdcall GetFrame(int n, IScriptEnvironment* env);
};
```

```
PVideoFrame __stdcall Invert::GetFrame(int n, IScriptEnvironment* env) {

    PVideoFrame src = child->GetFrame(n, env);
    PVideoFrame dst = env->NewVideoFrame(vi);

    const unsigned char* srcp = src->GetReadPtr();
    unsigned char* dstp = dst->GetWritePtr();

    const int src_pitch = src->GetPitch();
    const int dst_pitch = dst->GetPitch();
    const int row_size = dst->GetRowSize();
    const int height = dst->GetHeight();

    for (int y = 0; y < height; y++) {
        for (int x = 0; x < row_size; x++)
            dstp[x] = srcp[x] ^ nuevo_atributo;
        srcp += src_pitch;
        dstp += dst_pitch;
    }

    return dst;
}


AVSValue __cdecl Create_Invert(AVSValue args, void* user_data,
IScriptEnvironment* env) {
return new Invert(args[0].AsClip(), args[1].AsInt());
}


extern "C" __declspec(dllexport) const char* __stdcall
AvisynthPluginInit2(IScriptEnvironment* env) {
    env->AddFunction("Invertte", "ci", Create_Invert, 0);
    return "‘Invert’ sample plugin";
}
```

En la clase “Invert” se crea un nuevo atributo[3] *nuevo_ atributo* y al constructor de la clase se le añade un nuevo parámetro *valor_ del_ slide*, que actualizará el valor del nuevo atributo creado (*nuevo_ atributo = valor_ del_ slide*), como se muestra mas abajo:

[3]Un atributo es una variable dentro de una clase

```
class Invert : public GenericVideoFilter {
private:
int nuevo_atributo;
public:
    Invert(PClip _child, int valor_del_slide) :
    GenericVideoFilter(_child) {
nuevo_atributo = valor_del_slide;
    }

    PVideoFrame __stdcall GetFrame(int n, IScriptEnvironment* env);
};
```

Se incluye en el código el parámetro que se pasa al filtro. En este caso sustituimos el antiguo *255*, que estaba multiplicando en la "XOR", por el atributo privado (*private:*) *nuevo_ atributo*, que tomará su valor con la llamada al filtro.

```
    for (int y = 0; y < height; y++) {
        for (int x = 0; x < row_size; x++)
            dstp[x] = srcp[x] ^ nuevo_atributo;
        srcp += src_pitch;
        dstp += dst_pitch;
    }
```

En el código a continuación, se pone el nuevo parámetro *args[1].AsInt()*, que previamente se añade al constructor de la clase *Invert*.

```
AVSValue __cdecl Create_Invert(AVSValue args, void* user_data,
IScriptEnvironment* env) {
return new Invert(args[0].AsClip(), args[1].AsInt());
}
```

Por último se cambia el segundo parámetro de la función *AddFunction()*, *"ci"*, que indica los parámetros que recibirá la función *Create_ Invert*. Estos parámetros pueden ser de varios tipos, como por ejemplo clip "c", entero "i" o float "f". En este caso la cadena *"ci"* indica que tendrá dos parámetros, uno de tipo clip y otro de tipo int, que se pasarán a la función *Create_ Invert*.

```
extern "C" __declspec(dllexport) const char* __stdcall
AvisynthPluginInit2(IScriptEnvironment* env) {
    env->AddFunction("Invertte", "ci", Create_Invert, 0);
    return "‘Invert’ sample plugin";
}
```

Una vez compilado el archivo en una dll, se crea un script de AviSynth (como se indica en el capítulo 5.1 en la página 88) de la siguiente manera:

```
LoadPlugin("C:\path\InvertDll.dll")
clip = AVISource("C:\path2\example1.avi")
return clip.Invertte(255)
```

Capítulo 3

Introducción a PyQt4

El programa Forevid esta implementado en PyQt4, y en este apartado se hace una breve introducción de cómo funciona para poder entender mejor el código del programa, ya que la mayor parte de las clases de éste, heredan de otras clases de Qt [4].

Este capítulo trata de explicar como funciona de la manera mas breve y sencilla posible, es decir, a base de sencillos ejemplos que ayudan a comprender como funciona Qt de forma general [2].

3.1. Crear una ventana

Este ejemplo de código es muy sencillo, sólo muestra una pequeña ventana. Sin embargo, se puede hacer mucho con esta ventana, como cambiar su tamaño, maximizar, minimizar, etc. Esto requeriría una gran cantidad de codificación, pero por suerte, esta funcionalidad ya ha sido codificada. Como se repite en la mayoría de las aplicaciones, no es necesario que el código se redacte de nuevo. PyQt4 es un conjunto de herramientas de alto nivel, y si se quisiera codificar en un conjunto de herramientas de nivel inferior, el ejemplo de código siguiente podría tener docenas de líneas.

```
import sys
from PyQt4 import QtGui


def main():

    app = QtGui.QApplication(sys.argv)

    w = QtGui.QWidget()
    w.resize(250, 150)
    w.move(300, 300)
    w.setWindowTitle('Simple')
    w.show()

    sys.exit(app.exec_())


if __name__ == '__main__':
    main()
```

El código python de arriba muestra una pequeña ventana en la pantalla. A continuación se explica cómo ha sido programado.

```
import sys
from PyQt4 import QtGui
```

Aquí se proporcionan los archivos necesarios que se tienen que importar. Los widgets básicos de la GUI (Graphic User Interface) se encuentran localizados en el módulo *QtGui*.

```
app = QtGui.QApplication(sys.argv)
```

Cada aplicación PyQt4 debe crear un objeto de aplicación. El objeto de la aplicación se encuentra en el módulo de *QtGui*. El parámetro "sys.argv" es la lista de argumentos desde la línea de comandos.

```
w = QtGui.QWidget()
```

El widget *QtGui.QWidget* es la clase base de todos los objetos de interfaz de usuario en PyQt4. Aquí se proporciona el constructor predeterminado para *QtGui.QWidget*. El constructor por defecto no tiene padre. Un widget sin padre se llama "ventana".

```
w.resize(250, 150)
```

El método *resize* dimensiona la ventana del widget. En este caso le proporciona un ancho de 250 pixels y un alto de 150 pixels.

```
w.move(300, 300)
```

El método *move* posiciona el widget en el lugar de la pantalla que deseemos. En este caso en la posición x=300 e y=300.

```
w.setWindowTitle('Simple')
```

Aquí se proporciona un título a nuestra ventana que en este caso es el texto "Simple".

```
w.show()
```

El método *show* muestra la ventana en la pantalla.

```
sys.exit(app.exec_())
```

Por último, se entra en el bucle principal de la aplicación. La gestión de eventos comienza a partir de este punto. El mainloop (bucle pricipal) recibe eventos del sistema de ventanas y los envía a los controles de la aplicación. El mainloop termina, si se llama a *exit()* o el widget principal es destruido. El método *sys.exit()* asegura una salida limpia. El entorno será informado, de cómo termina la aplicación.

Figura 3.1: Ejemplo ventana "Simple"

Cuando un módulo en Python (archivo .py) es llamado para ser ejecutado, en vez de para importar su funcionalidad con "import", la variable $\_\_name\_\_$ contendrá el valor $\_\_main\_\_$. Esto se utiliza para saber si un módulo en Python (archivo .py) es llamado para ser ejecutado, o para importar su funcionalidad con "import".

Ventana de mensaje

Por defecto, si se hace clic en el botón "X" en la barra de título, la *QtGui.QWidget* se cierra automáticamente. En ocasiones es preferible modificar este comportamiento predeterminado. Por ejemplo, si se tiene un archivo abierto en un editor en el que se han realizado cambios, se mostrará un cuadro de mensaje para confirmar la acción, y así permitir guardar o cancelar los cambios realizados. Abajo se ve un ejemplo de este tipo de cuadros de diálogo, y se explica cómo funciona en lineas generales.

```
import sys
from PyQt4 import QtGui


class Example(QtGui.QWidget):

    def __init__(self):
    # Este es el constructor de la clase Example
        super(Example, self).__init__()
        # Llamada al constructor de la clase de la
        #que hereda Example(QtGui.QWidget)

        self.initUI()
        # Llamada a la función initUI(self) de la clase Example

    def initUI(self):
    # Descripción de los parámetros de la ventana de diálogo

        self.setGeometry(300, 300, 250, 150)
        self.setWindowTitle('Message box')
        self.show()

  # Codigo que genera la ventana de la siguiente imagen
    def closeEvent(self, event):

        reply = QtGui.QMessageBox.question(self, 'Message',
            "Are you sure to quit?", QtGui.QMessageBox.Yes |
            QtGui.QMessageBox.No, QtGui.QMessageBox.No)

        if reply == QtGui.QMessageBox.Yes:
            event.accept()
        else:
            event.ignore()
```

```
def main():

    app = QtGui.QApplication(sys.argv)
    ex = Example()
    sys.exit(app.exec_())
```

Si se cierra el *QtGui.QWidget*, se genera el evento *QtGui.QCloseEvent* que cierra la ventana. Para modificar este comportamiento en el widget, necesitamos volver a implementar el controlador de eventos *closeEvent()*.

```
reply = QtGui.QMessageBox.question(self, 'Message',
    "Are you sure to quit?", QtGui.QMessageBox.Yes |
    QtGui.QMessageBox.No, QtGui.QMessageBox.No)
```

Se muestra un cuadro de mensaje con dos botones, "Sí" y "No". El primer argumento que se muestra en la función es para el título de la ventana. El segundo argumento es el texto del mensaje que muestra el cuadro de diálogo. El tercer argumento especifica la combinación de botones que aparecen en el cuadro de diálogo. El último parámetro es el botón predeterminado que está inicialmente seleccionado por teclado. El valor de retorno se guarda en la variable "reply".

```
if reply == QtGui.QMessageBox.Yes:
    event.accept()
else:
    event.ignore()
```

Aquí se chequea el valor de retorno. Si se hace clic en el botón "Sí", se cierra la ventana y finaliza la aplicación. De lo contrario "close", impide que se cierre la ventana, a pesar de que el usuario intentó cerrarla pulsando el botón "x".

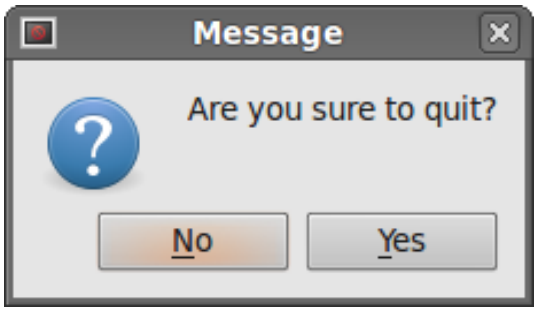


Figura 3.2: Ejemplo ventana "mensaje"

3.2. Ventana principal (barras de estados, de menú y de herramientas)

La clase *QtGui.QMainWindow* proporciona una ventana de la aplicación principal, esto permite crear el esqueleto de la aplicación clásica con una barra de estado, barras de herramientas y una barra de menú.

Aquí tenemos un ejemplo de código que genera las tres barras principales que se utilizan en la mayor parte de los programas, y como no en Forevid.

```
import sys
from PyQt4 import QtGui

class Example(QtGui.QMainWindow):

    def __init__(self):
        super(Example, self).__init__()

        self.initUI()

    def initUI(self):

        textEdit = QtGui.QTextEdit()
        self.setCentralWidget(textEdit)

        exitAction = QtGui.QAction(QtGui.QIcon('exit24.png'),
        'Exit', self)
        exitAction.setShortcut('Ctrl+Q')
        exitAction.setStatusTip('Exit application')
        exitAction.triggered.connect(self.close)
```

```
        self.statusBar()

        menubar = self.menuBar()
        fileMenu = menubar.addMenu('&File')
        fileMenu.addAction(exitAction)

        toolbar = self.addToolBar('Exit')
        toolbar.addAction(exitAction)

        self.setGeometry(300, 300, 350, 250)
        self.setWindowTitle('Main window')
        self.show()


def main():

    app = QtGui.QApplication(sys.argv)
    ex = Example()
    sys.exit(app.exec_())


if __name__ == '__main__':
    main()
```

Para empezar hablaremos de la barra de estado. La barra de estado es creada con la ayuda del widget *QtGui.QMainWindow*.

```
self.statusBar()
```

Para obtener la barra de estado, llamamos al método *statusBar()* de la clase *QtGui. QMain- Window*. La primera llamada del método crea una barra de estado, las llamadas posteriores devuelven el objeto de la barra de estado. Si escribimos *self.statusBar(). showMessage('Ready')* muestra el mensaje "Ready" en la barra de estado.

Una **barra de menú** es una parte común de una aplicación GUI. Se trata de un grupo de comandos situados en varios menús. Mientras que en las aplicaciones de consola tenemos que recordar varios comandos y sus opciones, aquí tenemos la mayoría de los comandos agrupados en partes lógicas. Estos son los estándares aceptados que reducen aún más la cantidad de pérdida de tiempo para aprender una nueva aplicación.

En el ejemplo de ventana principal, se crea una barra de menús con un menú. Este menú contiene una sola acción, cuya función es poner fin a la aplicación si se selecciona. Se puede acceder a la acción con las teclas "Ctrl + Q".

```
exitAction = QtGui.QAction(QtGui.QIcon('exit24.png'), 'Exit', self)
exitAction.setShortcut('Ctrl+Q')
exitAction.setStatusTip('Exit application')
```

Una "QtGui.QAction" es una abstracción de las acciones realizadas con una barra de menú, barra de herramientas o con un atajo de teclado personalizado. En las tres líneas, se crea una acción, con un icono específico y una etiqueta de "Exit". Por otra parte, se define un acceso directo para esta acción. La tercera línea crea una pauta de estado, que se muestra en la barra de estado, cuando pasamos el puntero del ratón sobre el elemento del menú.

```
exitAction.triggered.connect(QtGui.qApp.quit)
```

Al seleccionar esta acción en particular, se emite una señal de activación. La señal está conectada al método *Quit()* del widget *QtGui.QApplication*. Esto finaliza la aplicación.

```
menubar = self.menuBar()
fileMenu = menubar.addMenu('&File')
fileMenu.addAction(exitAction)
```

El método *menuBar()* crea la barra de menú. Se crea un menú y se añade la acción de salir al mismo.

Las barras de herramientas proporcionan un acceso rápido a los comandos más utilizados en los programas. En el ejemplo se crea una muy sencilla y con una sola acción definida. La acción es una función exit que finaliza la aplicación cuando se presiona.

```
self.toolbar = self.addToolBar('Exit')
self.toolbar.addAction(exitAction)
```

Aquí se crea la barra de herramientas con un objeto que aplica una acción dentro de ella.

```
textEdit = QtGui.QTextEdit()
self.setCentralWidget(textEdit)
```

Aquí se crea un widget de edición de texto. Lo hemos transformado en el widget central de la *QtGui.QMainWindow*. El widget central ocupará todo el espacio que falte por ocupar. El resultado gráfico que obtendríamos del código es el siguiente:

3.3. Eventos

Todas las interfaces gráficas son manejadas por eventos. Los eventos son disparados principalmente por el usuario de la aplicación, pero también pueden ser generados por otros como: conexiones a internet, gestor de ventanas, timer, etc.

Figura 3.3: Ejemplo ventana principal “Main indow”

Señales y slots

Las señales y slots (destino de la señal) se utilizan para comunicar objetos entre ellos. Cuando ocurre un evento se emite una señal. Los slots pueden ser cualquier cosa que pueda ser llamada en Python. Por lo que un slot es llamado cuando se emite una señal conectada a él. Para que quede mas claro se puede ver el siguiente ejemplo:

```
class Example(QtGui.QWidget):

    def __init__(self):
        super(Example, self).__init__()

        self.initUI()

    def initUI(self):

        #Se crean los objetos, pantalla lcd y barra sld
        lcd = QtGui.QLCDNumber(self)
        sld = QtGui.QSlider(QtCore.Qt.Horizontal, self)

        vbox = QtGui.QVBoxLayout()
        vbox.addWidget(lcd)
        vbox.addWidget(sld)

        self.setLayout(vbox)
        sld.valueChanged.connect(lcd.display)
```

```
        self.setGeometry(300, 300, 250, 150)
        self.setWindowTitle('Signal & slot')
        self.show()
```

En este ejemplo se muestran los widgets *QtGui.QLCDNumber* y *QtGui.QSlider*. La pantalla "lcd" cambia cuando se arrastra la barra del objeto "sld" mediante el siguiente comando:

```
#Conexion de una señal de barra deslizante con el slot lsd
sld.valueChanged.connect(lcd.display)
```

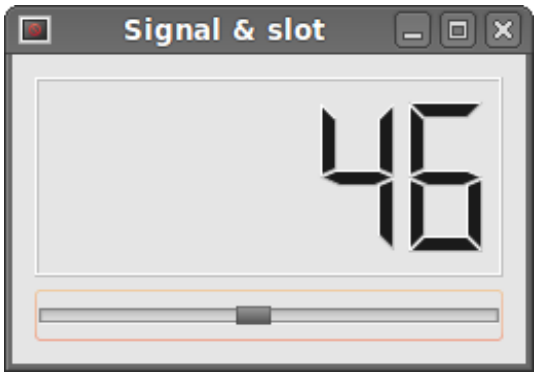


Figura 3.4: Ejemplo de señal y slot

El que envía la señal es un objeto (*QSlider*). El slot es el método que reacciona con la señal y el destino de la señal es el objeto *lcd.display*.

Eventos desde el teclado

Este ejemplo muestra como provocar un evento desde el teclado.

```
class Example(QtGui.QWidget):

    def __init__(self):
        super(Example, self).__init__()

        self.initUI()

    def initUI(self):

        self.setGeometry(300, 300, 250, 150)
        self.setWindowTitle('Event handler')
```

```
        self.show()

#Este método se utiliza para generar los eventos desde el teclado
    def keyPressEvent(self, e):

        #Enlaza la tecla Escape con la función close().
        if e.key() == QtCore.Qt.Key_Escape:
            self.close()
```

Cuando presionamos la tecla "Escape" se cierra la ventana en la que se encuentra el programa.

Botones para enviar eventos

En el ejemplo a continuación se crea una ventana con dos botones que generan eventos.

```
class Example(QtGui.QMainWindow):

    def __init__(self):
        super(Example, self).__init__()

        self.initUI()

    def initUI(self):

#Se crean dos botones y se situan en la ventana
        btn1 = QtGui.QPushButton("Button 1", self)
        btn1.move(30, 50)

        btn2 = QtGui.QPushButton("Button 2", self)
        btn2.move(150, 50)

       #Se enlazan los eventos de ambos botones al método buttonClicked
        btn1.clicked.connect(self.buttonClicked)
        btn2.clicked.connect(self.buttonClicked)

        #Se genera una barra de estado en la ventana
        self.statusBar()

        #Se dimensiona la ventana, se titula y se muestra en pantalla
        self.setGeometry(300, 300, 290, 150)
        self.setWindowTitle('Event sender')
```

```
        self.show()

    def buttonClicked(self):

        sender = self.sender()
        self.statusBar().showMessage(sender.text() + ' was pressed')
```

Los dos botones están conectados al mismo slot *buttonClicked()*. Dentro del método *buttonClicked()* se determina que botón ha sido pulsado a través del método *sender()*. El botón seleccionado se muestra en la barra de estado.

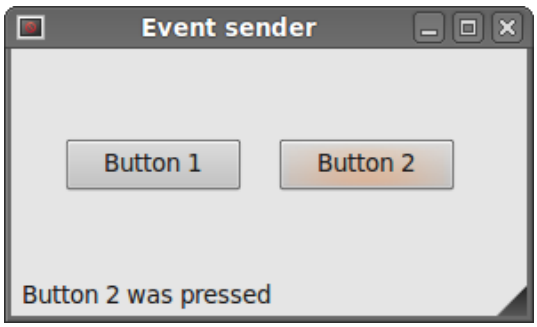


Figura 3.5: Ejemplo de ventana con eventos

Capítulo 4

Estructura del programa Forevid

Para empezar a entender como se ha elaborado el proyecto, en esta sección se describirá exclusivamente una introducción al código fuente (source code) del programa Forevid, en la cual se explican, los módulos, las funciones, las clases y los métodos, más necesarios, para comprender el desarrollo del trabajo de investigación realizado. El programa ha sido desarrollado en el lenguaje Python, y se explicara con el mismo esquema (en función de los módulos que se crearon) que utilizaron los desarrolladores de Forevid al crearlo [7].

4.1. AviSynth.py

Este modulo se usa para organizar las clases, constantes y variables que el programa requiere para el manejo de AviSynth en Forevid. En primer lugar se crea un objeto *avidll*, en el que se encuentra la librería "avs_forevid.dll" que contiene toda la funcionalidad de AviSynth. De esta librería se utilizan todas la funciones que aparecen definidas al final del módulo.

```
avidll = ctypes.windll.LoadLibrary("avs\_forevid.dl")
```

A continuación se describirán las clases más destacadas dentro del mismo.

PIScriptEnviroment

Esta clase es la encargada de manejar los scripts de AVS. Ejecuta los fragmentos del script haciendo de intermediario entre python y AviSynth. La mayoría de los métodos que utiliza se encuentran dentro de la funcionalidad de AviSynth para poder ser utilizados en el propio Python. A continuación se exponen algunos ejemplos de estos métodos:

- El método *FunctionExists* indica si la función introducida en el script existe para AviSynth o no.

- El método *AddFunction* es usado para añadir funciones que tiene que ver con los filtros, como por ejemplo, algunas de las funciones de "resize" o la función "blur".

VideoInfo y PVideoInfo

La clase *VideoInfo* tiene una serie de campos con información del vídeo, como el ancho, el tamaño, el alto, el autor, etc.

```
class VideoInfo(ctypes.Structure):
    _fields_ = [("width",ctypes.c_int),
                ("height",ctypes.c_int),
                ("fps_numerator",ctypes.c_uint),
                ("fps_denominator",ctypes.c_uint),
                ("num_frames",ctypes.c_int),
                ("pixel_type",ctypes.c_int),
                ("audio_samples_per_second",ctypes.c_int),
                ("sample_type",ctypes.c_int),
                ("num_audio_samples",ctypes.c_int64),
                ("nchannels",ctypes.c_int),
                ("image_type",ctypes.c_int)]
```

clase *VideoInfo*

Por otro lado los objetos creados por la clase *PVideoInfo* se usan para crear punteros que apuntan a la misma (esta clase envuelve a la clase *VideoInfo*) y tiene una serie de métodos que se utilizan para extraer información del vídeo.

VideoFrameBuffer, VideoFrame y PVideoFrame

VideoFrameBuffer y *VideoFrame* solamente contienen campos con información del vídeo y de sus frames.

La clase *PVideoFrame* envuelve a *VideoFrame* y se utiliza para acceder a la información de esa clase, de modo que se puede modificar, copiar el vídeo, borrarlo, escribir, etc.

AVS_Value

Facilita el acceso a los tipos de datos que están en el archivo DLL (que son del tipo C) y comprueba de qué tipo son los mismos (string, integer, float, boolean, etc.). Tiene métodos para manejar y modificar el valor de las variables.

BITMAPINFOHEADER

Esta clase maneja todo lo relacionado con los mapas de bits del vídeo que se está manejando en ese momento, como por ejemplo el tamaño de la imagen, el ancho, el largo, el número de bits, si está comprimida o no, etc. En el siguiente ejemplo se muestran algunas de las variables que se declaran en la clase.

```
class BITMAPINFOHEADER(ctypes.Structure):
    _fields_ = [("biSize",  ctypes.c_ulong),
                ("biWidth",   ctypes.c_long),
                ("biHeight",   ctypes.c_long),
                ("biPlanes",   ctypes.c_ushort),
                ("biBitCount",   ctypes.c_ushort),
                ("biCompression",  ctypes.c_ulong),
                ("biSizeImage",  ctypes.c_ulong),
                ("biXPelsPerMeter",   ctypes.c_long),
                ("biYPelsPerMeter",   ctypes.c_long),
                ("biClrUsed",  ctypes.c_ulong),
                ("biClrImportant",  ctypes.c_ulong)]
```

clase *BITMAPINFOHEADER*

4.2. Avs_Filter.py

Este módulo se encarga de la organización de los filtros y de sus definiciones. Importa funcionalidades de PyQt4 que es una librería Qt adaptada al lenguaje de Python. Para comprender mejor de qué trata, se explica de forma breve cada una de sus clases.

Avs_parameters

Esta clase guarda la información de los parámetros que usan los filtros, como son el nombre, el tipo de datos que se pasan al filtro (float, integer, text, bool...), si es visible o no, el valor de los parámetros del filtro, el valor mínimo y máximo de sus parámetros y si es modificable o no.

Avs_filter

Es la clase que representa al filtro en sí. Cada filtro se guarda en un objeto de esta clase. Dentro de la clase se guarda información del nombre del filtro, el comando que lo ejecuta, los parámetros del filtro, su descripción, el grupo al que pertenece, el color (espacio de colores definido) y si es visible o no. La clase está compuesta únicamente por los siguientes métodos:

- *convertColor* es un método que realiza conversiones de espacios de colores.
- *generate* genera un comando compatible con el correspondiente en AviSynth que hace esa función. Devuelve una cadena de texto que AviSynth interpreta directamente, por ejemplo, si se recorta parte de la escena de vídeo, genera el código AviSynth correspondiente.

```
AVISource("somevideo.avi")
Trim(0, 12000) ++ Trim(20000, 32000) ++ Trim(44000, 0)
```

La función *Trim* especifica los fotogramas del video original que se desean. En este caso solo se visualizaran los fotogramas [0, 12000], [20000, 32000] y [44000, hasta el final].

Avs_FilterChain

Solamente se encarga de guardar las cadenas de texto que han sido aplicadas al vídeo, siempre en orden cronológico.

Avs_Filters

Por último, este método contiene una lista de todos los filtros de los que dispone el programa para ser utilizados. Si se quisiera añadir un filtro nuevo, habría que declararlo aquí entre otros lugares.

4.3. Dialogs.py

Este módulo tiene una función muy básica, que es crear en "Qt" algunas de las ventanas de dialogo que aparecen en el programa. En el modulo se han definido una serie de clases y cada una de ellas se encarga de una ventana de dialogo distinta. En este apartado se da una breve explicación de cada una de ellas:

BookMarkDialog

"BookMarkDialog" es la ventana de dialogo que aparece cuando se pulsa *Avtions* → *Edit or Add bookmark*, o bien cuando hacemos click con el botón derecho sobre una bookmark. Hereda del tipo QDialog y su programación es simple.

Se crean dos objetos en el método *__init__*, *QPushButton* para el boton de "Ok" y de "Cancel" y se les asocian las funciones correspondientes con los métodos *okPressed* y *cancelPressed*, y el ultimo método *keyPressEvent*, se utiliza simplemente para darle funciones al teclado, como por ejemplo, al pulsar la tecla "Esc" se saldría de la ventana.

```
class bookmarkDialog(QDialog):
    def __init__(self, parent = None,  comment = ''):
        QDialog.__init__(self,parent)

        self.setWindowTitle(self.tr('Add bookmark'))

        self.ok = QPushButton("Ok")
        self.cancel = QPushButton(self.tr("Cancel"))
        self.text = QTextEdit()
        self.text.insertPlainText(comment)
         .
         .
         .
    def okPressed(self):

        self.comment =  unicode(self.text.toPlainText())
        self.accept()

    def cancelPressed(self):
        self.reject()

    def keyPressEvent(self, event):
        if event.key() == Qt.Key_Return:
            self.okPressed()
        if event.key() == Qt.Key_Escape:
            self.cancelPressed()
```

Ventana BookMarkDialog

MediaInfoBox

Otro ejemplo de ventana de dialogo es la clase *MediaInfoBox*. Esta clase es llamada cuando se selecciona la ventana de información (*Video* → *Media info*).

Lo que se crea también, es un objeto de esta clase que hereda de las clase *QDialog* de Qt. En esta ventana solo se crea el botón de "Ok" y se permiten acciones por teclado también ("Esc").

Otras

Todas las demás clases del módulo *dialogs* (*LoaderBox*, *LoaderBox*, *ExportImages-Dialog*, *SettingsDialog*, *AboutDialog*, *MessageBoxCheck*, *RecordScreenDialog*) funcionan

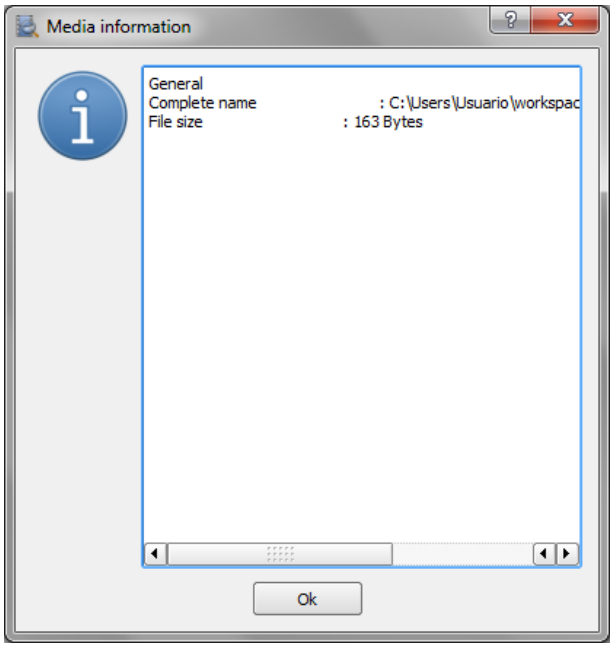


Figura 4.1: Ventana para crear o modificar bookmarks

de forma similar al anteriormente descrito, y heredan de la clase *QDialog*.

4.4. DockWidget.py

Este modulo maneja todas las funciones típicas de windows (arrastrar, copiar, pegar...) que tienen que ver con los widgets de la interfaz gráfica. Algunas de ellas son: marcar listas, mover vídeos, copiarlos, pegarlos, seleccionar una lista, arrastrar...

4.5. EncodeWindow.py

Este módulo gestiona todo lo relacionado con la ventana de "Export to video" que tiene la función de codificar los vídeos y cambiarles el formato de los mismos si se desea (pero nunca modificando el vídeo original). El método solo tiene una clase de la que hablaremos a continuación.

```
def startCommand(self):

    if self.extension == 'exe':
        self.makensis = True
    else:
        self.makensis = False
```

```
#Generate x264 command
selected = self.profileList.currentRow()
parameters = self.settings[selected]['parameters']
passes = self.settings[selected]['passes']

self.output.clear()
self.output.setOverwriteMode(True)


self.outputfile = self.fname.text()
basename, extension = os.path.splitext(str(self.outputfile))

if not extension:
    self.outputfile = self.outputfile + '.' + self.extension


if passes == 1:

    if(self.makensis):
        output = self.tempDir + "\\x264_temp.mp4"
    else:
        output = '"%s"' % self.path + self.outputfile

    command = 'x264 ' + parameters + ' --output ' +  output
    + ' ' + self.avsFile
    #print command
    self.encoding.start(command)
    self.startButton.setEnabled(False)
elif passes == 2:
    command = 'x264 --pass 1  --stats "' + self.tempDir +
    '\.stats" ' + parameters + ' --output NUL '  + self.avsFile
    self.analysis.start(command)
    self.startButton.setEnabled(False)
else:
    print "Unknown amount of passes"
    return
```

Método invocado al pulsar el botón "Encode"

Este método se encarga de hacer llamadas a **x264**, que es un programa libre que codifica los vídeos. En este caso, coge la información de la ventana "Expor to video" para generara el vídeo en el formato y con las características que se indiquen.

EncodeWindow

Esta es la única que se usa para manejar la ventana de "encode". Su método principal *__init__*, se encarga de crear la interfaz gráfica y hereda de la clase *QDialog* como muchos otros que se utilizan. Primero se inicializa con *QDialog.__init__(self, parent)* y se crean todas las variables necesarias para invocar un objeto de esta clase. A continuación empieza a introducir todos los widgets del panel que se muestran en la ventana.

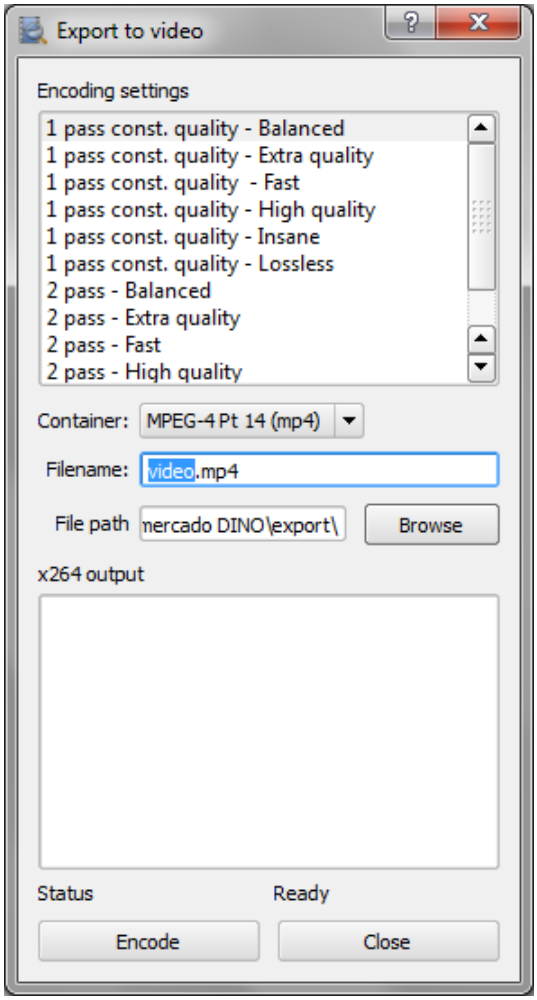


Figura 4.2: Ventana EncodeWindow

4.6. FilterWindow.py

Este módulo se encarga de manejar todo lo que esta relacionado con los filtros. Desde las ventanas que se encargan de gestionarlo hasta la aplicación y la modificación de los valores de cada filtro del programa. El módulo está compuesto por varias clases que veremos a continuación.

FilterWindow

Es una clase que hereda de *QDialog* y se encarga de crear en Qt la ventana "Add filter" en su método *__init__*. Los únicos botones definidos en esta clase son añadir ("add", añade un nuevo filtro), editar ("edit", edita un filtro previamente creado), borrar ("delete", borra un filtro) y cerrar ("close", cierra la ventana de filtros).

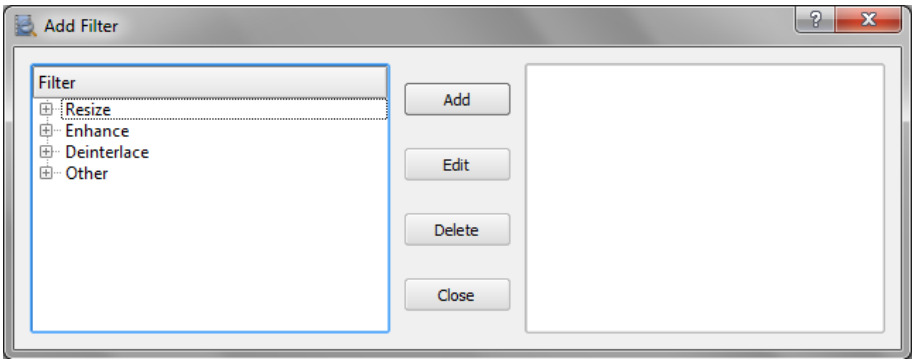


Figura 4.3: Ventana para añadir filtros

Ademas se crea un árbol para organizar todos los filtros que se pueden utilizar en el programa, cuyas ramas principales son "Resize", "Enhance", "Deinterlace" y "others" (se utiliza la función *QTreeWidgetItem* para crear cada una de estas ramas).

```
for filter in self.filters:
# La variable self.filters contiene todos los filtros diponibles.

            if filter.group == 'Resize':
                item = QTreeWidgetItem(resize)
                item.setText(0,filter.name)
            elif filter.group == 'Enhance':
                item = QTreeWidgetItem(enhance)
                item.setText(0,filter.name)
            elif filter.group == 'Deinterlace':
                item = QTreeWidgetItem(deinterlace)
                item.setText(0,filter.name)
```

```
            else:
                item = QTreeWidgetItem(other)
                item.setText(0,filter.name)
```

Función encargada de colocar los filtros en el árbol

Por otro lado, el método *EditFilter* es el que esta asociado al botón "edit" y permite modificar los parámetros del filtro seleccionado. A éste se asocian las ventanas "levelWindow", "resizeWindow" y "parameterWindow", que son lanzadas en función del filtro que utilicemos. En el fragmento de código que se muestra abajo se puede ver cómo se le pasan valores a la variable "pw" en función del tipo de filtro, y finalmente lo ejecuta (*value = pw.exec_()*).

```
def editFilter(self):

        row = self.listB.currentRow()
        if row > -1:
            filter = self.avi.filterChain[-1]


            if filter.name == 'Levels':
                pw = levelsWindow(self,self.avi,filter,row)
            elif filter.group == 'Resize':
                pw = resizeWindow(self,self.avi,filter,row)
            else:
                pw = parameterWindow(self,self.avi,filter,row)

            value = pw.exec_()
```

Método *EditFilter*

Lo mismo sucede con el método *addFilter* pero éste ademas añade cada filtro seleccionado en el recuadro que aparece a la derecha de la "Add Filter" window. Con la siguiente función añade el filtro a la lista de la derecha:

```
if value == 1:
     self.listB.addItem(filter.name)
     # Lista  de la derecha
```

Función *addItem* que añade filtros en el recuadro

Los métodos *clickedA* y *clickedB* se utilizan para poder señalar cada uno de los filtros seleccionados en los recuadros izquierdo (A) y derecho (B).

El método *removeFilter* se encarga de borrar los filtros de la ventana derecha (listB). Solo es posible borrarlos en el orden contrario al que han sido creados.

Por último el método *closeFilter* cierra la ventana de filtros.

ParameterWindow

Los objetos de esta clase, que hereda de *QDialog*, son lanzados cada vez que se presiona "add" en la ventana de "Add filters".

El método *__init__* lanza la ventana "parameterWindow" y con ella, los botones visibles de open, cancel, preview y update, y los botones ocultos de zoomIn, zoomOut y zoomReset. Y también el texto "modified" y la imagen del vídeo, que los mantiene ocultos.

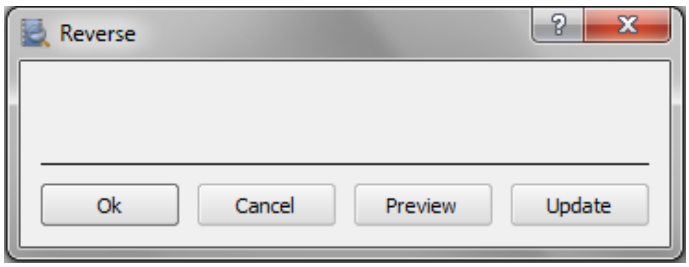


Figura 4.4: Ventana de ParameterWindow con botones ocultos

Una vez presionada la tecla "preview" se vuelven visibles los botones descritos arriba y también la pantalla con el vídeo. Todo aparece en la parte de arriba, y lo gestiona el método *view*.

El siguiente código hace que no aparezcan algunos de los botones ya comentados en el panel.

```
self.modified.hide()
        self.zoomInButton.hide()
        self.zoomOutButton.hide()
        self.zoomResetButton.hide()
        self.slider.hide()
        self.videoScroll.hide()
```

Declaración de botones ocultos

Cuando aparecen los nuevos parámetros, se cambia la tecla de "preview" por "hide" (asociada también al método *view*), que cuando es pulsada vuelven a ocultarse de nuevo.

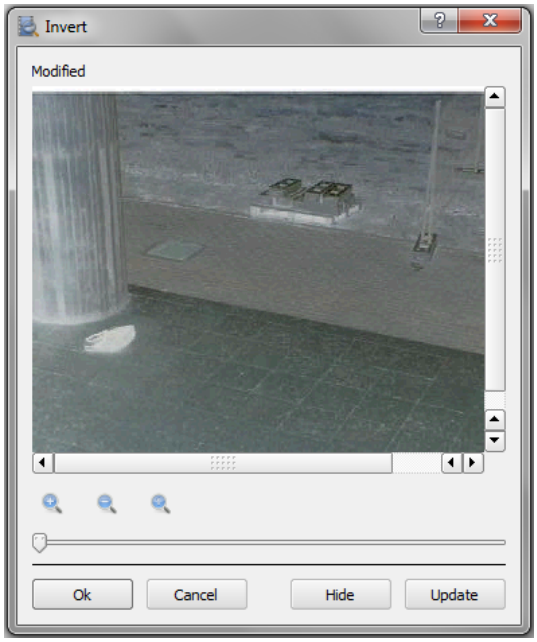


Figura 4.5: Ventana de ParameterWindow botones sin ocultar

El zoom es aplicado cuando se presiona cualquiera de sus teclas que se enlazan con cada uno de los métodos correspondientes (*zoomIn*, *zoomOut* y *zoomReset*).

El método *generateLayout* genera los parámetros para el manejo de los filtros aplicados y añade los widgets necesarios para cada filtro que lo requiere.

```
def generateLayout(self):

        self.paramObjects = []

        for parameter in self.filter.parameters:

            if parameter.type == 'float' and parameter.visible == True:
                slider = floatSlider(parameter)
            elif parameter.type == 'int' and parameter.visible == True:
                slider = intSlider(parameter)
            else:
                continue

            self.paramLayout.addWidget(slider)
            self.paramObjects.append(slider)
```

Método *generateLayout* para generar la barras

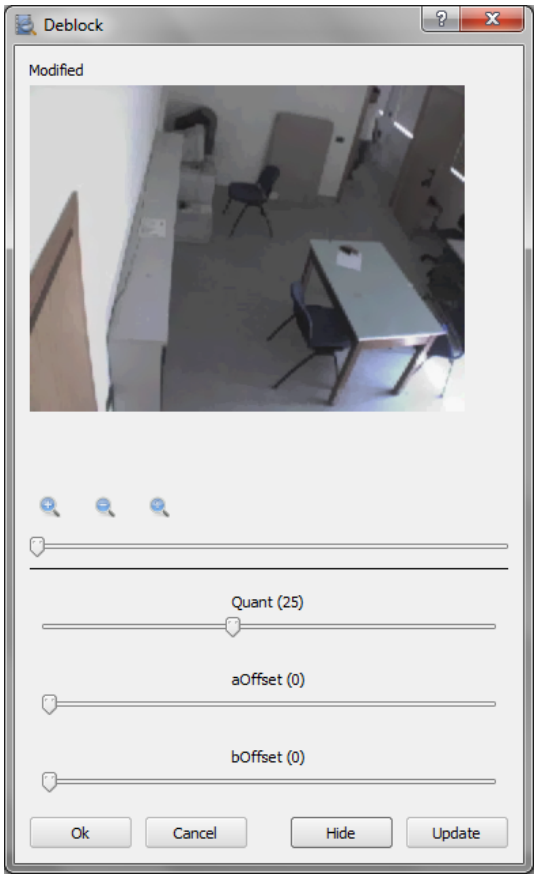


Figura 4.6: Ventana de ParameterWindow con tres widgets añadidos de tipo int

El método *updateFrames* se encarga de actualizar los fotogramas de cada imagen del vídeo que se selecciona, aplicándo el determinado "resize" y "zoom" que se le indica.

```
def updateFrames(self,index = 0):

    frame = self.slider.value()
    img = self.avi.returnImage(frame)
    self.image = img

    img = self.aviPreview.returnImage(frame)
    img = img.scaled(self.zoom*img.size())

    self.video.resize(img.size())
    self.video.setPixmap(QPixmap.fromImage(img))

    self.update()
```

El método *updateValues*, actualiza los valores de la ventana.

LevelsWindow

La clase *levelsWindow* hereda de la clase anterior (*parameterWindow*), es decir, que posee las mismas variables y métodos que *parameterWindow*.

Su método de iniciación es el *__init__*, como en todas las clases mencionadas. Este método crea dos nuevas variables que son "histLow" ("low" en el panel "Levels") e "histHigh" ("high" en el panel "Levels") y las inicializa a "0" y "255" respectivamente. A continuación llama a la función *__init__* creada en la clase *parameterWindow* y crea los mismos parámetros de la clase ya descrita, por lo que se genera una ventana exactamente igual a la anterior.

```
class levelsWindow(parameterWindow):

    def __init__(self, parent, avi, filter, editing=None):

        self.histLow = 0
        self.histHigh = 255


        parameterWindow.__init__(self, parent, avi, filter, editing)
```

Método *__init__* de *LevelsWindow*

El método *GenerateLayout*, se encarga de completar la ventana de parámetros que se genera en la función *__init__*. Coloca las barras de regulación de los nuevos parámetros ("Low" y "High") además de la de los parámetros adicionales que requiere el filtro en concreto, y genera un histograma con la función *VL1.addWidget(self.histogram)*. También se genera un widget con la función *QComboBox()* para elegir los colores del histograma por medio de un menú desplegable. A este menú se le añaden tres colores para elegir que son el rojo, el azul y el verde (con la función *combo.addItem()*).

```
        self.histogram = histogram.histogram(self.image,self)
        self.combo = QComboBox()
        self.combo.addItem(self.tr('Red histogram'))
        self.combo.addItem(self.tr('Green histogram'))
        self.combo.addItem(self.tr('Blue histogram'))

        VL1.addWidget(self.combo)
        VL1.addWidget(self.histogram)

        HL.addLayout(VL0)
        HL.addLayout(VL1)
```

```
        self.paramLayout.addLayout(HL)

        self.connect(self.slider, SIGNAL('sliderReleased()'),
        self.newHist)
        self.connect(self.combo, SIGNAL('activated(int)'),
        self.selectHist)
```

Código para generar los histogramas

Los demás métodos *updateLow*, *updateHigh*, *newHist*, *updateHist* y *selectHist*, se utilizan para actualizar todo lo que tiene que ver con el histograma, como pueden ser los limites, los dibujos, etc.

```
    def updateLow(self,index):
        self.histLow = index
        self.updateHist()

    def updateHigh(self,index):
        self.histHigh = index
        self.updateHist()

    def newHist(self):
        self.histogram.updateHist(self.image)

    def updateHist(self):

        low = float(self.histLow)/255
        high = float(self.histHigh)/255
        self.histogram.updateLimits([low, high])

    def selectHist(self,index):
        self.histogram.select(index)
```

Código para generar los histogramas

ResizeWindow

Esta clase se utiliza para modificar los valores predeterminados de las dimensiones del vídeo. En su método de inicio, crea una nueva variable y ademas llama al método *__init__* de la clase *parameterWindow*, que es de la que hereda, para generar la ventana correspondiente.

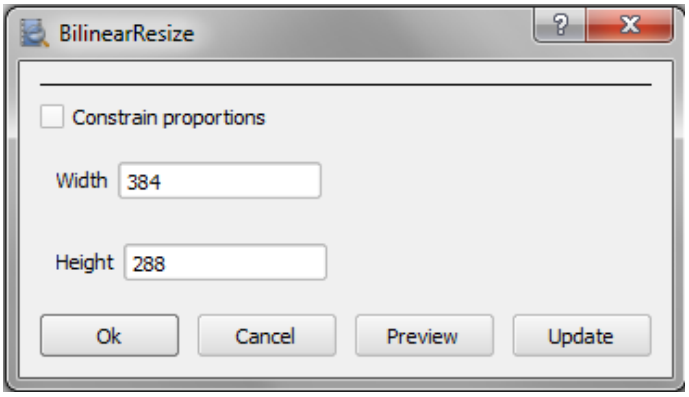


Figura 4.7: Ventana generada por la clase *resizeWindow*

El método *generateLayout* genera los widget que son necesarios en la ventana para llevar a cabo el control de los parámetros que se van a modificar. Estos son "width" (para la anchura), "height" (para la altura) y "constrain proportions" (para conservar la relación de aspectos).

Para finalizar la descripción de esta clase, tenemos los métodos *changeWidth* y *changeHeigth* que se encargan simplemente de que se actualicen los valores restablecidos, de la anchura y la altura de nuestro vídeo.

```
def changeWidth(self, value=0):

    if self.aspectCheck.checkState() == Qt.Checked:
        width = float(self.height.text.text())
        width = int(width*self.ar)
        self.width.text.setText(str(width))

def changeHeight(self, value=0):

    if self.aspectCheck.checkState() == Qt.Checked:
        height = float(self.width.text.text())
        height = int(height/self.ar)
        self.height.text.setText(str(height))
```

Funciones para cambiar el ancho y el alto

IntSlider y FloatSlider

Estas dos clases heredan a su vez de la clase *QWidget*, que es otra clase de Qt, y se utilizan para generar los widgets de las barras horizontales que manejan los parámetros de los filtros. Utilizan funciones como *QSlider()* para generar la barra, *SetOrienta-*

tion(Qt.Horizontal) para colocarla, etc. Las funciones (o métodos) de layout la llaman cada vez que se invoca un filtro que requiera la barra de calibración, como por ejemplo "TemporalSoften", "Levels" o "Blur".

```
class intSlider(QWidget):

    def __init__(self,parameter,parent = None):

        QWidget.__init__(self, parent)
        layout = QVBoxLayout()

        self.parameter = parameter

        self.label = QLabel(self.parameter.name)
        self.label.setAlignment(Qt.AlignHCenter)
        layout.addWidget(self.label)

        self.slider = QSlider()
        self.slider.setOrientation(Qt.Horizontal)

        self.slider.setRange(self.parameter.minimum,
        self.parameter.maximum)
        self.slider.setValue(self.parameter.value)

        layout.addWidget(self.slider)
        self.setLayout(layout)
        self.update()
        self.connect(self.slider, SIGNAL('valueChanged(int)'),
        self.update)
```

Código fuente de la clase *intSlider*

El método *update* se utiliza para actualizar el texto que aparece en el widget, y el método *getValue* para actualizar el valor que aparece en la barra.

IntBox

La clase *IntBox* hereda también de la clase *QWidget* y es llamada como la anterior por los métodos layout de otras clases. Su función *__init__* se encarga de crear las pequeñas cajas donde se escriben los parámetros que piden los filtros, como por ejemplo las cajas de "Width" y "Heigth" donde se modifican los parámetros en el filtro "BilinearResize".

```
def __init__(self,parameter,parent = None):
            QWidget.__init__(self, parent)
            H = QHBoxLayout()
            name = QLabel(parameter.name)
            self.text = QLineEdit(str(parameter.value))
            H.addWidget(name)
            H.addWidget(self.text)
            H.addStretch(1)

            self.setLayout(H)
```

Método *__init__* de la clase *IntBox*

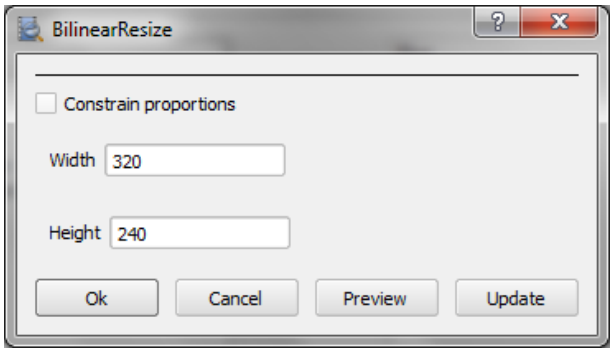


Figura 4.8: Cajas "Width" y "Heigth" generadas por la función *__init__* de la clase *IntBox*

4.7. Forevid.py

En este módulo se define la clase *forevid* que es la clase principal del programa. Es una clase de PyQt4 que hereda de la clase *QMainWindow*, y su papel es encargarse de mostrar la interfaz gráfica del programa y enlazarla con todas las funcionalidades.

A continuación se van a explicar varios de los métodos mas destacados que se utilizan en la clase, para poder entenderla:

init

Cuando el programa principal se ejecuta se comienza creando un objeto de la clase *forevid* y define todas sus variables. Al final del método *__init__* se lanza la ventana de selección de proyectos (ventana creada llamando a la clase *projectWindow* a través del método *getProject*), y una vez se elige uno, se termina de crear el objeto de la clase *forevid* con todos los atributos.

```
class forevid(QMainWindow):
    def __init__(self, parent=None):
            .
            .
            .
        value = self.getProject()
        if value == 0:
            sys.exit()
    def getProject(self):

        #Ventana Launch project
        proj = projectWindow(self.root)
        value = proj.exec_()

        if value == True:

            self.clearProject()

            self.root = proj.root
            self.settings.setValue('ProjectPath', self.root);
            self.project = proj.project
            self.description = proj.description
            self.author  = proj.author

            title = '%s - Forevid' % self.project
            self.setWindowTitle(title)

            self.loadLast()

            self.timer.start(self.backup,self)

        return value
```

Código *__init__* que llama a *getProject* y método *getProject* que llama a *getWindow*

Toda clase tiene su método *__init__* el cual se ejecuta siempre que se crea un objeto de la misma. En el caso de la clase *forevid* lo primero que hace es mostrar una imagen con el nombre del programa durante tres segundos, la cierra, muestra el diálogo de selección, permite seleccionar el proyecto, y cuando se elige uno, carga todas las configuraciones guardadas anteriormente dándole el valor por defecto a los atributos que no tienen un valor predefinido (atributos como los plugins usados en el vídeo, el idioma, las notas tomadas, etc.). Una vez cargados todos los atributos de nuestro espacio de trabajo, con la función *QMainWindow.__init__* se genera la ventana principal del programa, con todas las barras de herramientas, los menús y los botones que la componen (llamando a métodos como *generateToolbar*, *generateMenus*, etc) que pertenecen a la misma clase.

GenerateMenu

Este método se encarga de generar la barra de herramientas principal, que es la típica barra que tiene cualquier programa estándar situada en la parte superior. La función que se encarga de crear cada pestaña de la barra es *self.menuBar().addMenu()*. Dentro de cada una de ellas se añaden nuevos botones con la función *self.fileMenu.addAction()* y se crean separadores usando *self.fileMenu.addSeparator()*

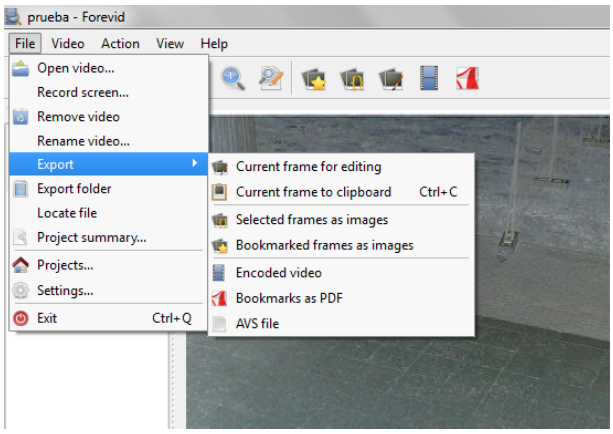


Figura 4.9: Ventana principal del programa

```
def generateMenus(self):
    self.fileMenu = self.menuBar().addMenu(self.tr("&File"))


    self.fileMenu.addAction(self.openVideoAct)
    self.fileMenu.addAction(self.recordScreenAct);
    self.fileMenu.addAction(self.removeVideoAct)
```

```
self.fileMenu.addAction(self.renameVideoAct)
self.exportMenu = self.fileMenu.addMenu(self.tr("&Export"))
self.fileMenu.addAction(self.browseAct)
self.fileMenu.addAction(self.locateFileAct)
self.fileMenu.addAction(self.projectSummaryAct)

self.fileMenu.addSeparator()
self.fileMenu.addAction(self.changeProjectAct)
self.fileMenu.addAction(self.settingsModifyAct)

self.fileMenu.addSeparator()
self.fileMenu.addAction(self.exitAct)

self.exportMenu.addAction(self.exportEditAct)
self.exportMenu.addAction(self.clipboardAct)
self.exportMenu.addSeparator()
self.exportMenu.addAction(self.exportSelectedAct)
self.exportMenu.addAction(self.exportMarkedAct)
self.exportMenu.addSeparator()
self.exportMenu.addAction(self.exportVideoAct)
self.exportMenu.addAction(self.exportPdfAct)
self.exportMenu.addAction(self.exportAvsAct)
```

Método *generateMenus*

En la figura de arriba se muestra un ejemplo de como van generando el menú poco a poco con funciones tales como *addAction* y *addSeparator*. Y en la figura de "Ventana principal del programa" se ve su representación gráfica.

GenerateToolBar

GenerateToolBar es un método llamado por la clase *forevid* y se encarga de mostrar la interfaz gráfica de la barra de tareas secundaria de nuestra ventana principal mediante la llamada de la función *self.addToolBar()*. Con la función *self.toolBar.addAction()* se van añandiendo los iconos de la toolbar de la ventana principal (como open vídeo, zoom in...) y con la función *self.toolBar.addSeparator()* se generan las separaciones que se ven a continuación (como el que se ve entre export folder y zoom in).

Figura 4.10: Barra de herramientas de la ventana principal

```
def generateToolBar(self):

    self.toolBar = self.addToolBar(self.tr("Toolbar"))

    self.toolBar.addAction(self.openVideoAct)
    self.toolBar.addAction(self.changeProjectAct)
    self.toolBar.addAction(self.browseAct)
    self.toolBar.addSeparator()
    self.toolBar.addAction(self.zoomInAct)
    self.toolBar.addAction(self.zoomOutAct)
    self.toolBar.addAction(self.zoomResetAct)
    self.toolBar.addAction(self.addFilterAct)
    self.toolBar.addSeparator()
    self.toolBar.addAction(self.exportMarkedAct)
    self.toolBar.addAction(self.exportSelectedAct)
    self.toolBar.addAction(self.exportEditAct)
    self.toolBar.addAction(self.exportVideoAct)
    self.toolBar.addAction(self.exportPdfAct)
```

Método *generateToolBar*

GenerateStatusBar

Con el método *GenerateStatusBar* se crean los widgets permanentes que muestran el número de fotograma del vídeo y el tiempo en horas, minutos y segundos que lleva recorrido el mismo. La función que se encarga de realizar esa tarea es *self.statusBar(). addPermanentWidget()* y es llamada dos veces, una para la pantalla de frames y segundos y la otra para la pantalla de información general.

```
def generateStatusBar(self):
    self.statusBar().showMessage(self.tr("Ready"))
    self.statusBar().addPermanentWidget(self.player.statusDisplay)
    self.statusBar().addPermanentWidget(self.player.statusDisplay2)
```

Método *generateStatusBar*

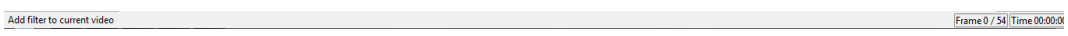


Figura 4.11: Barra de estado: a la derecha el tiempo y el fotograma y a la izquierda la pantalla de información general.

TimeEvent

Este método es bastante interesante, ya que se encarga de hacer copias de seguridad periódicas del estado del proyecto actual mediante la llamada al método *saveCurrent*, es decir, que cada cierto periodo de tiempo se auto guarda en el disco el estado actual del proyecto, y además, se deja un mensaje en la barra de estado ("Current status saved") en el momento en el que se realiza la copia.

Otros métodos interesantes

- *CleanProject*- Limpia todas las variables del proyecto actual.
- *Browser*- Abre la carpeta de exportación de proyectos
- *LocateFile*- Abre la carpeta que contiene el archivo de vídeo.
- *OpenVideo*- Crea un objeto de la clase *QFileDialog()* que se encarga de generar la interfaz gráfica de Qt que se utiliza para abrir un nuevo vídeo.
- *Zoom*- Los métodos *zoom in*, *zoom out* and *zoom reset*, son los que controlan las funciones de *zoom* para, aumentar, disminuir y restablecer el original.
- *VideoActivated*- Activa el vídeo seleccionado para poder manipularlo utilizando la función *self.videoList.list.currentItem()*.
- *VideoPopup*- Es el menu que Qt genera cuando se selecciona con el boton derecho del ratón sobre un vídeo.
- *CutVideo*- Se encarga de recortar y borrar fotogramas de partes del vídeo que se tienen en ese preciso momento. Primero se selecciona una franja de vídeo y luego se borran los fotogramas uno por uno utilizando un bucle.
- *VideoHash*- Abre la ventana de hash de Qt (hashWindow).
- *AddFilter*- Abre la ventana de filtros de Qt (*filterWindow*).
- *ExportPdf*- Exporta todas las "bookmars" creadas a un documento PDF para generar un informe.
- *ExportAvs* - Exporta todo el procesado que se a realizado en el vídeo a un script de AviSynth.
- *ExportVideo*- Codifica el vídeo modificado en un nuevo archivo de vídeo con el nombre que se le indique.
- *Clipboard*- Abre la ventana de "clip board" con el fotograma actual para poder manipularlo.

4.8. Hash.py

Este módulo se encarga simplemente de gestionar todo lo relacionado con el hash del vídeo. Para ello se importa en la librería que se utiliza para calcular el hash en Python, "import hashlib". El hash es un código que se utiliza para saber si el vídeo ha sido modificado o no, en caso de que alguien lo haya manipulado intencionadamente o por error. Este módulo posee solo dos clases:

ForevidHash

Esta clase tiene tres métodos: *__init__*, *getHash* y *compareHash*. El método de iniciación simplemente recoge el nombre del archivo de vídeo para identificarlo.

El método *getHash* se encarga de aplicar y obtener el valor de los distintos algoritmos para calcular el hash que proporciona el programa, que son: "md5", "sha1", "sha224", "sha256", "sha384" y "sha256". Lo hace llamando a funciones de la librería de hash de python (*hashlib.md5(fid.read()).hexdigest()*).

```
def getHash(self,method):

    try:
        fid = open(self.filename, 'rb')
    except:
        hexvalue = self.tr('Unable to open %1').arg(self.filename)
        return hexvalue

    if method == 'md5':
        hexvalue = hashlib.md5(fid.read()).hexdigest()
    elif method == 'sha1':
        hexvalue = hashlib.sha1(fid.read()).hexdigest()
    elif method == 'sha224':
        hexvalue = hashlib.sha224(fid.read()).hexdigest()
    elif method == 'sha256':
        hexvalue = hashlib.sha256(fid.read()).hexdigest()
    elif method == 'sha384':
        hexvalue = hashlib.sha384(fid.read()).hexdigest()
    elif method == 'sha512':
        hexvalue = hashlib.sha512(fid.read()).hexdigest()
    else:
        hexvalue = 'Unknown hash algorithm %s' % method
```

```
        #Cierra el archivo
        fid.close()

        return hexvalue
```

El método *compareHash* simplemente se encarga de comparar el nuevo hash con el anterior, y almacena la respuesta en la variable "value".

```
    def compareHash(self,method, target):

        newHash = self.getHash(method)

        #Asocia los hashes
        value = newHash == target

        return value
```

Método *compareHash*

HashWindow

Esta clase hereda de la clase *QDialog* y genera la ventana de cálculo del hash en el programa Forevid.

En su método *__init__* genera todos los widgets que corresponden a la ventana de "Compare hash values". Primero crea una caja para escribir (*QLineEdit*), debajo una fila de texto (*QLabel*), debajo una pestaña desplegable (*QComboBox*) y debajo una fila con texto (*QLabel*) que indicará si los hash comparados coinciden o no (es decir, si se trata del mismo vídeo o no).

```
class hashWindow(QDialog):
    def __init__(self, parent=None,fileName = 'suzie.avi'):
        QDialog.__init__(self, parent)
        self.fileName = fileName

        self.setWindowTitle(self.tr('Compare hash values'))

        form = QFormLayout()

        self.videohash = forevidHash(self.fileName)
        self.hash = self.videohash.getHash('md5')
```

```
self.given = QLineEdit(self)
self.current = QLabel(self.hash)
self.current.setTextInteractionFlags(
Qt.TextSelectableByKeyboard|Qt.TextSelectableByMouse)
self.combo = QComboBox(self)
self.result = QLabel('-')
  .
  .
  .
```

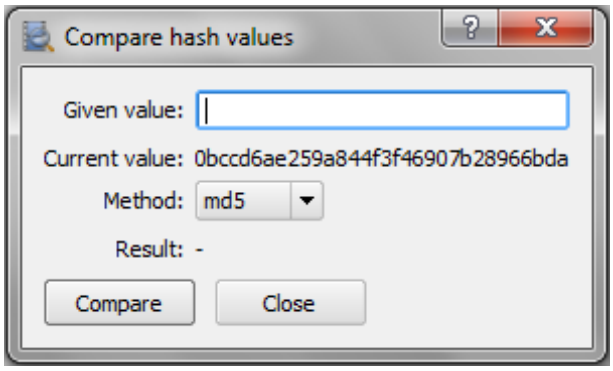


Figura 4.12: Objeto QDialog con los widgets para calcular el hash

Los otros tres métodos son *updateHash*, *calculateHash* y *exit*.

El método *updateHash* actualizan el valor del hash en función del método de cálculo seleccionado. El método *calculateHash* compara el hash del vídeo actual con el que se le introduce en el recuadro "Given value", y genera el mensaje "Hash is valid" o "Hash is invalid" si la comparación coincide o no coincide respectivamente. Y el método *exit* simplemente cierra la ventana "Compare hash values".

4.9. Histogram.py

En lo que se refiere a este módulo, el único papel que toma, es dibujar y actualizar cada histograma que utiliza el programa. Su única clase, *histogram*, se encarga de crear los objetos de este tipo. La clase hereda de *QWidget*, que como ya sabemos, es una clase de Qt que se utiliza en la creación de widgets. Este módulo se encarga de actualizar, dibujar, modificar, colorear, etc. La función *getHistogram* es la encargada de obtener los histogramas de la imagen.

4.10. Icons.py

El módulo "Icons.py" guarda la ruta de los iconos que tiene el programa y los almacena en constantes para hacerlo mas sencillo al programar.

4.11. Project.py

Es el modulo que crea y maneja la ventana que se ocupa de abrir los proyectos, siendo la misma que aparece también cuando ejecuta el programa. Lo componen, la clase principal *projectWindow* y las clases *projectItem* y *newProject*.

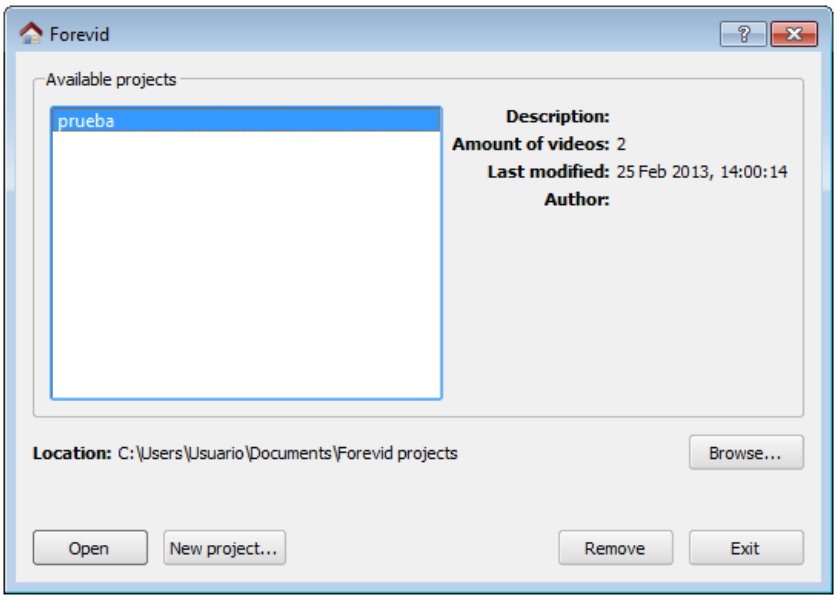


Figura 4.13: Venta de proyectos de Forevid

4.12. Pyavs.py

El módulo se en carga de manipular parte del manejo del vídeo. Esta compuesta por cuatro clases, *RECT*, *BITMAPINFOHEADER*, *BITMAPFILTERHEADER* y *AvsClip*.

RECT

Es una clase que se encarga de decir al programa donde se dibuja la pantalla del vídeo que se genera:

```
class RECT(ctypes.Structure):
    _fields_ = [("left", LONG),
                ("top", LONG),
                ("right", LONG),
                ("bottom", LONG)]
```

BITMAPINFOHEADER y BITMAPFILEHEADER

Guardan información sobre el mapa de bits como es el tipo de archivo, el tamaño, el espacio necesario a reservar en el buffer, etc.

```
class BITMAPINFOHEADER(ctypes.Structure):
    _fields_ = [("biSize",  DWORD),
                ("biWidth",   LONG),
                ("biHeight",   LONG),
                ("biPlanes",   WORD),
                ("biBitCount",   WORD),
                ("biCompression",  DWORD),
                ("biSizeImage",  DWORD),
                ("biXPelsPerMeter",   LONG),
                ("biYPelsPerMeter",   LONG),
                ("biClrUsed",  DWORD),
                ("biClrImportant",  DWORD)]

class BITMAPFILEHEADER(ctypes.Structure):
    _fields_ = [
        ("bfType",    WORD),
        ("bfSize",   DWORD),
        ("bfReserved1",    WORD),
        ("bfReserved2",    WORD),
        ("bfOffBits",   DWORD)]
```

AvsClip

En esta clase se guarda toda la información relativa a los vídeos que se editan, como por ejemplo, si se quiere añadir un filtro a un vídeo se hace a través de esta clase con el método *addFilter*. También los métodos *editFilter*, que permite que se modifique y reaplique, y *removeFilter*, que borra un filtro y actualiza los cambios.
"Refresh" se encarga de actualizar los cambios de los filtros para "editFilter" y "removeFilter".

```
def addFilter(self,filter):

    #Comprueba si se necesita cambiar el espacio de colores
    color =  filter.convertColor(self.Colorspace)

    print "Current colorspace is %s" % self.Colorspace

    if color:
        print "Color conversion to %s" % color
        self.convertColorspace(color)


    value = self.invokeCommand(filter)

    if value:
        #Añade un nuevo filtro a la cadena de procesado
        self.filterChain.append(filter)
        #self.displayVideo()
    else:
        print "Unable to add filter"
        return False


    self.vi=self.clip.GetVideoInfo()
    self.Framecount = self.vi.num_frames
    self.Width = self.vi.width
    self.Height = self.vi.height
```

Posee también un método que hace las conversiones de colores llamado *convertColorSpace*, es decir, que cambia de un espacio de colores a otro.

```
def convertColorspace(self,color):

    if color == 'RGB32':
        filter = avs_filter('RGB32', '', 'converttorgb32', [],
        visible=False)
    elif color == 'YUY2':
        filter = avs_filter('YUY2', '', 'converttoyuy2', [],
        visible=False)
    elif color == 'YV12':
        filter = avs_filter('YV12', '', 'converttoyv12', [],
        visible=False)
    else:
        print "Shouldn't be here?"

    self.invokeCommand(filter)
    self.filterChain.append(filter)
```

El método *InvokeCommand* invoca un comando de AviSynth, como puede ser cargar un vídeo a la clase, que cambie el espacio de colores (*convertColorSpace* llama a este método), aplicar filtros a los clips, etc. Cuando se reproduce un vídeo, esta clase indica el momento de la reproducción en el que se encuentra en segundos, con el método *GetTime*, o en el caso de que se desee recordar el fotograma se utiliza el método *GetFrame*.
ExportAvs: Devuelve un script de AviSynth con los vídeos que se han cargado y los filtros aplicados sobre ellos.
DisplayVideo: el vídeo que tenemos en la clase *avsClip* lo coloca en el buffer.

4.13. Summary.py

El módulo *summary* genera archivos de PDF con un resumen de filtros y modificaciones aplicado a todos los vídeos del proyecto. Los objetos de las clases de este módulo son llamados con la secuencia: *File* → *Project summary*.

4.14. VideoPlayer.py

El módulo se encarga de manejar el reproductor de vídeo de la interfaz de Qt, que es un widget de la clase.

La clase más importante es *videoPlayer*, que hereda de la clase *QWidget*. Se genera cada panel de reproducción de vídeo llamando a la función *QHBoxLayout()* y los vídeos son manejados por los botones que se crean en su método *generateButtons*.

```
class videoPlayer(QWidget):

    def __init__(self, videoOutput = 'Qt', parent=None):
        QWidget.__init__(self, parent)

        layout = QVBoxLayout()

        self.videoA = videoFrame(videoOutput)
        self.videoB = videoFrame(videoOutput)

        videos  = QHBoxLayout()
        videos.addWidget(self.videoA)
        videos.addWidget(self.videoB)
        self.videoB.hide()

        self.timeline = timeLine()

        self.generateButtons()
```

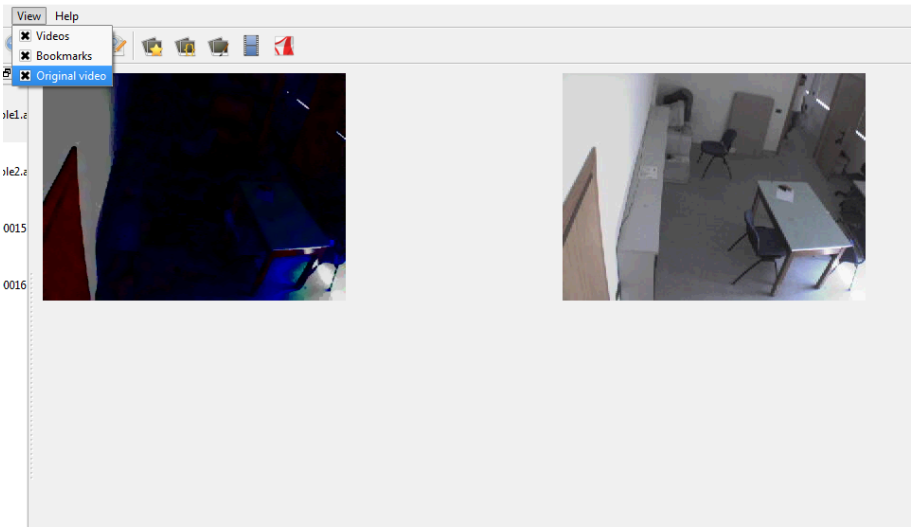


Figura 4.14: Ejemplo de como se muestra videoA y videoB

En el método *__init__*, el "videoA" (izquierda) muestra el vídeo sobre el que se aplican las modificaciones y los filtros, y el "videoB" (derecha) muestra el vídeo original, pero en un principio se muestra oculto. Para ver el vídeo original se selecciona view → Original video.

Las demás clases tales como, por ejemplo, *videoFrame*, *PMFrame*, *timeLine*, están asociadas a las acciones, el control de la linea de tiempo, los frames, etc.

4.15. Otros

Otros módulos aparte de los mencionado, se encargan de funciones como generar y organizar los archivos PDF, que es el caso de *Repor.py*, generar y cargar las rutas de todos los iconos de los widgets (*Icons.py*)...

Capítulo 5

Incorporar un algoritmo a Forevid

En este capítulo vamos a añadir un algoritmo a la funcionalidad de Forevid, barajando los distintos caminos que se han descubierto en la investigación.

Antes de nada, para que funcionen los algoritmos, es necesario incorporar los plugins de los mismos al programa. Por lo que se dedicará un apartado de este capítulo a ello.

La inclusión del algoritmo creado se podría hacer de dos formas distintas. La primera de ellas, y más sencilla, es creando un script en el que se hace la llamada a nuestro algoritmo. Y la solución más compleja seria incorporar una nueva función, por medio de un botón dentro del mismo programa de forevid. Para ver con más claridad de que se trata, en la explicación se utiliza el algoritmo que previamente ha sido creado en el capitulo 2.4.

Crear y utilizar una biblioteca de vínculos dinámicos (DLL) en C++

En el capítulo de 2.4, se explicó como se programa el código del algoritmo "invertir". En esta sección se muestra como incorporar el algoritmo al programa. Para eso lo primero que se debe hacer es generar el archivo ".dll" para que pueda ser añadido como un plugin a AviSynth. El archivo ".dll" ha sido creado con el program "Visual Studio 2010" de Microsoft. A continuación se procede a explicar lo que hay que hacer para crear el ".dll" de nuestro algoritmo.

Lo primero, es generar un nuevo proyecto de biblioteca de vínculos dinámicos (DLL). Los pasos a seguir, desde el "Visual Studio 2010", son los siguientes [3]:

- En el menú Archivo, se selecciona "Nuevo", y a continuación, "Proyecto".
- En el panel "Tipos de proyecto", en "Visual C++", se selecciona "Win32".
- En el panel "Plantillas", se escoge la opción "Aplicación de consola Win32".

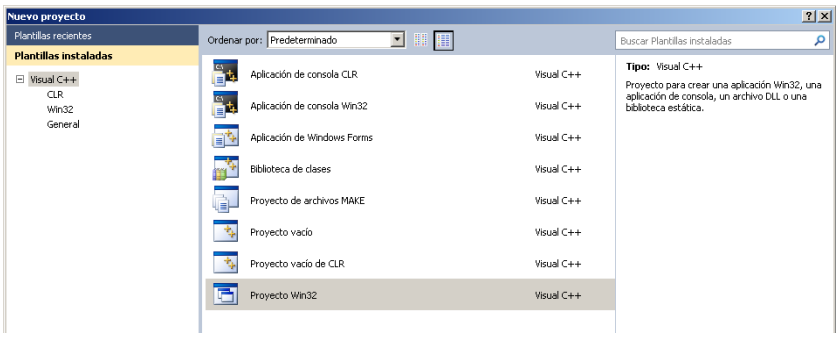


Figura 5.1: Ventana "Nuevo proyecto"

- Se selecciona un nombre para el proyecto, como "ProyectoDll", y se escribe en el campo "Nombre". Se elige un nombre para la solución, como "DynamicLibrary", y se escribe en el campo "Nombre de la solución".
- A continuación se pulsa "Aceptar" para iniciar el asistente para aplicaciones Win32. En la página "Información general" del cuadro de diálogo "Asistente para aplicaciones Win32", presionamos siguiente.
- En la página "Configuración de la aplicación" del "Asistente para aplicaciones Win32" en "Tipo de aplicación" se escoge, "DLL" si está disponible o "Aplicación de consola" si "DLL" no está disponible.

- En la página “Configuración de la aplicación” del “Asistente para aplicaciones Win32” en “Opciones adicionales”, se elige “Proyecto vacío”.

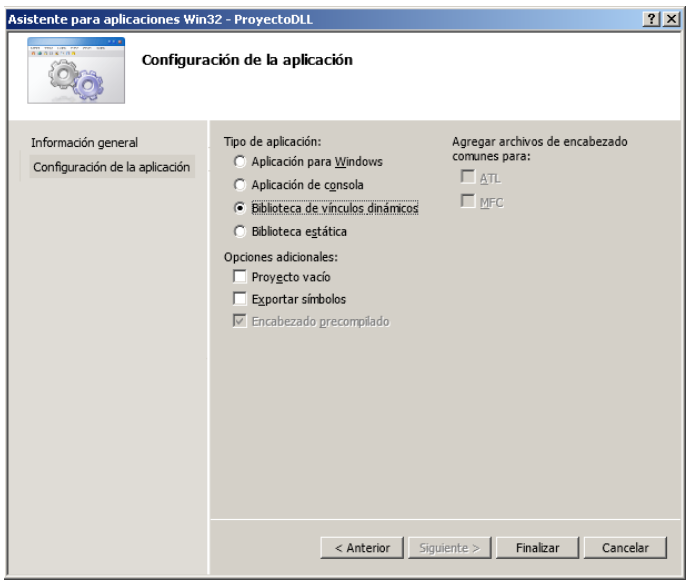


Figura 5.2: Configuración de la aplicación

- Por ultimo se selecciona “Finalizar” para generar el proyecto.

A continuación se agrega una clase a la biblioteca de vínculos dinámicos, que va a ser la clase de nuestro filtro “invertir”.

- Para crear un archivo de encabezado para una nueva clase, en el menú “Proyecto”, se selecciona “Agregar nuevo elemento” (aparece el cuadro de diálogo “Agregar nuevo elemento”). En el panel “Categorías”, bajo “Visual C++”, se selecciona “Código”. En el panel “Plantillas”, se escoge “Archivo de encabezado (.h)”. Se escoge un nombre para el archivo de encabezado, como “ProyectoDll.h”, y se agrega. Se mostrará un archivo vacío.

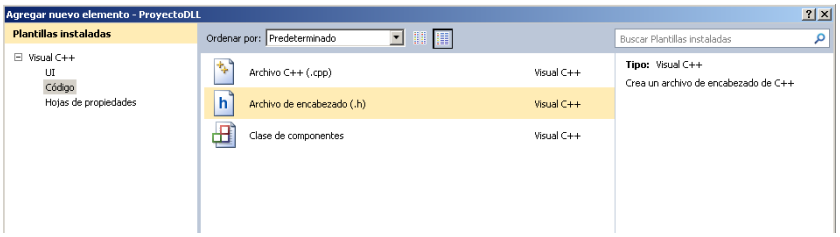


Figura 5.3: Ventana “Agregar nuevo elemento”

- Se agrega una clase simple a la que en este caso se ha denominado "ProyectoDLL.h" y se escribe el código descrito a continuación.

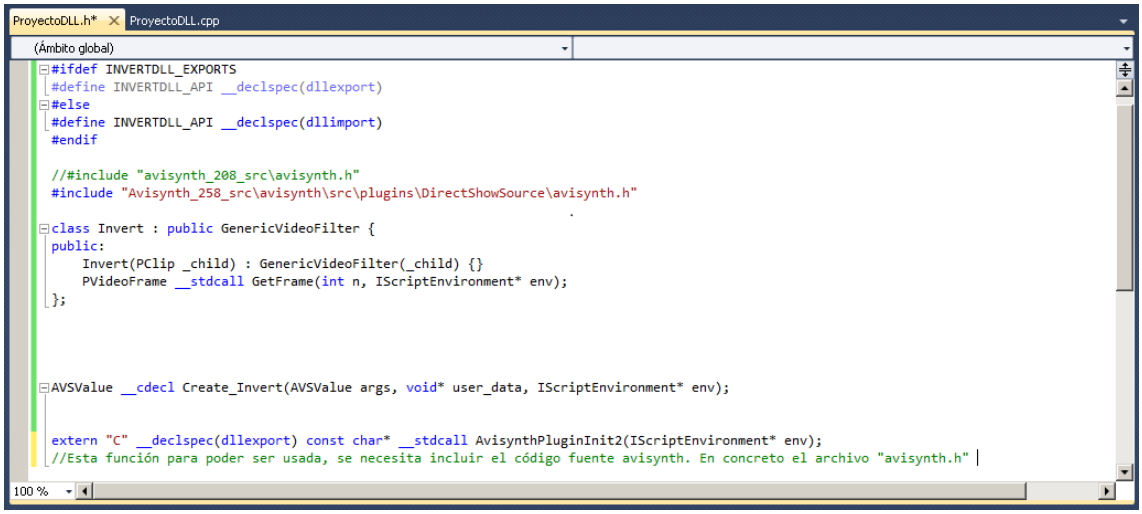


Figura 5.4: Código del archivo de cabecera

- Para crear un archivo de código fuente para una nueva clase, en el menú "Proyecto", se selecciona "Agregar nuevo elemento...". Aparece el cuadro de diálogo "Agregar nuevo elemento". En el panel "Categorías", bajo Visual C++, se selecciona "Código". En el panel "Plantillas", se selecciona "Archivo C++" (".cpp"). Elegimos un nombre para el archivo de origen, como "ProyectoDll.cpp", y se pulsa "Agregar". Se muestra un archivo vacío. A continuación se implementa la función de nuestro algoritmo.

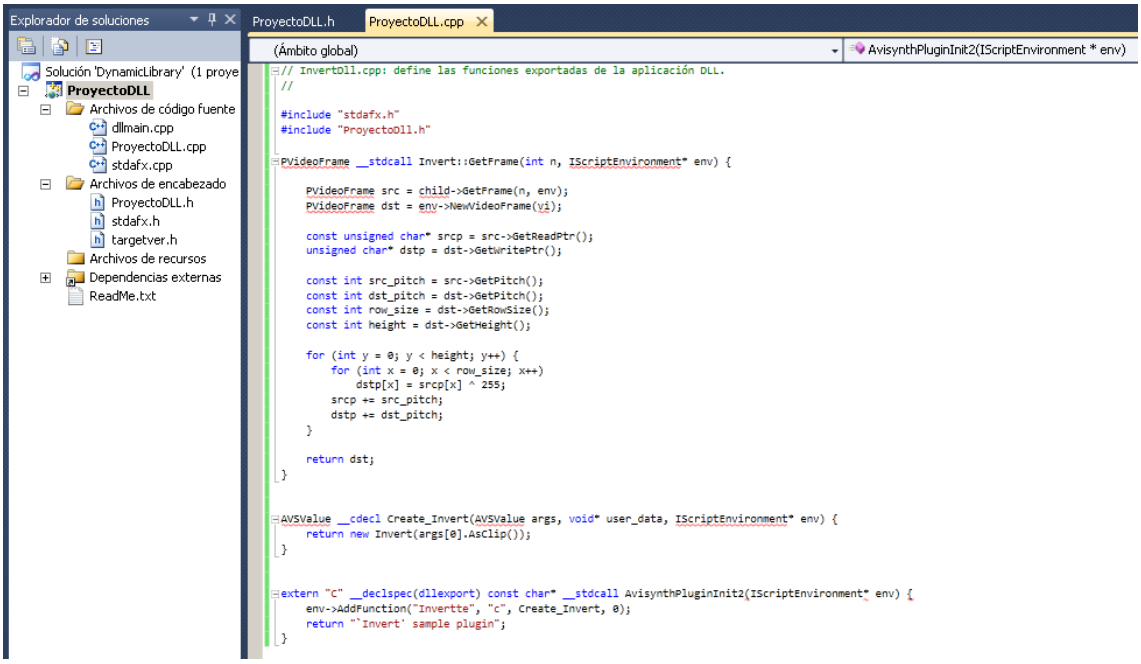


Figura 5.5: Código ".cpp"

- Para compilar el proyecto en un archivo DLL, en el menú "Proyecto", se selecciona "Propiedades... de ProyectoDll". En el panel izquierdo, en "Propiedades de configuración", se selecciona "General". En el panel derecho, se cambia el Tipo de configuración a "Biblioteca dinámica (.dll)". Finalmente se pulsa "Aceptar" para guardar los cambios.

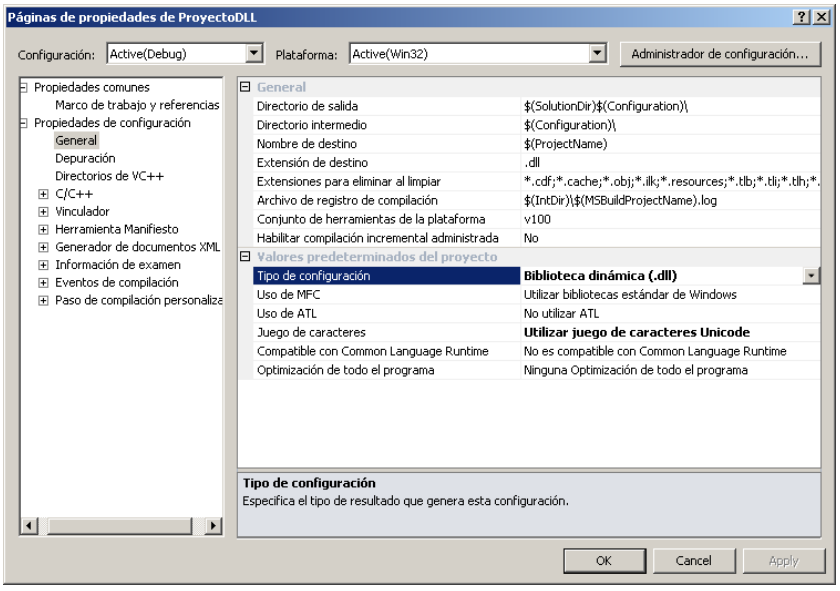


Figura 5.6: Ventana de propiedades del proyecto

- Descargamos la librería "Avisynth_258_src"[1], la descomprimimos, y la copiamos dentro de nuestro proyecto en la carpeta con el nombre de la clase creada (en este caso "ProyectoDLL").
- Compilamos la biblioteca de vínculos dinámicos seleccionando "Generar solución" en el menú "Generar". Esto crea un archivo DLL que puede ser utilizado por otros programas.

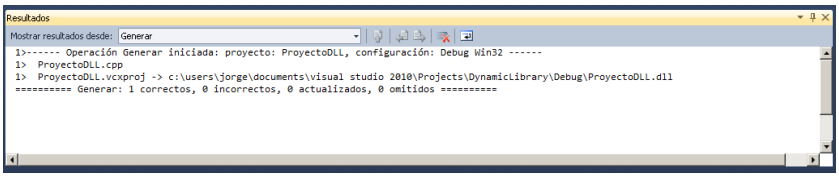


Figura 5.7: Ventana de resultados

Por último para que Forevid reconozca el algoritmo se incorpora nuestro archivo ".dll" a la carpeta "AVS_Plugins" de el espacio de trabajo que se este utilizando en nuestro proyecto de "eclipse".

5.1. Crear un nuevo script

Este método de incorporación de algoritmos, es el mas apropiado en caso de que no se desee modificar el código fuente del programa. Pero además, también sería apropiado si se pretende incorporar el algoritmo con la mayor brevedad posible, ya que solo se deben añadir unas cuantas lineas de código.

En este caso lo primero que se debe hacer, una vez se programa el nuevo algoritmo y se genera nuestro archivo ".dll", es abrir cualquier editor de texto (como por ejemplo el "Bloc de notas" de windows). Una vez hecho esto se escribe lo siguiente:

```
LoadPlugin("C:\Users\Usuario\workspaceTFC\TFCproject\AVS_Plugins
\InvertDll.dll")
```

Esta primera linea de código, se encarga de cargar el plugin ".dll", así AviSynth puede reconocer el nombre de la función creada para ejecutar nuestro algoritmo. De este modo cada vez que se llama a la función (que en este caso ha sido llamada "Invertte", para ser diferenciado de "Invert" que ya existe), AviSynth la busca dentro del plugin **invertDll.dll** cuya dirección ha de ser especificada como se ve en el ejemplo.

[1] Para descargar la librería mirar el capítulo A

```
clip = AVISource("C:\Users\Usuario\Desktop\example2.avi")
```

Este comando genera la variable "clip", y en ella guarda la ruta del vídeo que se manipula. Ahora cada vez que se desee modificar algo, simplemente habrá hacerlo sobre esta variable.

```
return clip.Invertte()
```

Y por último, se aplica la nueva función del algoritmo, sobre el vídeo al que apunta la variable "clip"[2]. En caso de tener que definir algún parámetro en nuestro algoritmo, se especificaría aquí.

Con esto el algoritmo esta casi listo para ser utilizado. Así se vería el código de AviSynth completo, desde el "Bloc de notas" de Windows.

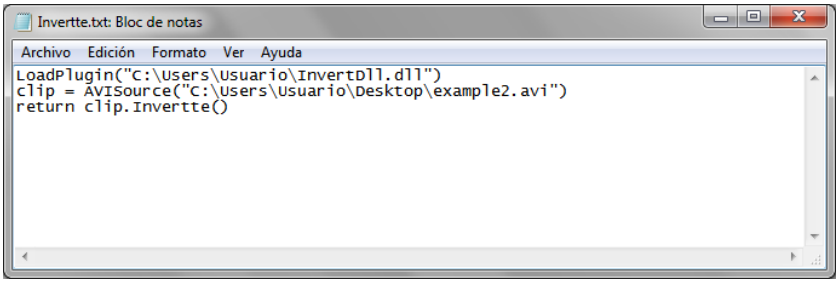


Figura 5.8: Código AviSynth en bloc de notas

Ahora se guarda con extensión ".avs". Por tanto el archivo podrá ser abierto con cualquier reproductor de vídeo que pueda reproducir en formato AVS.

Finalmente solo quedará abrir el programa Forevid, y en lugar de cargar el vídeo original, se carga el archivo creado con extensión ".avs". De esta forma se abre el vídeo en Forevid con el algoritmo ya incorporado, por lo que se visualizan los fotogramas con el algoritmo aplicado (modificados) en lugar de los originales del vídeo.

El único inconveniente que tiene incorporar algoritmos de este modo, es que quedan permanentemente añadidos al vídeo que manipulamos en Forevid. Para poder trabajar

[2]Recordar que la gran ventaja de AviSynth, es que todas la modificaciones que se hagan al vídeo, son aplicadas en el preciso momento de su reproducción, ya que en un *framserver* el vídeo original nunca es modificado.

en el vídeo sin utilizar el algoritmo solo se tiene que cargar el vídeo original o modificar el código de AviSynth con el que ha sido programado.

5.2. Incorporar un comando para nuestro algoritmo en Forevid

Esta solución sería la más larga y apropiada, si lo que queremos es que el algoritmo forme parte del programa Forevid permanentemente. Además, permite modificar, de manera interactiva, los parámetros del filtro, de forma que el usuario puede encontrar los valores óptimos para el vídeo actual de forma más cómoda.

En este caso se procederá a modificar el código fuente del programa para añadir un nuevo comando que aplique el algoritmo creado. De esta forma el algoritmo se verá como uno más de los que aparecen en el programa.

Para añadir el algoritmo se puede, o bien incluir en uno de los arboles ya creados en el programa (resize, enhance, deinterlae y other), o añadir un nuevo árbol con un nuevo grupo para añadir el algoritmo. Todo esto en función de que el algoritmo pertenezca a alguno de los grupos ya creados o no.

5.2.1. Inclusión del algoritmo

Para incluir un nuevo algoritmo a la lista de filtros de Forevid, bastará con incorporar en la clase "avs_filter" del módulo "avs_filter" el filtro creado. A continuación se muestra un ejemplo en el que se agrega el filtro del capítulo 2.4 en la página 28:

```
avs_filter('Invertir', 'Nuevo grupo', 'Invertte', [],['RGB32', 'YUY2',
'YV12'])
```

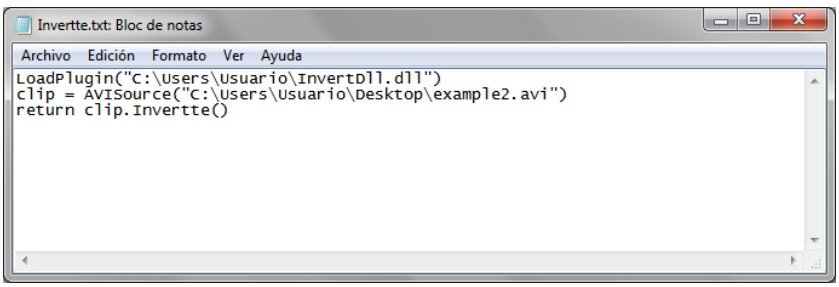


Figura 5.9: Clase avs_filter

- En el primer argumento de la función se escribe el nombre que se desea que aparezca visualizado en el programa Forevid. En este espacio se debe escribir el nombre por el que se vaya a reconocer el filtro o algoritmo, para que futuros usuarios lo reconozcan fácilmente.
- En el segundo argumento se escribe el grupo al que queremos que pertenezca el algoritmo. Es conveniente que el grupo indicado tenga relación con el algoritmo que queremos aplicar, para que el uso del programa sea lo más sencillo e intuitivo posible. Si el grupo escrito no pertenece a los grupos ya creados, debemos crear un nuevo grupo (más adelante se explica como añadir un nuevo grupo) o añadirlo al grupo "others".
- El tercer argumento se usa para indicar el nombre de la función de llamada en AviSynth para aplicarla, es decir, el nombre con el que hacemos la llamada al algoritmo en AviSynth.
- En el cuarto argumento iría la lista con los parámetros del filtro, que son objetos de la clase *avs_parameter*. En este caso el nuevo filtro no necesita parámetros, por ese motivo se deja sin rellenar. Mas adelante veremos como se incluyen los argumentos.
- Y el último objeto se refiere a los espacios de colores con los que trabaja la función, que en este caso son tres: "RGB32", "YUY2" y "YV12".

5.2.2. Ejemplo de un filtro con parámetros

Para incluir un filtro con parámetros se hace de la misma forma que en el ejemplo anterior. El único campo que cambia dentro del objeto *avs_filter* es el cuarto argumento.

```
avs_filter('Invertir', 'Nuevo grupo', 'Invertte',
[avs_parameter('Parameter 1','int',0,True,0,255)],['RGB32', 'YUY2',
'YV12'])
```

En este ejemplo, hemos utilizado el código del filtro "Invert" que se creó en el capítulo 2.4.1 de la página 35. El cuarto argumento se rellena con un objeto de la clase *avs_filter*. El primer argumento del objeto, *Parameter 1*, indica el nombre del parámetro, el segundo el tipo de dato, el tercero el valor por defecto (en este caso 0), el cuarto la visibilidad del parámetro ("True" visible y "False" invisible) y los dos últimos argumentos el valor mínimo y máximo respectivamente.

El nuevo parámetro incorporado se puede modificar mediante una barra "Slide". Cuando se añade el filtro al vídeo, se permite modificar el parámetro entre los valores enteros que se han definido (desde 0 hasta 255).

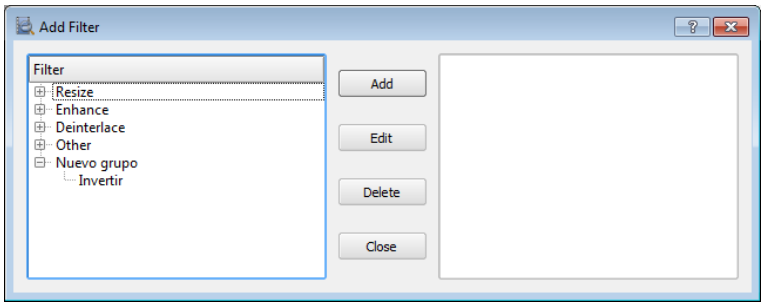


Figura 5.10: Seleción del nuevo filtro "Invertir"

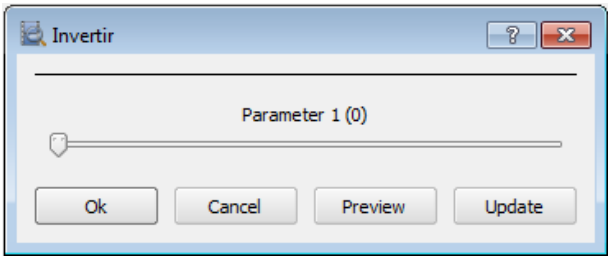


Figura 5.11: Barra "Slide" que modifica el nuevo parámetro

Añadir un nuevo grupo

Para añadir un nuevo grupo de algoritmos, se tiene que trabajar sobre el módulo *filterwindow*. En este módulo, para ser mas concretos en la función $\_\_int\_\_$, es donde se describen todos los grupos de filtros que aparecen en el programa, y por lo tanto, el lugar donde se ha de añadir el nuevo grupo para el algoritmo.

```
resize = QTreeWidgetItem(self.listA)
resize.setText(0,self.tr('Resize'))

enhance = QTreeWidgetItem(self.listA)
enhance.setText(0, self.tr('Enhance'))

deinterlace = QTreeWidgetItem(self.listA)
deinterlace.setText(0, self.tr('Deinterlace'))

other = QTreeWidgetItem(self.listA)
other.setText(0, self.tr('Other'))
```

Código donde se describen los grupos para organizar los filtros

Como se puede observar, para generar el nuevo grupo se tiene que crear una variable del tipo *QTreeWidgetItem*. Esta nueva variable se añade directamente a la lista A,

o “listA” como viene descrita en el código, lo que significa que se genera un árbol en el recuadro izquierdo de la ventana de “filterWindow”, por cada variable de este tipo creada. En este caso hay cuatro variables que son “resize”, “enhance”, “deinterlace” y “other”.

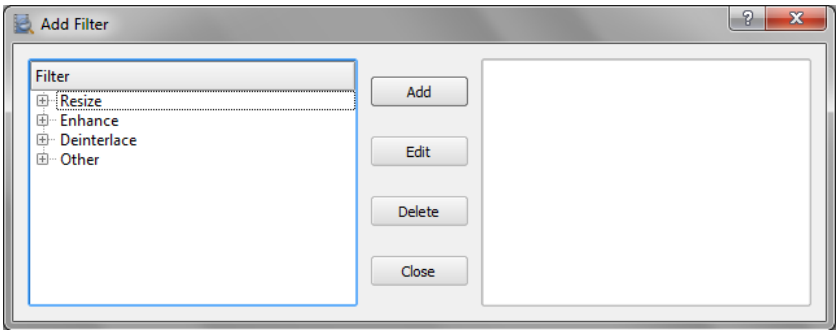


Figura 5.12: Arboles de filtros sobre el recuadro “listA”

Aquí se crea un nuevo grupo de filtros, añadiendo el nuevo fragmento de código debajo del último árbol creado (other). Este nuevo grupo se creó con el nombre “nuevo_grupo” para simplificar el ejemplo.

```
resize = QTreeWidgetItem(self.listA)
resize.setText(0,self.tr('Resize'))

enhance = QTreeWidgetItem(self.listA)
enhance.setText(0, self.tr('Enhance'))

deinterlace = QTreeWidgetItem(self.listA)
deinterlace.setText(0, self.tr('Deinterlace'))

other = QTreeWidgetItem(self.listA)
other.setText(0, self.tr('Other'))

nuevo_grupo = QTreeWidgetItem(self.listA)
nuevo_grupo.setText(0, self.tr('Nuevo grupo'))
```

Ejemplo de como añadir un nuevo grupo “nuevo_grupo”

Una vez añadido desde Eclipse, se salvan los cambios y se ejecuta el programa (para que se ejecute correctamente, ha de hacerse siempre desde el módulo Forevid.py). Ahora cuando se lanza la ventana de filtros aparecerá modificada, con el nuevo árbol incorporado en la “listA”.

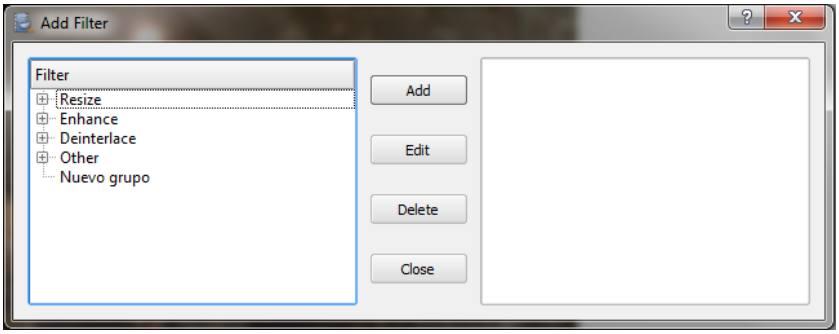


Figura 5.13: Arboles de filtros sobre el recuadro "listA" con el nuevo grupo añadido

Ahora solo queda modificar el programa, para que añada el nuevo algoritmo al nuevo grupo de filtros recién creado. Para ello se baja hasta el siguiente fragmento de código, en el que podemos observar, que mediante un bucle "for" se van añadiendo al programa Forevid todos los filtros y algoritmos a cada grupo que existente hasta el momento.

```
for filter in self.filters:

    if filter.group == 'Resize':
        item = QTreeWidgetItem(resize)
        item.setText(0,filter.name)
    elif filter.group == 'Enhance':
        item = QTreeWidgetItem(enhance)
        item.setText(0,filter.name)
    elif filter.group == 'Deinterlace':
        item = QTreeWidgetItem(deinterlace)
        item.setText(0,filter.name)
    else:
        item = QTreeWidgetItem(other)
        item.setText(0,filter.name)
```

Si no se añade a esta función nuestro nuevo grupo no sucederá lo esperado, aunque especifiquemos el grupo del algoritmo en la clase "avs_filter" del módulo "avs_filter.py". Lo que sucedería es que el algoritmo sería añadido al grupo "other". Por tanto se debe añadir la siguiente linea de código dentro de la función "for" para que sea reconocido.

```
    elif filter.group == 'Nuevo grupo':
        item = QTreeWidgetItem(nuevo_grupo)
        item.setText(0,filter.name)
```

Se debe añadir el fragmento como si se tratara de una condición "elif" más de la función "for" antes de la ultima cláusula "else" (la que se refiere al grupo "other").

Finalmente se puede incluir cualquier filtro o algoritmo al nuevo grupo sin ningún problema. En el ejemplo de mas abajo se ha añadido el algoritmo de inversión, con el que se ha trabajado hasta ahora, a nuestro nuevo grupo.

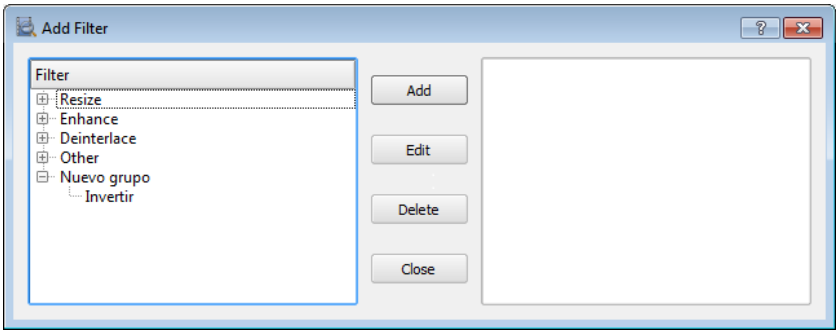


Figura 5.14: Ventana de filtros con el nuevo grupo y el filtro Invertir añadido

Los algoritmos son aplicados al vídeo una vez cerrada la ventana "Add Filter" (pulsando el botón "Close") siempre y cuando hayan sido previamente añadidos (pulsando el botón "Add") en el cuadro de la derecha.

Apéndice A

Instalación

En este anexo se incluye una breve y sencilla explicación de como se ha desarrollado el entorno gráfico utilizado para el estudio del trabajo final de carrera.

La herramienta principal que se utiliza para el estudio de Forevid es Eclipse. Eclipse es un entorno de desarrollo integrado de código abierto multiplataforma para desarrollar "aplicaciones de cliente enriquecido" (proporcionan una interfaz gráfica escrita con una sintaxis basada en XML, que proporciona funcionalidades como pueden ser, arrastrar y soltar, pestañas, ventanas múltiples, menús desplegables, etc). Esta plataforma, básicamente ha sido usada para entornos de desarrollo integrados (del inglés IDE), como el IDE de Java, llamado Java Development Toolkit (JDT), y el compilador (ECJ), que se entrega como parte de Eclipse (y que son usados también para desarrollar el mismo Eclipse). Sin embargo, también se puede usar para otros tipos de aplicaciones cliente, como BitTorrent o Azureus.

Eclipse fue desarrollado originalmente por IBM como el sucesor de su familia de herramientas para VisualAge. Eclipse es ahora desarrollado por la "Fundación Eclipse", una organización independiente sin ánimo de lucro que, fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios.

- El primer paso es descargar Eclipse e instalarlo.

    1. Vamos al la pagina http://www.eclipse.org/downloads/ y descargamos la versión "Eclipse IDE for Java Developers" de 32bits.
    2. Ahora se descomprime y se ejecuta, ya que no necesita instalación.

- Pasos para la instalación de los complementos y paquetes necesarios para instalar Python en Eclipse.

    1. Desde Eclipse, nos vamos a la pestaña "help" y seleccionamos "new software".

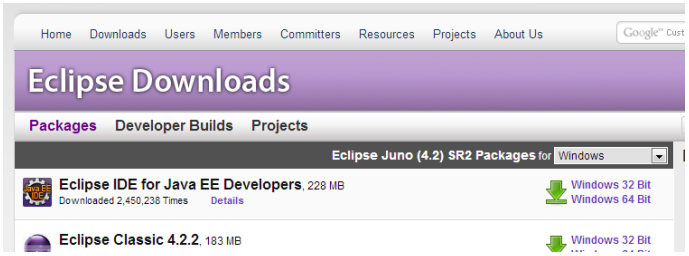


Figura A.1: Descarga de Eclipse

2. Copiamos la siguiente URL que encontramos en la pagina de PyDev (http://pydev.org/ → Downloads), debajo de "URLs for PyDev as Eclipse plugin" y seleccionamos la principal (main → http://pydev.org/updates).
3. La pegamos en Eclipse, donde pone "work with", a continuación seleccionamos "PyDev" y "PyDev Mylyn Integration" y los instalamos.
4. Creamos un nuevo work space o espacio de trabajo (New ws) y lo nombramos y situamos en el lugar donde deseemos guardar posteriormente nuestro proyecto.

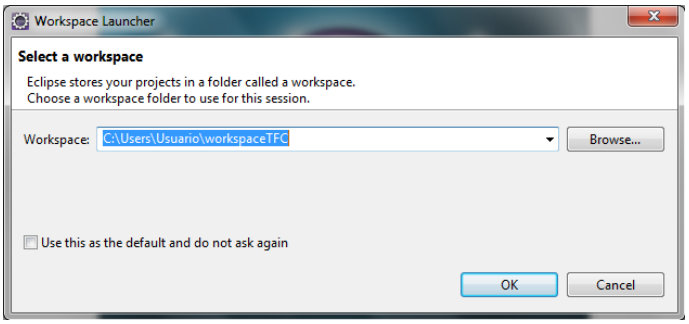


Figura A.2: Ventana "Workspace launcher"

5. Creamos un nuevo proyecto de tipo "pydev pj" con la siguiente secuencia: File → New → Project → Pydev Project.
6. Descargamos e instalamos un interprete de Python desde la web oficial

   (http://www.python.org/download/)

   Una vez llegado a este punto, seleccionamos la siguiente versión del intérprete: Python 2.7.3 Windows Installer (Windows binary -- does not include source).

   La instalamos en un lugar que recordemos y a continuación nos bajamos los siguientes paquetes, los cuales se instalarán por defecto en la carpeta en la que hemos instalado el interprete anteriormente.

Aquí se muestra como se tienen que instalar los paquetes necesarios para ejecutar el programa Forevid desde Eclipse. También se muestra donde descargar el código fuente del programa y como ejecutarlo. En el siguiente apartado se explica como se han instalando los paquetes que pide el programa cada vez que se intenta ejecutar el código fuente (PyQt4, PyOpenGL, PIL y reportlab). Se pueden instalar antes de la ejecución del código fuente para ahorrar tiempo.

- Instalación de los paquetes:

  1. Descargamos e instalamos el paquete PyQt4 el cual encontraremos en el siguiente link:

     http://ignum.dl.sourceforge.net/project/pyqt/PyQt4/PyQt-4.9.5
     /PyQt-Py2.7-x86-gpl-4.9.5-1.exe

  2. Descargamos e instalamos el paquete PyOpenGL en el siguiente link:

     http://pypi.python.org/packages/any/P/PyOpenGL/PyOpenGL-3.0.2.
     win32.exe#md5=4f1c66d6d87dcdf98ecc7ff4ce61a7e6

  3. Descargamos e instalamos el paquete PIL en el siguiente link:

     http://effbot.org/downloads/PIL-1.1.7.win32-py2.7.exe

  4. Descargamos e instalamos, por último, el paquete Reportlab desde el siguiente link:

     http://pypi.python.org/packages/2.7/r/reportlab/reportlab-2.6.
     win32-py2.7.exe#md5=4fdbb80bead7701c26f765f2a9c1ec19

- Descarga del código fuente de Forevid.

  1. Una vez hecho esto nos descargamos el código fuente de la pagina oficial de forevid “forevid.org”, cuya url es:

     http://sourceforge.net/projects/forevid/files/Releases/1.2.1
     /forevid-1.2.1-src.zip/download

  2. Y con el código fuente descargado, solo se tiene que importar a Eclipse de la siguiente manera *File* → *Import* → *General* → *File System* y seleccionar la carpeta que lo contiene.

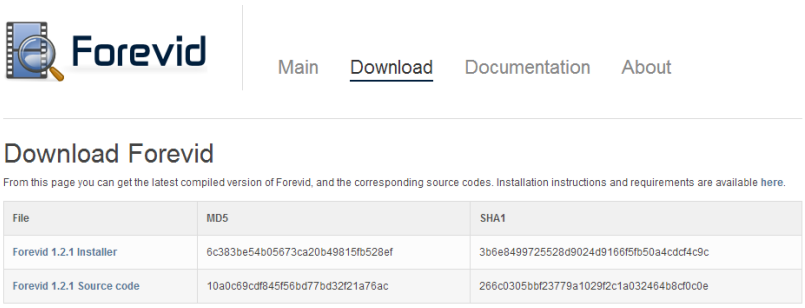


Figura A.3: Descarga del código fuente de Forevid

3. Ahora solo tenemos que ejecutar el archivo "forevid.py" (presinanado la tecla "Run" mientras esta seleccionado) para ejecutar el programa.

Por último, para crear el archivo DLL del pluguin en el capítulo 2, se descarga de la librería "Avisynth_258_src" de AviSynth a través de la siguiente página web:

http://sourceforge.net/projects/avisynth2/files
/AviSynth%202.5/AviSynth%202.5.8/

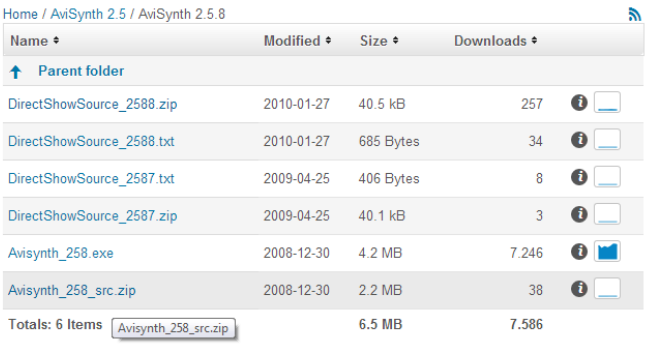


Figura A.4: Descarga de la librería AviSynth

Bibliografía

[1] Klaus Post Richard Berg Ian Brabham Ben Rudiak-Gould, Edwin van Eggelen et al. *AviSynth*. http://www.avisynth.nl, 2008.

[2] Riverbank Computing. *PyQt4 tutorial*. wiki.python.org/moin/PyQt4.

[3] Microsoft. *Microsoft Developer Network*. http://msdn.microsoft.com/es-es, 2010.

[4] Daniel Molkentin. *The Book of Qt 4: The Art of Building Qt Applications*. Paperback, 2007.

[5] National Bureau of Investigation. *Forevid*. http://www.forevid.org/.

[6] Fco. Javier Ceballos Sierra. *C/C++ Curso de programacion*. RA-MA, 2002.

[7] Guido van Rossum. *Python*. http://www.python.org/, 1991.