ESCUELA UNIVERSITARIA DE INGIENERÍA TÉCNICA
DE TELECOMUNICACIÓN

AMPLIACIÓN Y MEJORA DE APLICATIVO DE ANÁLISIS FORENSE DE IMÁGENES

PROYECTO FIN DE CARRERA

Juan Francisco Sanz Andrés

SEPTIEMBRE 2014

Agradecimientos

A mi madre,

Índice

Índice de figuras

1. Introducción

El desarrollo de dispositivos que permiten la captura o grabación de imágenes para diferentes usos (en el campo de la seguridad, vigilancia, comunicación vía internet mediante videoconferencia, etc.) ha provocado el desarrollo de numerosas herramientas digitales para su posterior análisis, interpretación, modificación, transporte o almacenamiento, ya sea en tiempo real o no. Las imágenes captadas, en la mayoría de los casos, son tratadas como datos que se interpretan y modifican, tratando de obtener a partir de una imagen origen, y mejorando o alterando ciertas características de la misma, otra que posibilite efectuar sobre ella otro tipo de operaciones de análisis o estudio, transporte o almacenamiento mediante técnicas digitales (filtrado, realzado, compresión, etc.).

Las técnicas a desarrollar que aquí ocupan se centran en aquellas que permiten compensar la falta de resolución o calidad en las imágenes de los dispositivos utilizados para la captura, producidas en muchas ocasiones por la baja calidad de los mismos o las condiciones en las que se efectúa dicha la grabación o toma de instantáneas. Como ejemplo de estos problemas, un caso clásico es el ruido en una imagen. Esta acepción tiene muchos significados, ya que se puede presentar de muchas formas y debido a diversidad de causas, las cuáles son importantes analizar para ayudar a realizar un tratamiento que minimice todo lo posible el impacto producido sobre la imagen, reducirlo de manera imperceptible e incluso en ocasiones eliminarlo por completo.

En el mundo del tratamiento digital existen numerosas herramientas que permiten realizar filtrados, realzados o redimensionados, muchas ellas de código accesible y modificable por cualquier usuario. Esto hace posible la creación de programas potentes basados en la integración de ideas de muchos individuos en un proyecto conjunto, aunando en un software de distribución libre diversas técnicas. En este caso se utilizó Forevid[1], un software desarrollado por el Laboratorio Forense de la Oficina Nacional de Investigación de Finlandia y cuya UI se programó en lenguaje de programación interpretado Python[2], de libre distribución e incluido en muchos sistemas operativos Linux, lo que posibilita la inclusión de módulos de mejora que se añaden a la versión base en un proceso de aprendizaje corto y sencillo para programadores con experiencia en este tipo de entornos de desarrollo. Avisynth[3] es la herramienta empleada para la elaboración de plugins de tratamiento de video, organización no lúdica creada en la red y ampliada por usuarios anónimos de la misma que comparten su tiempo y esfuerzo a este fin. Cabe decir que este tipo de librerías, como ya sucede con la UI y siguiendo la misma filosofía, se implementan en lenguaje C++.

Se considera la necesidad de incluir diversas técnicas básicas en la edición y análisis de video, así como corregir ciertas funcionalidades o completar las ya provistas del software. Además se propuso mejorar el funcionamiento a nivel de interfaz de usuario,

[1] www.forevid.org
[2] www.python.org
[3] www.avisynth.nl

entre ellas una versión traducida al español, descripción de texto en los botones de reproducción o inclusión de atajos en la pantalla principal de la aplicación. En el plano de tratamiento de vídeo se añaden herramientas básicas: filtrado en el dominio espectral, cambios de perspectiva, transformada discreta del coseno, ajustes básicos (tono, saturación, contraste…) o convolución espacial con matrices personalizadas, que se detallarán en el transcurso del documento.

Esta no es más que una pequeña aportación de las numerosas que se pueden seguir implementando y coleccionando en pro de ampliar campos de uso a una herramienta diseñada con perspectivas de evolución.

2. Procesamiento Digital de la Imagen

2.1. Digitalización de la imagen

Fundamentado en la captación de la realidad que el ser humano hace del entorno (formas, contrastes, colores, brillo, etc.) a través del sentido de la vista, la grabación de imágenes trata asemejar este comportamiento artificialmente, congelando imágenes semejantes a la percepción que el ser humano hace de la realidad. Esta cadena tecnológica ha ido evolucionando desde transformar la reflexión espectral de las superficies de los objetos en valores de gris o color de acuerdo a la intensidad de la radiación recibida, hasta la exploración de la superficie por un foto-sensor que recoge muestras de la radiación proveniente de la misma a intervalos regulares.

El encargado de dicha tarea es el CCD (Charge Coupled Device), constituido por una matriz de miles de fotoceldas microscópicas o sensores que convierten la luz en un voltaje eléctrico que se almacena momentáneamente en pequeños capacitores hasta pasar a un convertidor análogo – digital o ADC donde es convertido en código binario discreto. Las series numéricas obtenidas de esta conversión son enviadas a un microprocesador de señales digitales (DSP), el cual ajusta el contraste y detalle, y comprime la imagen antes de que sea almacenada en la memoria de la cámara y, usualmente, más tarde transferida a una computadora donde la imagen puede ser vista y manipulada, resultante una matriz discreta de valores numéricos que representan niveles de gris o color.

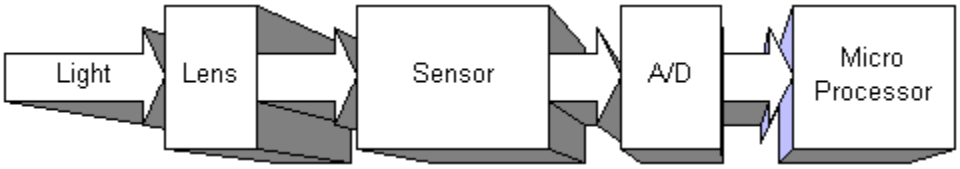


Figura 2.1: Cadena de captación de una imagen digitalmente

Normalmente un pixel contiene información en un rango valores que depende del número de bits utilizados en su cuantificación, teniendo una imagen monocromática. Cada banda espectral de un sensor multi-espectral es una imagen monocromática que podemos combinar con otras bandas para producir una imagen color, muestreando el margen de frecuencias con mayor fotosensibilidad para las células del ojo humano; azul (0,4 a 0,5 μm), verde (0,5 a 0,6 μm) y rojo (0,6 a 0,7 μm). A estos colores se les denomina primarios de luz ya que su combinación permite la representación de todos los colores del rango visible en el espectro de frecuencia.

La digitalización de una imagen o captura de video es obligatoria para su posterior procesamiento en el terreno digital, ya que se puede considerar la fuente de información con la que vamos a trabajar a posteriori, y por lo tanto de este proceso dependerán la dificultad para ejecutar transformaciones que se pueden realizar, elección de aquellas que sean más conveniente y el resultado final. El número de muestras, bits de resolución para cada muestra o si se trata de una imagen monocromática o a color son

características básicas a tener en cuenta a la hora de trabajar con contenido de este tipo, por lo que se le otorga una nomenclatura general:

- MxN: Tamaño de la imagen tomada como una matriz bidimensional, donde M es el número de pixel por fila de la imagen y N el número de pixel por columna. Cada pixel de la imagen es ubicada por unas coordenadas x e y, habitualmente tomado como origen el extremo superior izquierda.
- k: Número de bits de cada muestra o profundidad del pixel y que determinará los niveles de brillo posibles para cada pixel. Una imagen estándar usa 8 bits, 256 niveles, dicho así porque es el mínimo número direccionable por la mayoría de los microprocesadores. El número de planos de la imagen determinará si para cada pixel se mantienes estos 8 bits por pixel en caso de una imagen monocromática, determinando su plano de luminancia de forma habitual o brillo en escala de grises, o bien se trata de una imagen tricromática o a color, componiéndose de tres valores de brillo por pixel o tres planos (RGB o YUV), cada uno de 8 bits, sumando un total de 24 bits por pixel.

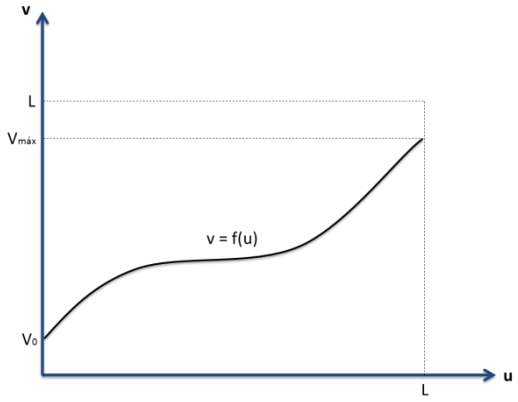
En el tratamiento digital cada valor de un pixel en la imagen modificada normalmente dependerá de sí mismo, adyacentes a él y/o variables externas. En el caso de videos se añade la variable tiempo, lo que significa que también puede variar en función de las imágenes previas y posteriores a la actual.

2.2. Introducción al procesamiento de imágenes

Durante el punto anterior se observa que la imagen es caracterizada por su tamaño donde tenemos: m filas y n columnas, y con p nº de bits para la cuantificación de una muestra. Cada punto de la matriz es un pixel, por lo que el número de pixeles es MxN, definiendo la resolución de la imagen, y cada pixel está cuantificado en un número determinado de bits definiendo desde 0 a 2^k posibles valores lo que define la calidad de la imagen.

Cuando la operación se aplica de la misma forma a todos los pixeles de la imagen, uno tras otro, se denomina operaciones puntuales o globales, ya que el nuevo valor de un pixel depende exclusivamente del valor que tenía anteriormente:

Se puede definir toda operación puntual como función del valor de la muestra original que es modificada por un parámetro externo:



$$(x, y) = f(x, y)$$

Figura 2.2: Función de transferencia real

Con x e y como coordenadas espaciales que asocian la ubicación a un valor o nivel emprendido entre 0 y 255, en el caso de utilizar dígitos binarios de 8 bits para cada uno.

En las operaciones locales todos los pixeles vecinos al pixel a ser trabajado son tenidos en cuenta para obtener el nuevo valor, los filtros espaciales con una ventana que se desplaza por la imagen es un caso típico.

El objetivo es desarrollar algoritmos que mejoren la calidad y/o la apariencia de la imagen original resaltando ciertas características (bordes, contraste…) y ocultar o eliminar otras (como el ruido) en una etapa previa para posteriores fases de análisis (segmentación, extracción de características, reconocimiento e interpretación).

Los principales tipos de procesados se pueden dividir en realzado y filtrado.

2.3. Realzado

Habitualmente se buscan propósitos de modificación de rango de niveles de brillo específicos limitando, recortando o invirtiendo sus valores, por lo que muchas de estas operaciones se evalúan desde la observación del histograma de la imagen. El histograma es la representación gráfica, rápida e intuitiva de la frecuencia con la que cada nivel de gris aparece en una imagen. Se trata de una herramienta fundamental para el análisis que permite condensar información sin tener en cuenta la localización espacial. En ella se sitúa el rango dinámico (conjunto de niveles de gris presentes) a lo largo del eje horizontal y el recuento de píxeles para dicho valor de brillo en el eje vertical, informando entre que valores de brillo está comprendida la mayoría de pixeles, o el equitativito reparto de valores de gris a lo largo de su rango dinámico. Las modificaciones del histograma se pueden visualizar mediante funciones de transferencia, que corresponden a curvas acotadas en abscisas y ordenadas entre 0 y 1 que se van a definir a continuación.

2.3.1. Operaciones de punto

Son aquellas en las que cada pixel de la imagen modificada depende únicamente del valor de brillo del pixel homólogo, por lo que su posición no es alterada. Este tipo de operaciones se suelen expresar como ecuación por tramos, donde cada brillo la imagen

"salida" se obtiene en función de la imagen "entrada": $v = f(u)$. Se definen diferentes tramos de comportamiento para cada valor de brillo, y como explicación general se van a desarrollar tres, no siendo en absoluto la única posibilidad como se verá:

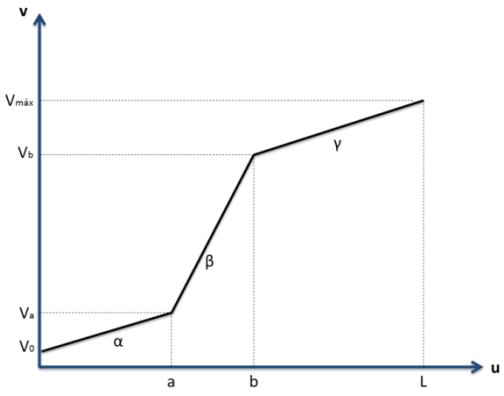


Figura 2.3: Función por tramos de transferencia

$$v = f(u)\begin{cases} \propto u + V_0, & 0 \le u \le a \\ \beta(u-a) + V_a, & a \le u \le b \\ \gamma(u-b) + V_b, & b \le u \le L \end{cases}$$

- α, β, γ; Constante escalar que determinan la pendiente de ese tramo de la función
- a, b; Niveles de brillo para u a partir de los cuales varía la pendiente de la función de transferencia
- V_o, V_a, V_b, V_{255}; niveles de v para cada valor correspondiente de u
- L; máximo valor de brillo

La mayoría de casos que se pueden encontrar pueden considerarse particulares de esta con diferentes valores, recibiendo un nombre específico atendiendo al resultado obtenido o forma de actuación, que se pueden visualizar mediante funciones de transferencia correspondientes a curvas acotadas (entre valores de 0 y 1 normalmente) tanto en abscisas como en ordenadas. Las más utilizadas son:

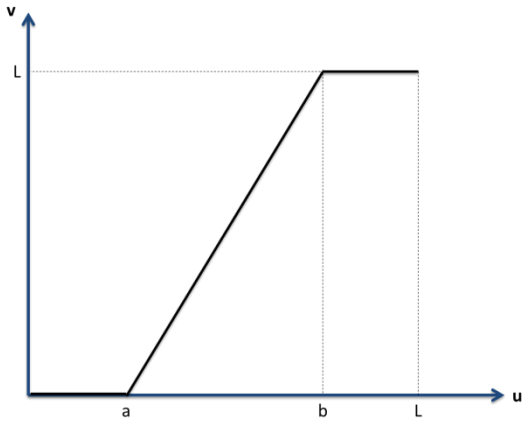


Figura 2.3: Clipping por ambos lados

- **Clipping**: realiza una ampliación de contraste en aquellas fotos cuyos valores de máximo y mínimo brillo sean cercanos entre ellos.

$$v = \begin{cases} 0, & u < a \\ \frac{L(u-a)}{(b-a)}, & u < a < b \\ L, & u > b \end{cases}$$

Existe clipping por ambos lados como en el ejemplo anterior, además de superior e inferior.

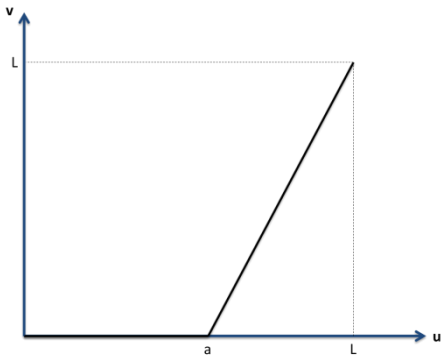


Figura 2.5: Clipping superior

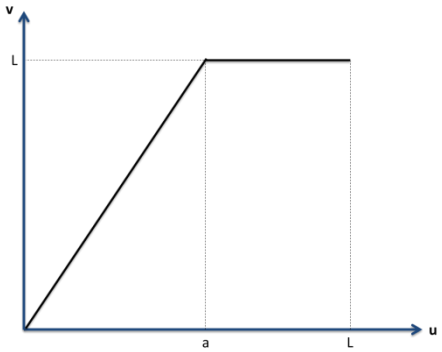
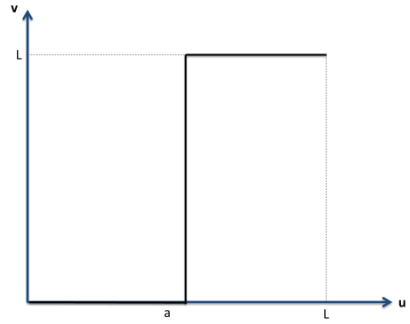


Figura 2.4: Clipping inferior

- **Binarización**: genera una imagen con dos valores de brillo (cero o L) donde el valor de a marca el umbral.



$$v = \begin{cases} 0, & u < a \\ L, & u \geq a \end{cases}$$

Figura 2.6: Binarización

- **Umbralización**: Se respetan los valores de u intactos salvo una porción, que tendrá valores de cero o L, o un tramo de cada valor. Si se mantienen los valores de u para v desde un valor hasta L se habla de umbralización por la parte inferior, y en caso de que se respete desde el mínimo brillo hasta un valor se denomina superior.

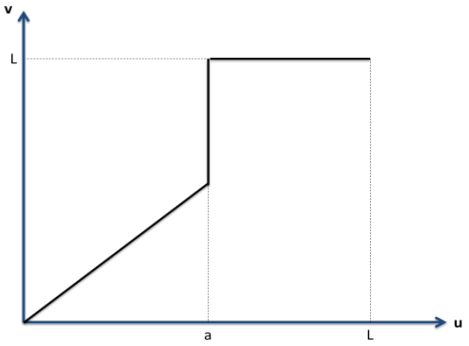


Figura 2.7: Umbralización superior a L

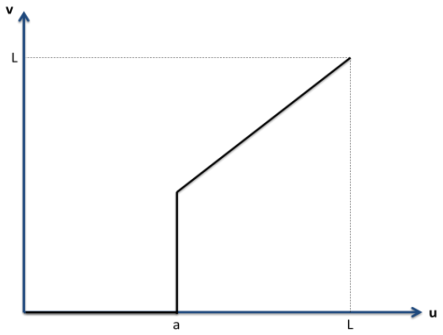


Figura 2.8: Umbralización inferior a L

Es posible forzar el brillo al valor contrario al expuesto en los casos anteriores, aunque su efecto corresponde menos con la imagen original y es más poco habitual encontrar este tipo de efecto. Estas serías sus funciones de transferencia correspondientes:

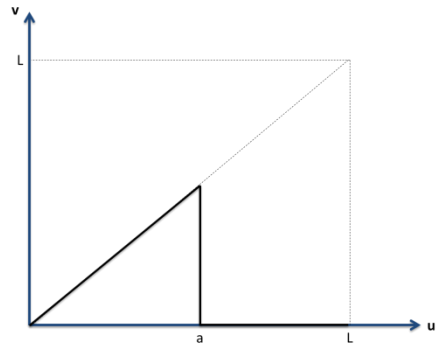


Figura 2.10: Umbralización superior a 0

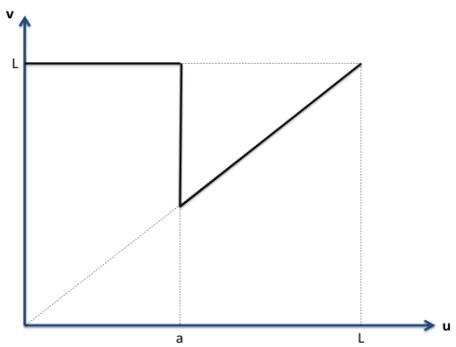


Figura 2.9: Umbralización inferior a 0

- **Sciling**: Se resaltan ciertos valores de brillo comprendidos entre dos valores ajustándolos a cero o a L, y manteniendo el resto de brillos intacto. Tiene la posibilidad de conservar el fondo o eliminarlo, realizando la operación contraria a la anterior,

forzando todos los valores de brillo a blanco o negro que estén comprendidos entre dos valores y mantener los que sí estén comprendidos entre ambos.

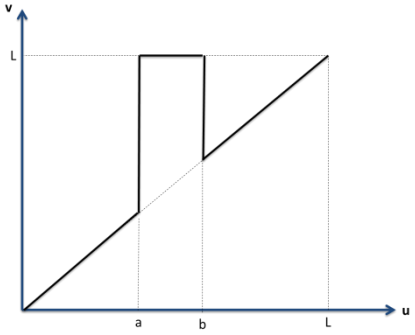


Figura 2.12: Sciling a L manteniendo el fondo

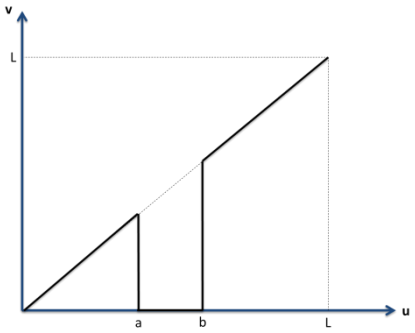


Figura 2.11: Sciling a 0 manteniendo el fondo

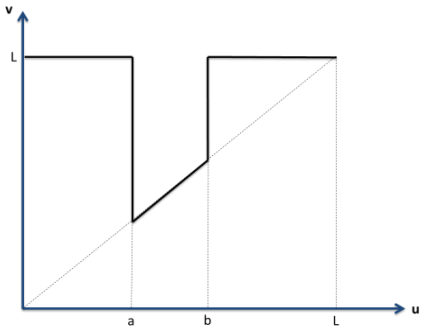


Figura 2.14: Sciling a L eliminando el fondo

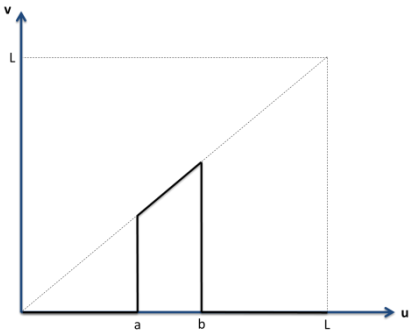


Figura 2.13: Sciling a 0 eliminando el fondo

- **Separando franja y resto**: Es una binarización especial que tratar de resaltar una determinada franja de brillos otorgándola un valor máximo o mínimo, contraponiéndolo al resto de valores de brillo asignándoles el contrario.

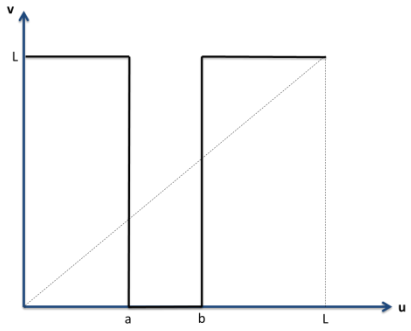


Figura 2.16: Separación franja a 0

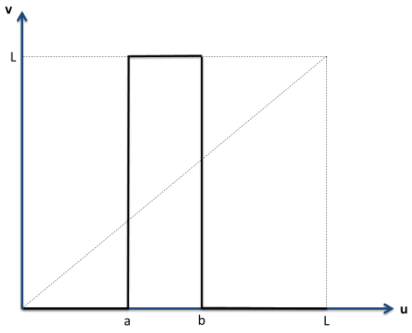
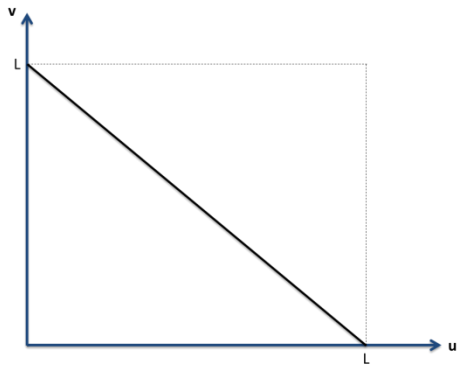


Figura 2.15: Separación franja a L

- **Complementación**: o efecto negativo. Muy conocido y utilizado en artes gráficas de fotografía, invirtiendo los valores realizando la diferencia entre el máximo nivel y el valor propio del pixel.

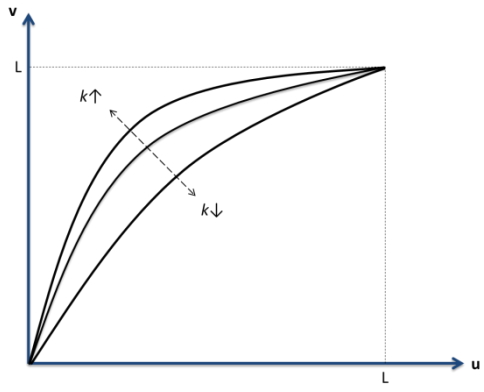


$$v = L - u$$

Figura 2.17: Efecto negativo

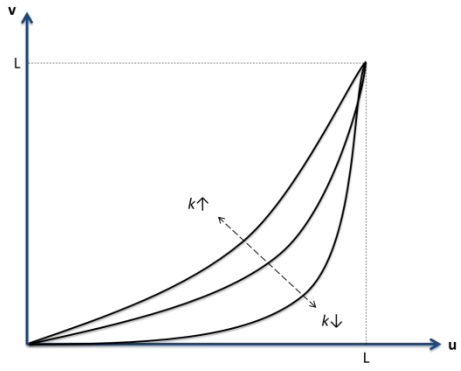
- **Logarítmico, exponencial y gamma**: Todas ellas atienden a funciones de transferencia sin una pendiente por tramos constante.

El primer caso pretender aumentar el contraste de los brillos de valores bajos de gris, en detrimento de los niveles altos. Como función contrapartida, la función exponencial realiza una tarea de "expansión" para los valores altos de brillo. Ambas operaciones dependen de una constante *k* que ajusta las curvas para un mayor o menor efecto del resultado final.



$$v = \frac{L\left(log\left(\frac{u}{L}(e^k - 1) + 1\right)\right)}{k}$$

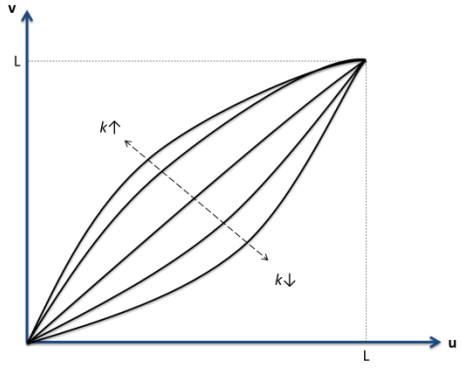
Figura 2.18: Función logarítmica



$$v = Lk\left(e^{\frac{ulog\left(1+\frac{1}{k}\right)}{L}} - 1\right)$$

Figura 2.19: Función exponencial

Gamma trata de ajustar diferencias de brillo que se producen entre diferentes dispositivos de visualización.



$$v = L\left(\frac{u}{L}\right)^{\frac{1}{k}}$$

Figura 2.20: Función gamma

2.3.2. Operaciones aritméticas

Comparten varias de las características con las operaciones de punto, ya que el valor del pixel en la imagen de salida dependerá de su valor de entrada y una constante, invariando la posición. Utilizando el histograma de la imagen se puede interpretar rápidamente la operación que se debe realizar para adecuar el resultado que se desea obtener. Unos ejemplos:

- **Suma:** $v = u + k$,incrementa linealmente la luminancia de cada pixel desplazando el histograma hacia la derecha un valor k. Vigilar que no se produzca overflow o saturación sobrepasando el valor de L.
- **Resta:** $v = u - k$,disminuye linealmente la luminancia de cada pixel desplazando el histograma hacia la izquierda k pixeles.
- **Producto:** $v = u \cdot k$, incrementa proporcionalmente la luminancia de cada pixel "estirando" hacia la derecha el histograma de la imagen resultante. El efecto de multiplicar cada pixel por una valor 0 < k < 1 es idéntico a la división, realizando efecto "encogimiento" del histograma de la imagen resultante.

Todas ellas son transformaciones lineales que se pueden definir de forma genérica como: $v = a \cdot u + k$, aunque también hay operaciones no lineales:

- **Logaritmo:** $v = klog(1 + u)$, aumenta el contraste de las zonas oscuras en detrimento de las zonas claras
- **Exponencial:** $v = ke^{(u-1)}$, aumenta el contraste en las zonas claras en detrimento de las zonas oscuras

2.3.3. Operaciones lógicas

La principal diferencia se debe a que intervienen dos o más imágenes $v = f(u_1, u_2 \ldots u_n)$, en lugar de una como sucedía anteriormente, para marcar el valor de la imagen resultante, siguiendo unas reglas de carácter sencillo, pero con múltiples combinaciones posibles. Al menos una de las imágenes debe ser binaria para que la operación tenga un cierto sentido, interpretando el valor 0 como negativo, y 1 o L como positivo, dicha imagen se denomina máscara k. Su uso se centra en análisis de formas y

transformaciones de imágenes binarias. A continuación se detallan algunas de las operaciones más básicas de conjuntos.

- **Complemento**: Complementa el conjunto, conteniendo todos los elementos que no pertenecen a A: $v = \boldsymbol{NOT}\ A$
- **Unión**: En dos imágenes binarias se selecciona el pixel si al menos uno de los valores es "1"borra los bits que se indique en la máscara: $v = A\ \boldsymbol{OR}\ B$
- **Intersección**: Extracción de los bits que estén en "1" en ambos juntos: $v = A\ \boldsymbol{AND}\ B$
- **Exclusiva**: Selecciona los pixeles sólo cuando los valores son diferentes entre ambos: $v = A\ \boldsymbol{XOR}\ B$

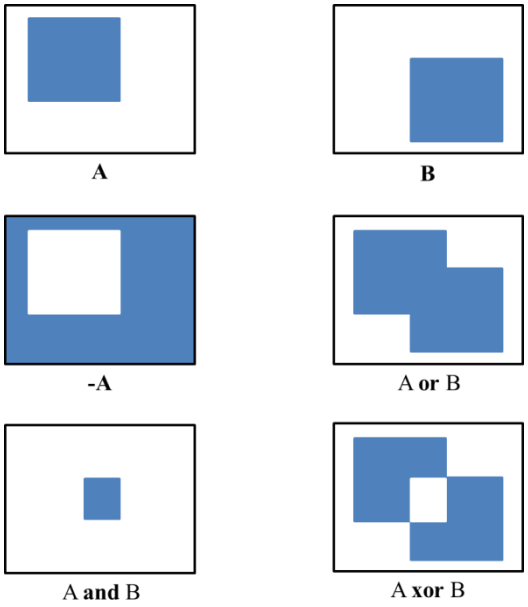


Figura 2.21: Representación gráfica de operaciones lógicas

2.3.4. Transformaciones morfológicas

El hecho de determinar objetos o formas en la imagen posibilita el tratamiento individual a cada una de estas formas reconocibles para la eliminación de detalles o píxeles corruptos que pueden incluirse dentro del objeto para simplificación y/o clarificación de la misma, por ello estas transformaciones se realizan en el marco del lenguaje morfológico basado en la teoría de conjuntos, la geometría y la forma. Estas aplicaciones van desde el suavizado de bordes, separación o unión de regiones unidas modificadas tras la segmentación, o enumeración de las regiones de una imagen, además de simplificar los datos de la imagen preservando las características relevantes de las mismas. Son aplicables tanto para imágenes binarias y, a pesar de la idea inicial que posteriormente fue desarrollándose, como a las de varios niveles de grises trasladando las técnicas de conjuntos, aunque en este proyecto se van a desarrollar ejemplos basándose en imágenes de dos niveles de brillo. Por la actuación e interpretación de las formas y conjuntos por supuesto se tratan de filtros no lineales, que interpretan dos subconjuntos Ay B sobre un espacio Z2 al tratarse de imágenes binarias (Z3 para subconjuntos de niveles de grises).

Vamos a repasar las operaciones más básicas para el tratamiento de conjuntos (dilatación, erosión, apertura y cierre), a partir de las cuales se derivan otras que no se verán en este desarrollo (esqueletización, adelgazamiento, ensanchado, granulación, etc).

2.3.4.1. Dilatación

Sean A y B dos conjuntos en Z2, la dilatación de A con B, denotada como A⊕B, se define como la ampliación de un subconjunto B a través de un elemento A que es el elemento que dilata conocido como elemento estructurante de la dilatación. Cuando un elemento del campo que crece tiene un valor común con el otro campo, este último se convierte en su posición de referencia del subconjunto de dilatación:

$A \oplus B = \{c \in Z2 \mid c = a + b, \text{ para algún } a \in A \text{ y } b \in B\}$

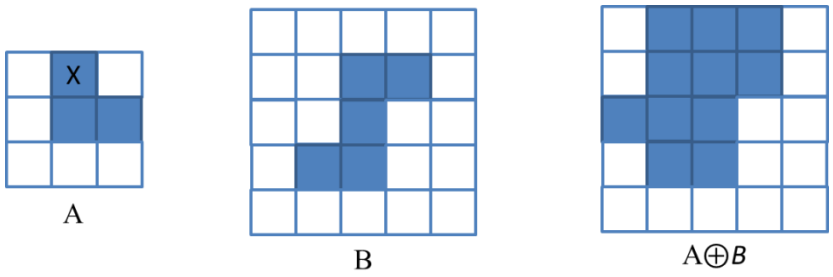


Figura 2.22: Ejemplo dilatación con elemento incluido en estructura dilatante

En este primer ejemplo el elemento estructural de dilatación contiene al origen como manera más sencilla de los posibles casos, pero puede suceder que esto no sea así como se denota a continuación:

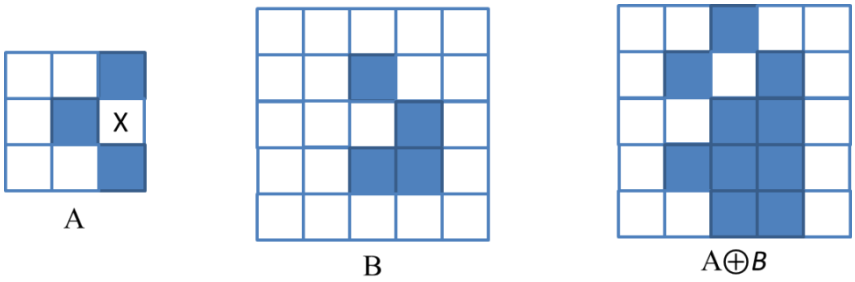


Figura 2.23: Ejemplo dilatación sin elemento incluido en estructura dilatante

2.3.4.2. Erosión

De forma contraria, la erosión trata la reducción de uno de los subconjuntos mediante otro subconjunto degradante. Sean A y B dos conjuntos en Z2, la erosión de A con B, denotada como AΘB, será el campo que cumpla que todos los elementos del conjunto degradante pertenezcan al degradado:

$A\Theta B = \{x \in Z2 \mid x + b \in A \text{ y } b \in B\}$

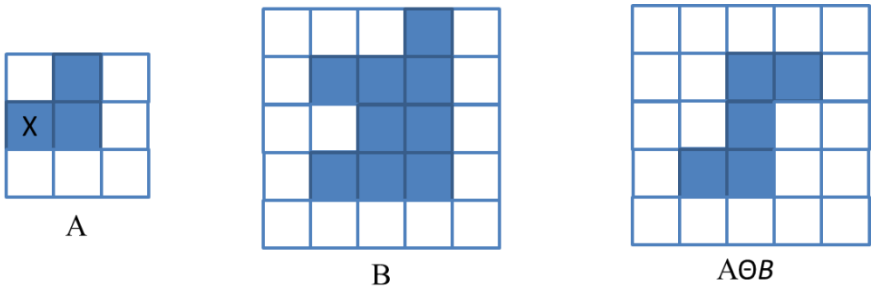


Figura 2.24: Ejemplo erosión con elemento incluido en estructura erosiva

Como en la operación anterior, es posible que el elemento estructural no contenga al origen:

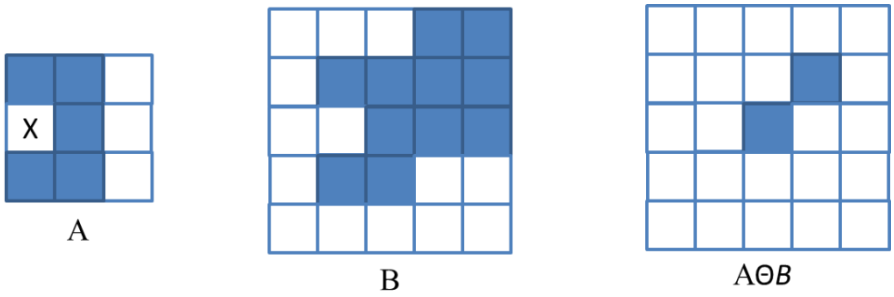


Figura 2.25: Ejemplo erosión sin elemento incluido en estructura erosiva

2.3.4.3. Apertura

Se trata de realizar una operación de erosión seguida de una dilatación. Se define como: $A \circ B = (A \ominus B) \oplus B$

Su efecto, dependiendo del elemento estructural utilizado, es el de rotura de pequeños enlaces entre conjuntos o suavizado de contornos.

2.3.4.4. Cierre

Es la operación dual a la de apertura, realizando en este caso primero una dilatación seguida de una erosión. Se define como: $A \bullet B = (A \oplus B) \ominus B$

Su efecto también es el de suavizado de contorno pero agregando al conjunto nuevos valores rellenando los huecos o agujeros y completando los espacios dentro del conjunto.

2.3.5. Operaciones geométricas

A diferencia de las anteriores técnicas operativas, donde se modificaban brillos o valores de pixel, en este tipo de operaciones se transforman las relaciones espaciales entre píxeles de una imagen: traslaciones, rotaciones, diezmados, interpolaciones, cambios de perspectiva, etc. Se definen matrices de trasformación para este tipo de operaciones, pudiendo ser:

- **Traslación**: $\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$
- **Escalado**: $\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$
- **Rotación**: $\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$

Dependiendo del tipo de operación que se realiza, es más conveniente utilizar uno u otro, si bien para reflexiones o simetrías se debería utilizar traslación, y para giros es más apropiada la forma matricial de rotación.

- Introducimos conceptos de **simetría** en espejo básicos, que producen un efecto reflejo de la imagen, bien puede ser horizontal o vertical:

Simetría Horizontal en campo discreto:

$$\begin{bmatrix} m' \\ n' - 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & N \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} m \\ n \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} m' \\ n' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & N \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} m \\ n \\ 1 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

Simetría Vertical en campo discreto:

$$\begin{bmatrix} m' - 1 \\ n' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & N \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} m \\ n \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} m' \\ n' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & N \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} m \\ n \\ 1 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

- **Traslaciones** de imágenes, deben tener un fondo o tamaño capaz de asimilar los desplazamientos y alojar la imagen completa en otra posición de la misma.

$$\begin{matrix} x' = x + \Delta x \\ y' = y + \Delta y \end{matrix} \rightarrow \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \Delta x \\ 0 & 1 & \Delta y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- **Escalado:** determinado por el valor k, y cuyo valor unidad no modifica la imagen obtenida, por lo tanto, para valores 0<k<1 la imagen quedará "comprimida" y para valores k>1 ésta se reescalará a tamaños mayores.

$$\begin{matrix} x' = k_1 \cdot x \\ y' = k_2 \cdot y \end{matrix} \rightarrow \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} k_1 & 0 & 0 \\ 0 & k_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- **Rotaciones:** para estos casos de utilizan matrices de rotación como método de anotación más propicio, y se valorará el ángulo de giro respecto de la posición inicial creando una posición relativa nueva de los ejes de coordenadas.

$$\begin{matrix} x_0' = x_0 \cos\alpha - y_0 \sin\alpha \\ y_0' = y_0 \cos\alpha + x_0 \sin\alpha \end{matrix} \rightarrow \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

Al realizar el giro de coordenadas se debe tener en cuenta que un giro en sentido horario de la imagen supone un giro anti-horario de los ejes (α negativo) y viceversa. Por lo que

para realizar giros de la imagen en sentido horario utilizando valores de alpha positivos se reescribe la ecuación:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Este método provoca que algunas posiciones de los píxeles (x´ e y´) no sean reales, y queden sin brillo o luminancia en la imagen modificada provocando un efecto de "huecos" negros. Para solucionar este efecto notable, se realiza la operación inversa, buscando para cada posición de la imagen final un valor de brillo interpolado hallado a partir de los píxeles más cercanos de la imagen original, originando posiciones reales pero valores de brillo mediadas.

- **Shear ó inclinación**: Inclinación de la imagen determinada por una cantidad *b*. Para un efecto horizontal:

$$\begin{matrix} x' = x \\ y' = y + x(b/M) \end{matrix} \rightarrow \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ b/M & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

En cambio, si se desea ejercer una inclinación vertical:

$$\begin{matrix} x' = x + y(b/N) \\ y' = y \end{matrix} \rightarrow \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & b/N \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

Mediante trasformaciones definidas en matrices se pueden realizar operaciones sucesivas aunadas en una única matriz obteniendo el resultado de la multiplicación de cada matriz de transformación, teniendo en cuenta que el orden de cada efecto afectará al siguiente y, por lo tanto, no es conmutativo.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = M_1 \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}; \begin{bmatrix} x'' \\ y'' \\ 1 \end{bmatrix} = M_2 \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}; \begin{bmatrix} x''' \\ y''' \\ 1 \end{bmatrix} = M_3 \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} x''' \\ y''' \\ 1 \end{bmatrix} = M_1 M_2 M_3 \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x^n \\ y^n \\ 1 \end{bmatrix} = M_N M_{N-1} \dots M_2 M_1 \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- **Transformación afín**: convierten un romboide en otro conservando paralelismos entre lados del mismo:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\begin{matrix} x_1' = ax_1 + by_1 + c \\ y_1' = dx_1 + ex_1 + f \end{matrix}; \begin{matrix} x_2' = ax_2 + by_2 + c \\ y_2' = dx_2 + ex_2 + f \end{matrix}; \begin{matrix} x_3' = ax_3 + by_3 + c \\ y_3' = dx_3 + ex_3 + f \end{matrix}; \begin{matrix} x_4' = ax_4 + by_4 + c \\ y_4' = dx_4 + ex_4 + f \end{matrix}$$

- **Transformación bilineal**: en este caso no se conserva el paralelismo entre líneas del romboide:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ xy \\ 1 \end{bmatrix}$$

- **Transformaciones de perspectiva**: se sitúa la imagen en un espacio tres dimensiones a una distancia y posición cualquiera, que se utiliza como patrón para trazar los rayos desde todos los puntos P de la misma hasta el centro de observación. A una cierta distancia focal un plano de proyección paralelo al plano XY servirá para ubicar las proyecciones de los puntos P´, como intersección entre dicho plano y los rayos. Las proyecciones de los puntos P = (x, y, z) vendrán dadas sobre el plano de proyección como P´= (x/z, y/z):

$$\begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \rightarrow x' = \frac{x_1}{z_1};\ y' = \frac{y_1}{z_1}$$

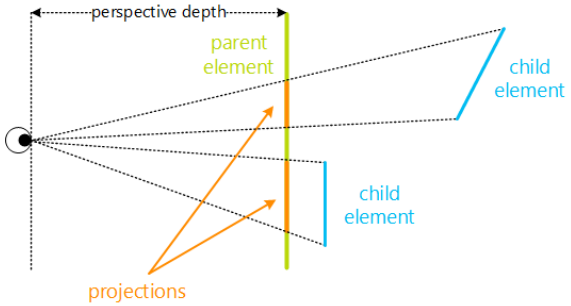


Figura 2.26: Demostración de planos para cambios de perspectiva

La proyección conserva las líneas, en el sentido de que una línea sobre un plano se transforma en otra línea del otro, aunque de forma distorsionada bajo el efecto de la perspectiva, por lo que habitualmente las líneas paralelas no se mantienen como tal. Debido a que se define un sistema de coordenadas en el espacio XYZ anteriormente explicado, el sistema de proyección se puede expresar como x´=Hx donde H es una matriz 3x3 invertible. Es posible por tanto revertir la deformación hallando la matriz inversa H y aplicándola sobre la imagen x´.

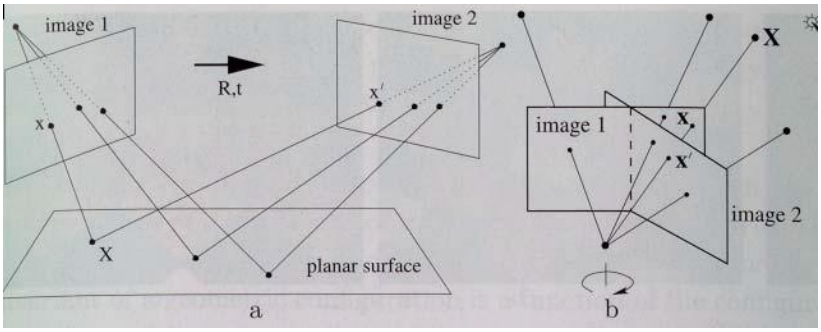


Figura 2.27: Reversión de trasformación por perspectiva

Figura 2.28: Ejemplo cambio de perspectiva

2.4. Filtrado

Persiguen la modificación de las imágenes con una finalidad concreta, normalmente asociada a la eliminación de una parte de la energía dentro del espectro de frecuencia que la caracteriza, o bien destacar bordes o zonas concretas para su posterior análisis. También se utiliza cuando se producen imperfecciones en la captura de una imagen o se modifica con la inclusión de ruido durante el transporte aéreo de que se utiliza como medio en la radiodifusión por ejemplo.

2.4.1. Filtrado espacial

Las operaciones de punto modifican cada pixel independientemente mientras que las operaciones locales lo modifican en función del valor de los pixeles vecinos.

Estas técnicas se usan como filtrado espacial o de convolución. Es posible también trabajar en el dominio de la frecuencia por ejemplo usando las transformadas de Fourier como veremos en el punto 2.4.4. La idea de hacer un filtrado de la imagen es reforzar o suavizar los contrastes espaciales existentes entre los valores de los pixeles, transformando esos valores para que se asemejen o se diferencien más a los pixeles vecinos.

El proceso de convolución citado anteriormente es el utilizado para la aplicación de filtros, definidas como matrices de valores, a imágenes, contenidas en el mismo formato. Como caso específico útil para el tema que se trata se procede con la variante bidimensional de esta técnica, sin embargo se trata de una herramienta generalizada en el uso de señales discretas unidimensionales. Vamos a comenzar por detallar el proceso de convolución para una dimensión por su mayor simplicidad, ayudando a comprender la desarrollada en dos dimensiones.

- **Convolución unidimensional**: dos secuencias (x[n] e y[n]) dan lugar a otra (z[n]) dada por la expresión matemática:

$$z[n] = x[n] * y[n] = \sum_{n'=-\infty}^{\infty} x[n']y[n-n'], \qquad -\infty < n < +\infty$$

El tamaño resultando de z[n] será: $N_3 = N_1+N_2-1$, siendo N_1 el número de muestras de x[n] y N_2 las de y[n]. Quizás parezca poco intuitivo a primera vista ejecutar esta operación, por lo que vamos a detallar un sencillo ejemplo que permita ver la metodología visual de una convolución. Se trata de desplazar una señal sobre la otra e ir multiplicando los valores emparejados de cada señal para realizar el sumatorio de todas las multiplicaciones obtenidas. Ya que, al inicio y al final del proceso algunos valores no tiene pareja de la otra señal con la que multiplicarse se considera un valor 0 o nulo para completar estos huecos. Como ejemplo supongamos x[n] = [1, 1, 2, 1, 1] e y[n] = [1, 2], para facilitar el proceso se selecciona la señal más corta para desplazarla (en nuestro caso y[n]) sobre la otra señal, ya que posee la propiedad conmutativa y es indiferente el orden de los factores, $z[n] = x[n] * y[n] = y[n] * x[n]$, pero con

simetría inversa en el caso de que alguna de las dos señales no posea esta característica (en este ejemplo y[n] no es simétrica):

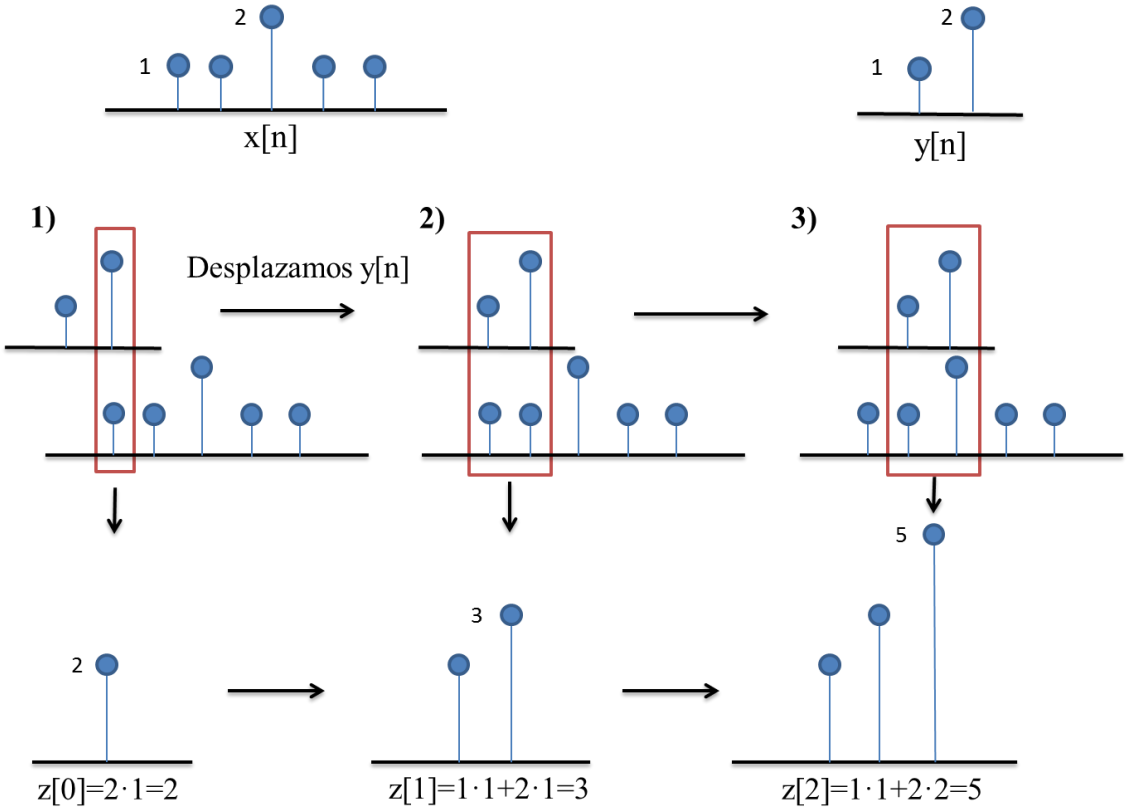


Figura 2.29: Ejemplo de proceso convolutivo unidimensional

El proceso finaliza cuando el último valor de x[n] es alcanzado por el primero de y[n], obteniendo como resultado z[n] siguiendo el mismo procedimiento descrito:

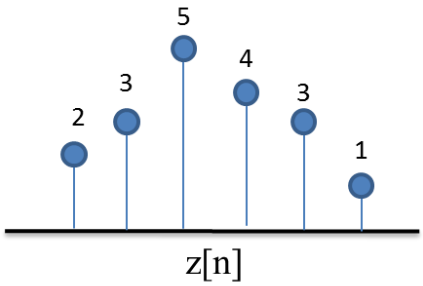


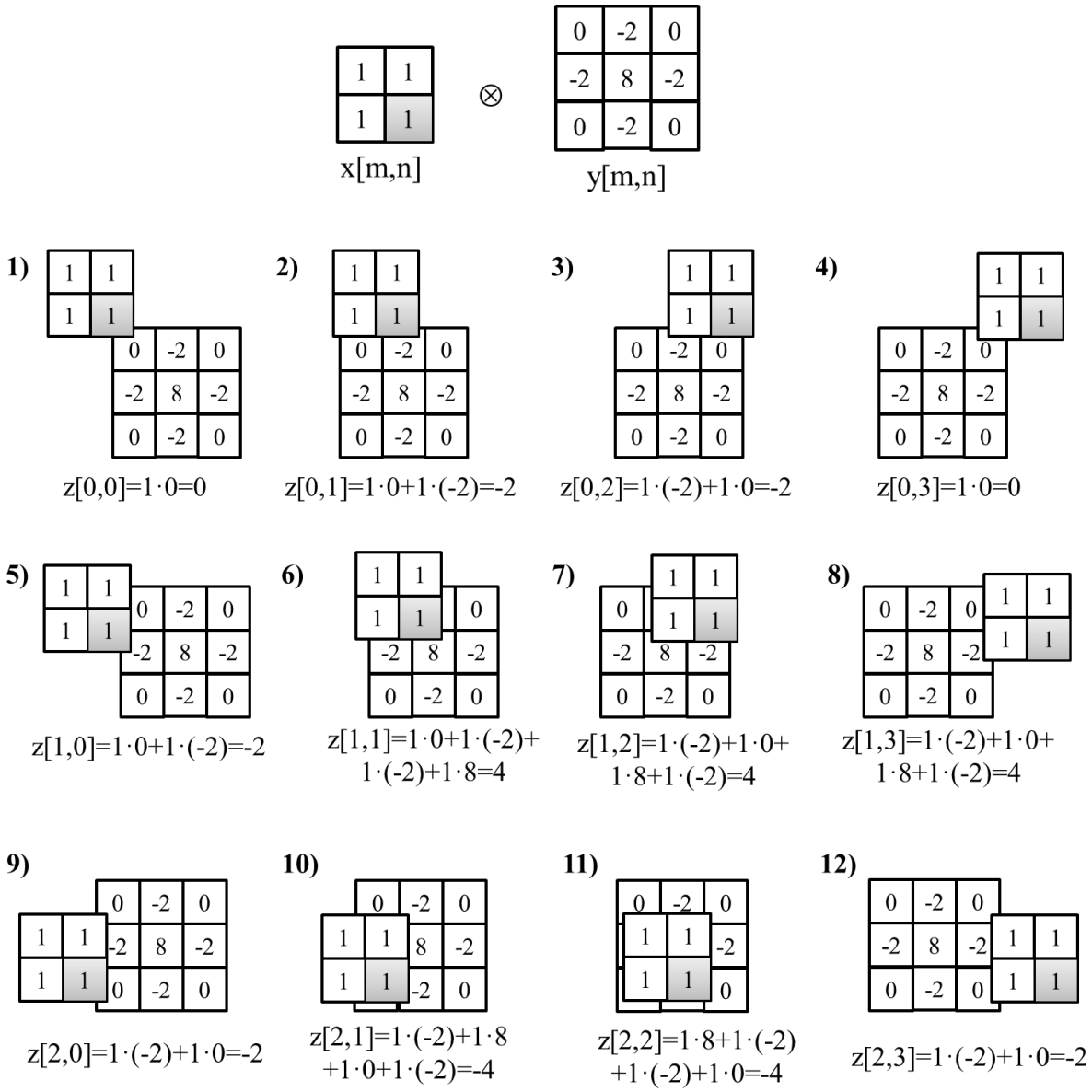
Figura 2.30: Resultado: x[n]*y[n]

- **Convolución bidimensional**: como se puede deducir de su expresión es posible realizar una convolución bidimensional a partir de dos convoluciones unidimensionales:

$$z[m,n] = x[m,n] * y[m,n] = \sum_{m'=-\infty}^{\infty} \sum_{n'=-\infty}^{\infty} x[m',n']y[(m-m'),(n-n')],$$
$$-\infty < m < +\infty, -\infty < n < +\infty$$

También, al igual que ocurre para una única dimensión, el tamaño resultante será de M_1+M_2-1 filas y N_1+N_2-1 columnas y posee la propiedad conmutativa. Esta operación se realiza en MATLAB con el comando *"conv2"*.

El proceso consiste en mover una pequeña ventana o filtro (filter kernel) a través de toda la imagen, calculando el valor del nuevo pixel a través de todos los valores de los pixeles dentro de ese filtro, eligiendo como punto ancla o de referencia una posición fija del filtro. Este filtro es una matriz de números que se usa para calcular el valor del nuevo pixel en función de los valores de sus vecinos, la matriz se desplaza un pixel a cada paso en la imagen de entrada guiada por su posición de ancla, obteniendo el valor para el pixel central del filtro en la imagen de salida, y así sucesivamente va calculando los puntos de salida para toda la imagen. Normalmente estas matrices son de 3x3 pixeles, 5x5 o 7x7 dependiendo del número de pixeles involucrados en la operación, y mientras más aumente mayor es el efecto del filtrado. En los filtros de tamaño impar, el núcleo suele ser el central, aunque no tiene porqué cumplirse esta norma, además para filtros con un tamaño par (2x2, 4x4, etc), donde no hay núcleo o valor central y no se podría aplicar. A continuación se realiza un pequeño ejemplo, desplazando la posición ancla de x[n,m] por todas y cada una de las posiciones de y[m,n] tomando ésta posición como referencia para la ubicación final del valor en z[m,n]:



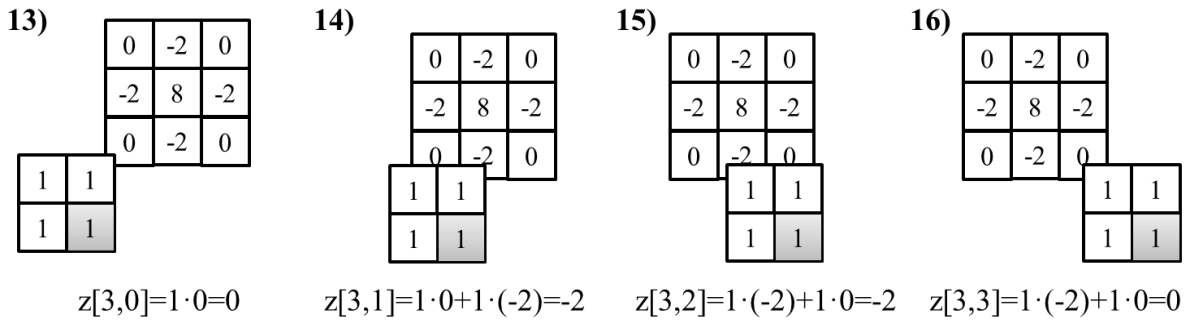


Figura 2.31: Ejemplo de proceso convolutivo bidimensional

Quedando z[n] con los valores:

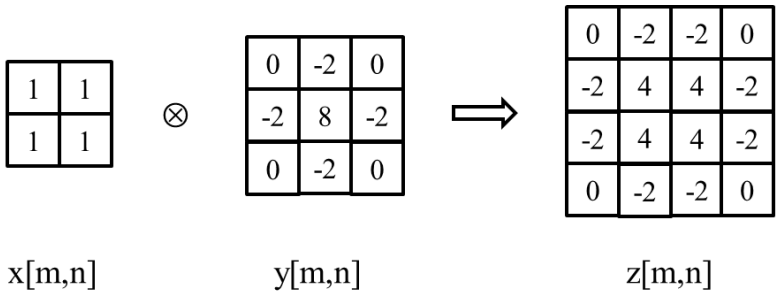


Figura 2.32: Resultado: x[m,n]*y[m,n]

El resultado combina las propiedades de ambas matrices, creando una máscara de perfilado no tan sensible al ruido, y su tamaño aumenta como se describía anteriormente en N_1+N_2-1 filas y M_1+M_2-1 columnas.

El efecto de estos filtros depende del tamaño de la matriz (número de filas y columnas) y de la distribución y del peso relativo que tengan los coeficientes dentro de la misma. Cuando el sumatoria de los valores difiere mucho de 0 se debe acompañar con un valor escalar que aplica a toda la matriz que impide la saturación de la imagen y truncamientos, dividiendo el valor final obtenido del sumatorio dentro del kernel por la suma de los cocientes de la matriz del filtro, por ejemplo, una matriz de media con todos sus cocientes unidad y tamaño 3x3, habrá que dividir por 9.

$$1/9 \cdot \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}$$

Como se ha dicho en el ejemplo expuesto, es posible combinar dos máscaras combinando las propiedades de ambos para obtener resultados específicos o extraer ciertas características al aplicarla sobre una imagen. El resultado no varía cuando se convolucionan ambas máscaras entre sí y posteriormente se aplica a la imagen, o se realiza el proceso convolucionando una máscara sobre la imagen, y sobre el resultado obtenido se aplica la segunda máscara. Este segundo método es el preferido debido a que es más sencillo y breve por el número de cálculos que se realizan convolucionar dos matrices de pequeño tamaño, que una pequeña con otra de grandes dimensiones.

2.4.2. Filtros suavizantes

Reciben este nombre al efecto que producen sobre la imagen, eliminando bordes y detalles, igualando los valores de brillos entre píxeles vecinos semejante al producido por un filtro paso bajo, y disminuyen el efecto producido por el ruido en imágenes muy afectadas. Se caracterizan por poseer coeficientes positivos y con valores cercanos para toda la matriz. El tamaño juega un papel importante, dando mayor protagonismo a los píxeles vecinos cuanto mayor es el filtro, produciendo un mayor difuminado. Hay que tener precaución al aplicar este tipo de máscaras debido a posibles truncamientos de brillo si no se aplica un valor de división a cada brillo obtenido cuando los coeficientes son altos.

2.4.2.1. Filtros Lineales

Funcionan como máscaras convolutivas con la imagen original que pueden desempeñar diferentes efectos que veremos a continuación como ejemplos más utilizados en la restauración y análisis del tratamiento digital de imágenes.

- **Media**: el más sencillo y habitual que puede representar de ejemplo como filtro de tipo suavizador, resaltando bajas frecuencias e igualando valores de luminancia entre todos los pixeles de la imagen. Este tipo de filtros, a pesar de ensuciar o difuminar los bordes, reducen la relación SNR de las imágenes afectadas por algún tipo de ruido introducido:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & \mathbf{1} & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

- **Suavizado direccional**: Puede interesar realizar filtrados direccionales en ciertas imágenes donde el ruido se presente de forma particular. Por ejemplo, en retransmisión se producen interferencias que afectan al entrelazado de las imágenes y oscilaciones de valores bruscos entre filas vecindarias. Para solucionar este tipo de problemas es necesario aplicar una matriz cuya forma no será rectangular:

$$\begin{bmatrix} 1 \\ \mathbf{1} \\ 1 \end{bmatrix}$$

- **Suavizado Gaussiano**: En ocasiones se producen efectos indeseados de deformación al aplicar este tipo de máscaras donde todos sus coeficientes tiene un peso equivalente entre ellos producidos por la forma cuadrada que aplica y su tamaño, para lo que se utilizan valores de media que minimizan este efecto, realizando una distribución de valores en la matriz filtro que otorgan un mayor peso a las posiciones más cercanas a la posición ancla o referencia. Una forma coherente de aplicar este reparto de valores es considerar una simulación de distribución gaussiana en forma de campana para una dimensión 2D:

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & \mathbf{4} & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

2.4.2.2. Filtros no lineales

Su aplicación difiere de la metodología aplicada en el punto 2.4.2.1. donde se usaba la convolución bidimensional entre matrices bidimensionales. La definición de filtros no lineales ofrece posibilidades de elección del valor del brillo considerando lógicas matemáticas o reglas preestablecidas para todos los pixeles que alberga la rejilla o máscara. Ya que no siguen reglas fijas, para una mejor interpretación de cada uno de ellos se ofrecen ejemplos ilustrativos.

- **Media geométrica**: Se realiza mediante el producto de los píxeles dentro de la máscara, elevando su resultado a $1/N^2$:

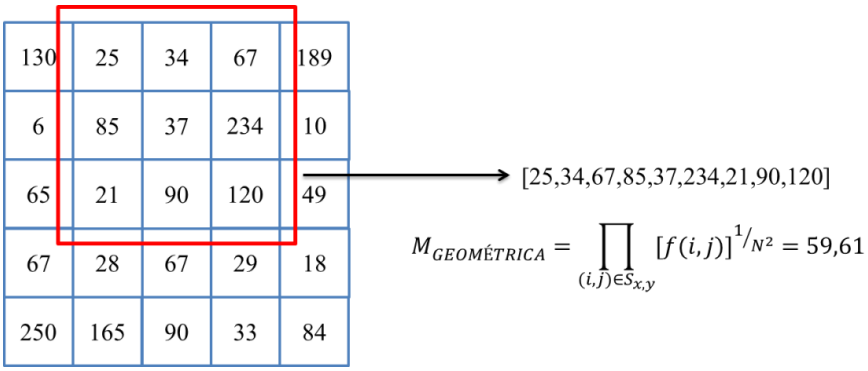


Figura 2.33: Ejemplo de Media Geométrica

No es muy efectivo para ruido de tipo sal y pimienta, pero trabaja bien con ruido gaussiano ya que retiene mejor los detalles de la información que el filtro de media aritmética.

- **Mediana**: Es otro tipo de filtros donde el pixel central se sustituye por el valor de la mediana del kernel, por lo que es afectado en menor grado que un filtro de promedios por los valores extremos (0,255) que muchas veces se deben sólo a valores erróneos de los pixeles, el llamado ruido tipo salt&peeper (sal y pimienta) de distribución aleatoria por toda la imagen. El proceso para este filtro es seleccionar el valor mediano que queda dentro de la ventana. Por ejemplo, para unos valores de: 23, 75, 58, 112, 90, 34, 210, 245 y 87, el valor seleccionado para la imagen de salida sería 87 y cuya posición será la central de la rejilla, que se irá desplazando a o largo de toda la imagen de entrada.

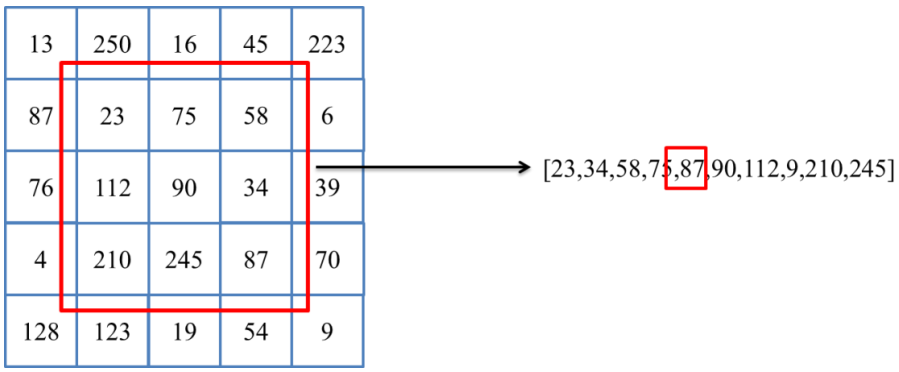


Figura 2.34: Ejemplo de Mediana

Su efecto no es muy reseñable en ruidos con distribución normal, o bien la relación SNR de la imagen es muy estrecha.

Los filtros de máximo o mínimo actúan de forma similar, otorgando al pixel de la imagen de salida el valor mayor (para el filtro de máximo) o menor (para el mínimo) dentro del kernel que recorre la imagen.

En imágenes binarias este tipo de filtros pueden llegar a reducir el estado defectuoso de una imagen mejorando ostensiblemente su calidad para posteriores procesos de análisis.

- **Moda**: Selecciona de la matriz el valor más repetido incluido dentro del kernel, colocándolo en la posición ancla dentro del mismo. La función de MATLAB *mode* que ejecuta esta operativa selecciona el menor valor encontrado dentro de la matriz en caso de que existan más de un valor repetido en el mismo número de ocasiones.

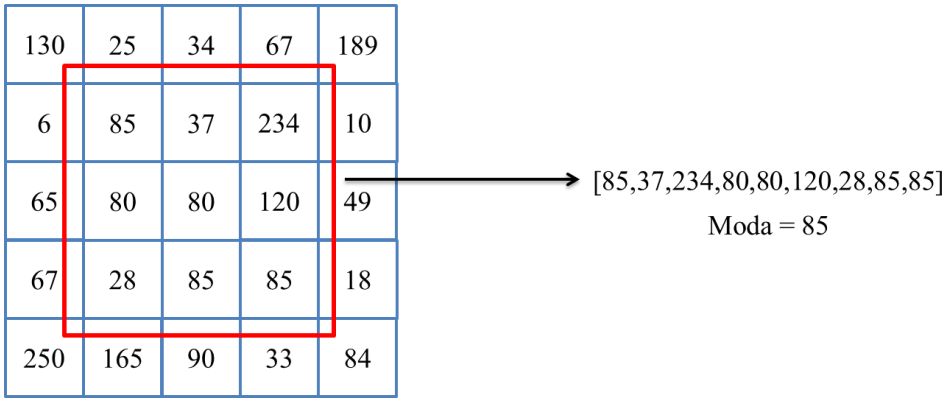


Figura 2.35: Ejemplo de Moda

Una de sus principales ventajas es el atenuamiento de ruido impulsivo sal y pimienta, sin embargo en muchas ocasiones, debido a que los niveles de brillo son diferentes para sus vecinos, el efecto es imperceptible en muchas ocasiones.

- **Máximo**: Muy útil para la eliminación de ruido tipo pimienta (ruido impulsivo donde ciertos pixeles tienen valores de negro) ya que selecciona el valor más alto dentro de la rejilla del filtro.

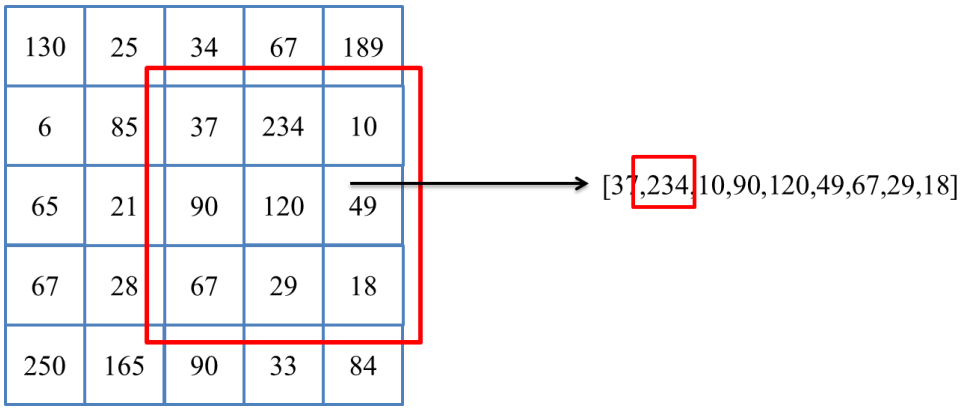


Figura 2.36: Ejemplo de Máximo

Esta máscara tiende a aclarar y modificar la imagen y únicamente funciona de forma muy efectiva para la eliminación del tipo de ruido descrito:

- **Mínimo**: Si anteriormente teníamos un filtro que seleccionaba el mayor valor, éste realiza lo complementario, estableciendo el valor mínimo en la imagen de salida en la posición ancla de la máscara.

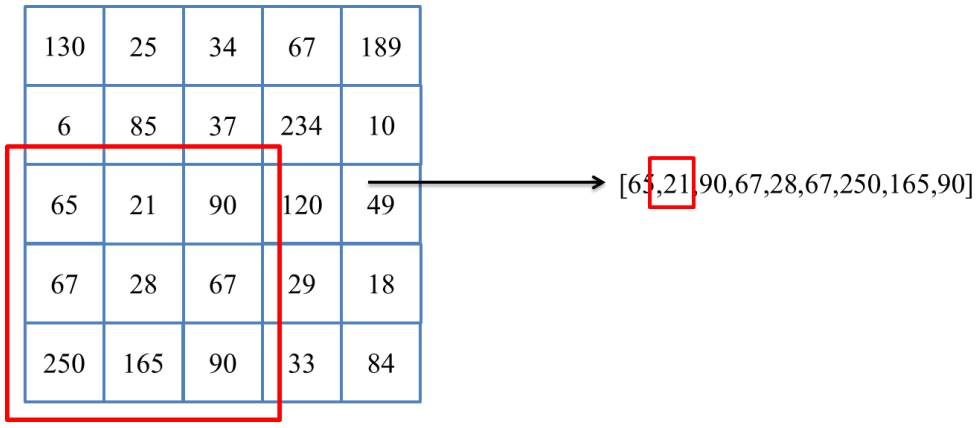


Figura 2.37: Ejemplo de Mínimo

Elimina el ruido impulsivo sal de forma eficiente y tiende a modificar la imagen de salida oscureciéndola.

2.4.3. Filtros agudizadores

Para algunas aplicaciones es conveniente realzar pequeños detalles o bordes, que son los componentes de alta frecuencia de una imagen. Si en el punto 2.4.2 se realizaban operaciones de suavizado con filtros de media, su equivalente contrario son los operadores derivativos o diferenciales que suponen un filtrado en alta frecuencia para el realzado de bordes y detalles.

Sirven para mejorar imágenes con problemas de detalles o desenfocadas, un ejemplo típico es el resalte o subrayado de los bordes, para dotarles de mejor nitidez.

2.4.3.1. Operador gradiente

El gradiente de una función continua f(x,y) es un vector que indica la dirección y cantidad de la máxima variación de un punto de la función. Para un espacio 2D, visualmente se puede describir como la dirección hacia la que iría el cauce un río que "busca" siempre la máxima inclinación en cada punto. El objetivo de los operadores gradiente es detectar cambios en los niveles de gris que tienen lugar en zonas o regiones reducidas, que coinciden con los máximos valores de un gradiente. El gradiente y la magnitud se expresan por:

$$\nabla f(x,y) = [G_x, G_y] = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

Módulo: $|\nabla f| = \sqrt{G_x^2 + G_y^2}$

Dirección: $\theta\nabla f = \tan^{-1}\left[\frac{G_y}{G_x}\right]$

Este operador está relacionado directamente con las derivadas parciales ortogonales (ejes x e y). Para el tratamiento digital se van a tener valores discretos, por lo que se deben adecuar estas funciones derivativas, que no es más que la diferencia entre píxeles adyacentes:

$$G_x = \frac{\nabla f}{\nabla x} = f(x,y) - f(x+1,y)(1), \qquad H_x = \begin{bmatrix} 1 & -1 \\ 0 & 0 \end{bmatrix}$$

$$G_y = \frac{\nabla f}{\nabla y} = f(x,y) - f(x,y+1)(2), \qquad H_y = \begin{bmatrix} 1 & 0 \\ -1 & 0 \end{bmatrix}$$

Esquemáticamente, la forma de cálculo de un operador gradiente, es:

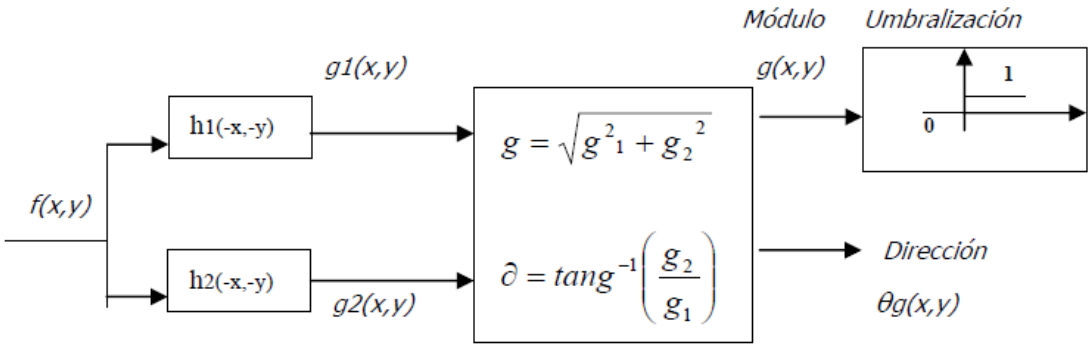


Figura 2.38: Esquema de cálculo del operador gradiente

Habitualmente se representa únicamente el módulo del gradiente.

Las transformaciones basadas en diferencias de brillos entre vecinos son muy sensibles al ruido, por lo que para reducir este efecto se han propuesto diferentes ventanas. Se presentan a continuación algunas de las más populares.

Métodos basados en la primera derivada: Operador Gradiente. Constituyen los métodos con más proliferación dentro de la Comunidad del Análisis de Imagen y la Visión Computacional. Se fundamentan en que un borde existe sí hay una discontinuidad en la función de intensidad de la imagen, es decir, si la derivada de los valores de intensidad de la imagen es un máximo.

- **Operador de Roberts**: $H_1 = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}; H_2 = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$

Tiene buena respuesta en bordes horizontales y verticales, además de sencillez computacional, provocando rapidez de cálculo en el proceso.

A igual que en el caso genérico utiliza muy pocos píxeles para aproximar el gradiente haciéndolo muy sensible al ruido, crea una mala respuesta en bordes diagonales sin dar información acerca de la orientación del borde.

- **Operador de Prewitt**: $H_1 = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}; H_2 = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$

- **Operador de Sobel**: Igual que el anterior, con la excepción que le da más peso a los píxeles más cercanos al centro de las máscaras.

$$H_1 = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}; H_2 = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

- **Operador Isotrópico**: Idéntico en la forma a los dos anteriores, exceptuando los valores cercanos al centro de las máscaras donde aparece un valor intermedio. Este operador es invariante en la rotación.

$$H_1 = \begin{bmatrix} -1 & 0 & 1 \\ -\sqrt{2} & 0 & \sqrt{2} \\ -1 & 0 & 1 \end{bmatrix}; H_2 = \begin{bmatrix} -1 & -\sqrt{2} & -1 \\ 0 & 0 & 0 \\ 1 & \sqrt{2} & 1 \end{bmatrix}$$

Estos operadores mantienen aspectos comunes a diferencia del operador Roberts, que es menos sensible al ruido de la imagen y alberga una mejor respuesta en bordes horizontales y verticales, a diferencia de bordes diagonales. En cambio requiere un mayor gasto computacional y de tiempo para su cálculo y admite bordes de mayor grosor al poder variar el tamaño de las máscaras. Además, la fijación por parte del usuario de los umbrales y el tamaño de la máscara afectará a la posición del borde. El gradiente presenta una excesiva dependencia con respecto a la dirección de barrido, por ello, las aristas cuyas pendientes están próximas a la dirección de barrido no se detectan fácilmente, esto provoca la pérdida de puntos relevantes y de marcado de junturas.

2.4.3.2. Operador compás

Otro tipo de operados para la extracción de bordes es considerar el caso específico de derivada para cada dirección en el espacio 2D con una resolución de 45 grados, a diferencia de los anteriores operadores cuyas direcciones de cálculo son perpendiculares, y únicamente en los ejes X e Y. Permiten calcular la variación de luminancia en una imagen en una determinada dirección con una única matriz, en lugar de los dos necesarios con los métodos de operación de gradiente generales. Se deben calcular las ocho posibles direcciones para cada punto y hallar el mayor de los resultados, que será el valor gradiente. Éste valor será idéntico al obtenido con otros métodos (Prewitt, Sobel…). Estas son las matrices de cálculo para hallar cada gradiente en una dirección:

↖Noroeste $\begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -1 \end{bmatrix}$ **↑Norte** $\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$ **↗Noreste** $\begin{bmatrix} 0 & 1 & 1 \\ -1 & 0 & 1 \\ -1 & -1 & 0 \end{bmatrix}$

←Oeste $\begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -1 \end{bmatrix}$ **→Este** $\begin{bmatrix} 0 & 1 & 1 \\ -1 & 0 & 1 \\ -1 & -1 & 0 \end{bmatrix}$

↙Suroeste $\begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -1 \end{bmatrix}$ **↓Sur** $\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$ **↘Sureste** $\begin{bmatrix} 0 & 1 & 1 \\ -1 & 0 & 1 \\ -1 & -1 & 0 \end{bmatrix}$

2.4.3.3. Operador Laplace

Hasta ahora todos los operadores gradientes dependen de la primera derivada de la función imagen para hallar variaciones de niveles de brillo en determinadas direcciones y, por lo tanto, bordes. El operador Laplaciano constituye una manera de detectar bordes independientemente de la orientación o dirección de los mismos. Se fundamentan en que cuando la imagen presenta un cambio de intensidades a lo largo de una determinada dirección, existirá un máximo en la primera derivada a lo largo de dicha dirección y un paso por cero en la segunda derivada:

$$\nabla^2 f(x,y) = \frac{\delta^2 f}{\delta x^2} + \frac{\delta^2 f}{\delta y^2} \ (3)$$

Al igual que ocurría con los gradientes, se debe desarrolla para un entorno de valores discretos, utilizando las ecuaciones en diferencias de muestras para cada eje de coordenadas:

$$\frac{\delta^2 f}{\delta x^2} = \frac{\partial G_x}{\partial x} \underset{(1)}{\rightarrow} \frac{\partial G_x}{\partial x} = \frac{\partial (f(x+1,y) - f(x,y))}{\partial x} = [f(x+1,y) - f(x,y)] - [f(x,y) - f(x-1,y)] = f(x+1,y) - 2f(x,y) + f(x-1,y) \ (4)$$

$$\frac{\delta^2 f}{\delta y^2} = \frac{\partial G_y}{\partial y} \underset{(2)}{\rightarrow} \frac{\partial G_y}{\partial y} = \frac{\partial (f(x,y+1) - f(x,y))}{\partial y} = [f(x,y+1) - f(x,y)] - [f(x,y) - f(x,y-1)] = f(x,y+1) - 2f(x,y) + f(x,y-1) \ (5)$$

Sumando las componentes de (4) y (5) como se indica en (3) se obtiene el Laplaciano: $\nabla^2 f(x,y) = [f(x+1,y) - 2f(x,y) + f(x-1,y)] + [f(x,y+1) - 2f(x,y) + f(x,y-1)] = f(x+1,y) f(x-1,y) - 4f(x,y) + f(x,y+1) + f(x,y-1)$

$$L = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Esto indica que al aplicarla sobre una imagen, la detección de bordes se realiza en no necesariamente cambios muy bruscos en cualquiera de las direcciones, provocando una mayor sensibilidad frente al ruido, por lo que se modifica para que incluya un pequeño efecto filtrado paso bajo que atenúe este efecto. Otros ejemplos destacables de matrices Laplacianas más utilizadas:

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}; \begin{bmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{bmatrix}$$

Estas y otras matices derivadas de la anterior son muy utilizadas para el análisis de imágenes y extracción de formas en el tratamiento de imagen, ya que la imagen resultante es de menor tamaño y contiene menos información (e incluso puede binarizarse) pero distingue los bordes de la misma, y por lo tanto son de más fácil manejo y procesado más rápido en caso de querer trabajar únicamente con los objetos y contornos que contiene.

2.4.3.4. Perfilado

Esta técnica trata de recuperar imágenes afectadas por un cierto emborronamiento o difuminado producido por imperfecciones de la lente, o bien para fotografías a las que se quieren dotar de un cierto efecto artístico, creando un efecto de realzado de bordes mediante la suma a la imagen original de otra modificada y filtrada donde se eliminan determinadas frecuencias.

Para ello, se emplea la suma de máscaras ponderadas que dan el peso deseado a cada una de ellas (imagen original y filtrada) para obtener distintos resultados. Esto, traducido a máscaras matriciales se interpreta como la suma de una máscara de perfilado tipo Laplaciano y otra máscara unidad invariante de la imagen:

$$a\begin{bmatrix}0 & -1 & 0\\ -1 & 4 & -1\\ 0 & -1 & 0\end{bmatrix}+b\begin{bmatrix}0 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 0\end{bmatrix}=\begin{bmatrix}0 & -a & 0\\ -a & 4a+b & -a\\ 0 & -a & 0\end{bmatrix}$$

2.4.4. Filtrado en el dominio de frecuencia

Un filtro puede entenderse como una función que modifica o transforma una señal obteniendo otra a su salida, una función de transferencia a través de la cual "ambos mundos se conectan". A continuación se detalla el mundo del dominio frecuencial, sus propiedades, para qué sirve de utilidad, ventajas, o la comunicación entre este nuevo entorno y "el mundo temporal" en el que se trabaja habitualmente. Comenzamos por explicar esta puerta entre ambos mundos que, como se comenta, es una función, y al igual que ocurría con el proceso convolutivo se necesita de su versión de cálculo concreta para señales discretas 2D. Se debe escoger una puerta entre las muchas disponibles dependiendo de la utilidad y contexto en el que se use. Para el ámbito en que se está tratando este proyecto, integrado en el procesado de imágenes, se disponen de varias opciones comúnmente utilizadas: Transformada de Fourier (funciones base seno y coseno), transformada del Coseno (función coseno), transformada Wavelet (funciones base Haar, Daubechies, etc.) ó transformadas de Karhaunen-Loeve o Análisis de componentes principales (PCA).

De éstas, las dos primeras son obviamente las más reconocibles y manejadas. Para desarrollar el proceso de transformación se utilizará la teoría formulada por el matemático francés Joseph Fourier en el s. XVIII y que tiene su mismo nombre. Esta teoría consigue definir una función temporal como aproximación de suma de señales sinusoidales (seno y coseno) de diferente frecuencia y peso constituyendo una base de funciones en el dominio de la frecuencia. Los coeficientes obtenidos marcan la relevancia o peso de la señal transformada en una determinada frecuencia, lo que permite manipular directamente componentes frecuenciales variando este peso en una paso intermedio, y realizando el proceso inverso para retomar la señal temporal una vez que se ha modificado su espectro de frecuencia. Para señales discretas bidimensionales se realiza la llamada TFD 2D, (Transformada discreta de Fourier en dos dimensiones), y considerando *f (m, n)* señal temporal discreta de tamaño *M y N*:

$$F(u,v) = \Im[f(m,n)] = \frac{1}{\sqrt{MN}} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f(m,n) e^{-j2\pi\left(\frac{um}{M}+\frac{vn}{N}\right)}; \quad \begin{matrix} u = 0,1,2,\dots,M-1 \\ v = 0,1,2,\dots,N-1 \end{matrix}$$

Obtenemos una matriz del mismo tamaño que la imagen, en realidad ésta matriz es sólo un periodo de la DFT que nos interesa. La separabilidad permite realizar dos DFTs, una por cada abscisa, y obtener los mismos resultados. Es una propiedad importante que permite utilizar algoritmos de DFT 1D más rápidos.

Cada valor de *F (u, v)* obtenido es un número complejo, no asustarse, esto únicamente indica el módulo o amplitud y la fase inicial de la sinusoide. Además, para el trabajo que ocupa nos centraremos en la representación del módulo para cada sinusoide que ayudará a detectar dónde se acumula la mayor parte de la energía.

Evidentemente, para nosotros no resultaría útil quedarnos con la representación espectral de la imagen modificada y no poder "volver" de nuevo al mundo temporal. La transformada inversa de Fourier (un nombre muy conveniente) nos proporciona este proceso:

$$f(m,n) = \Im^{-1}[F(u,v)] = \frac{1}{\sqrt{MN}} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} F(u,v) e^{j2\pi\left(\frac{um}{M}+\frac{vn}{N}\right)}; \quad \begin{matrix} m = 0,1,2,\dots,M-1 \\ n = 0,1,2,\dots,N-1 \end{matrix}$$

2.4.4.1. Representación gráfica del espectro de frecuencia

Es importante señalar la distribución en un espacio 2D de las señales sinusoidales centrándonos en la frecuencia que las gobierna; aquellas de baja frecuencia se encontrarán en las esquinas de la imagen, posiciones cercanas a F(u=0,v=0), F(u=0,v=N), F(u=M,v=0) y F(u=M,v=N). A medida que nos alejamos de estas posiciones de la imagen en el dominio de la frecuencia nos acercamos a frecuencias mayores.

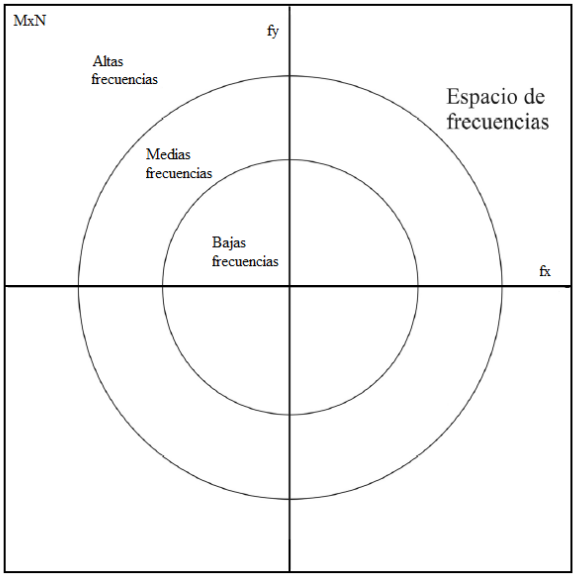


Figura 2.39: Ubicación en la imagen del espacio de frecuencias

En imágenes convencionales la mayoría de energía se centra en las bajas frecuencias, por lo que en muchas ocasiones la representación matricial de los módulos se modifica, invirtiendo simétricamente las posiciones y situando todas las componentes de baja frecuencia en la parte central de la matriz, desplazando a posiciones más alejadas del centro componentes de frecuencias media y alta (*fftshift* en MATLAB).

Se produce otro efecto negativo en la representación debido al mismo motivo de concentración de energía en baja frecuencia, la falta de resolución en niveles bajos. El brillo máximo está ubicado en el centro de la imagen, pero en el resto de espectro la imagen queda prácticamente oscura, ya que la diferencia de energía es considerable entre ambas zonas, por lo que se debe expandir los niveles de brillo en baja frecuencia para obtener un mayor detalle de la zona donde se encuentra la mayor parte de la información, comprimiendo los niveles de brillo de las frecuencias altas. Habitualmente se utiliza una expresión exponencial dependiente de un valor "β" que puede variar en función del grado de expansión que se desee, esta función de trasferencia es la explicada en el punto 2.3.1 función exponencial que sirve perfectamente para resolver los problemas de resolución de brillos en baja frecuencia que se plantean:

$$v = \frac{255\left[\log\left(\frac{u}{255}\left(e^{\beta}-1\right)+1\right)\right]}{\beta}$$

2.4.4.2. Propiedades de la transformada de Fourier:

- **Separabilidad**: Ya mencionado, es posible calcula la DFT de una imagen bidimensional a través de dos DFTs unidimensionales para caja eje de ordenadas aplicadas sucesivamente. Aplica también a la transformada inversa.
- **Linealidad:** La Transformada de Fourier para dos imágenes sean:

$$\Im[x_1(\mathrm{m},\mathrm{n})] = X_1(\mathrm{u},\mathrm{v})$$
$$\Im[x_2(\mathrm{m},\mathrm{n})] = X_2(\mathrm{u},\mathrm{v})$$

Al Crear una nueva imagen: $x(m,n) = ax_1(m,n) + bx_2(m,n)$
Su transformada cumple que: $X(u,v) = aX_1(u,v) + bX_2(u,v)$

- **Traslación**: Una traslación temporal, supone un cambio de fase en el espectro de frecuencia:

$$x_2(u,v) = x_1[(m-m_0),(n-n_0)]$$
$$X_2(u,v) = X_1(u,v)e^{j2\pi m_0}e^{j2\pi n_0}$$

- **Periodicidad**: Como se ha comentado anteriormente, la Transformada de Fourier de una secuencia x(m,n) de tamaño MxN es una función periódica:

$$F(u,v) = F(u+M,v) = F(u,v+N) = F(u+M,v+N)$$

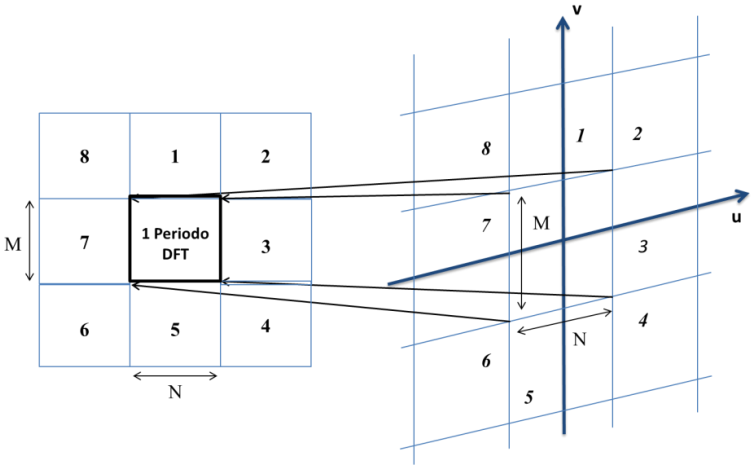
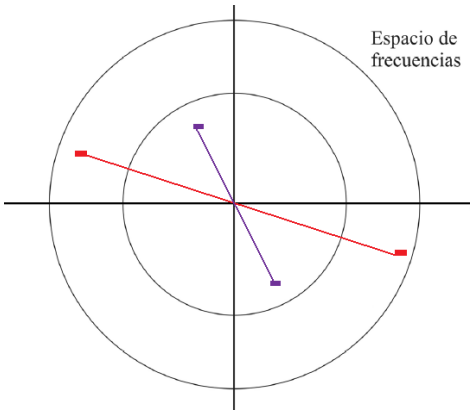


Figura 2.40: Periodicidad de la DFT

Sólo se considera un periodo, pero es importante tener en cuenta que la DTF se repite en el espacio 2D, sobre todo a la hora de realizar una convolución entre dos matrices en el dominio de la frecuencia.

- **Simetría conjugada**: Esta propiedad puede resultar muy útil a la hora de calcular una DFT y poder ahorrar el tener que hallar cantidad de valores que se pueden obtener a partir de otros ya calculados, además de comprender la distribución de las frecuencias en el espectro obtenido:



$$X(u,v) = X^*(-u,-v)$$

Figura 2.41: Simetría conjugada de la DFT

- **Rotación**: Si una matriz de valores imagen f(m,n) se rota un ángulo θ, su transformada rotará en el mismo valor.

- **Convolución**: La convolución entre dos matrices en el dominio temporal equivale a una multiplicación de elemento a elemento entre ambas matrices en dominio de la frecuencia:

$$\Im[x_1(\mathrm{m},\mathrm{n})] = X_1(\mathrm{u},\mathrm{v})$$
$$\Im[x_2(\mathrm{m},\mathrm{n})] = X_2(\mathrm{u},\mathrm{v})$$

$$x_1(m,n) \otimes x_2(m,n) \rightarrow X_1(u,v).^* X_2(u,v)$$

- **Producto**: Recíprocamente, el producto elemento a elemento entre dos matrices equivale a una convolución circular (advertíamos de que el espectro no es limitado en el desarrollo de la propiedad **Periodicidad**) entre ambas transformadas de Fourier:

$$\Im[x_1(\mathrm{m},\mathrm{n})] = X_1(\mathrm{u},\mathrm{v})$$
$$\Im[x_2(\mathrm{m},\mathrm{n})] = X_2(\mathrm{u},\mathrm{v})$$

$$x_1(m,n).^* x_2(m,n) \rightarrow X_1(u,v) \otimes X_2(u,v)$$

- **Escalado**:

$$\Im[f(m,n)] = F(u,v)$$

$$\Im[f(am,bn)] = \frac{1}{|ab|} F\left(u/a, v/b\right)$$

- **Interpolación de ceros**: realizar una interpolación de ceros entre muestras redimensionando la imagen por un factor X, supone repetir el espectro X veces
- **Añadir ceros al final**: equivale a introducir valores nuevos en el espectro de frecuencia
- **Valor medio**: El valor medio de brillos de una imagen equivale al valor F(0,0):

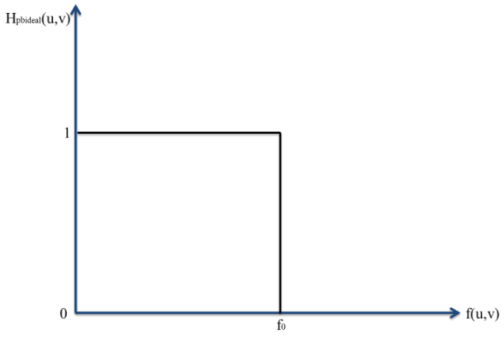
$$Valor\ medio\ de\ f(m,n) = \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f(m,n) = \frac{1}{MN} F(0,0)$$

Una vez vistas las principales propiedades de la DFT debemos definir filtros invariantes o funciones de transferencia H que actúen en el espacio de frecuencias: $Y = H \cdot X$, o lo que es lo mismo: $y = h * x$.
Esta relación da a entender que cualquier operación lineal se puede aplicar sobre la imagen en el espacio de frecuencias, generando una selección deseada que se mantengan invariante y limitando otra, produciendo en cualquier caso que el contenido de frecuencias de la imagen salida sea menor a la d entrada. Este propósito se consigue de forma generalmente más sencilla en el espacio de frecuencias que en el temporal analíticamente hablando, ya que es más fácil de imaginarse un diseño de filtro para este propósito en un espacio donde se conoce la ubicación de cada tipo de frecuencias, y poder definir un filtro adecuado al propósito. A continuación veremos la implementación de varias de estas funciones H y sus formas asociadas para el filtrado de frecuencias más usado en el dominio de las frecuencias

2.4.4.3. Filtro Paso Bajo

Para este tipo de filtro nos centramos en un filtro que permita el paso de las frecuencias con un margen determinado marcado por la frecuencia de corte o f_0 a partir de la cual el atenuamiento se producirá para el resto de frecuencias superiores. En este punto entra en juego otro concepto de atenuación o comportamiento del filtro para estas frecuencias cercanas a f0, ya que no es igual limitar completamente el margen de frecuencias de forma directa o ideal, o bien definir otros tipos de filtros más reales que sigan una función de trasferencia de un determinado orden o pendiente de atenuamiento.

- **Filtro ideal paso bajo**: Comenzamos con el más sencillo e intuitivo es de concebir, eliminando todas aquellas frecuencias a partir de la f0, por lo que su definición en el espacio de frecuencias sería:



$$H_{pb_{ideal}}(u,v)\begin{cases}1\ si\ f \le f_0\\0\ si\ f > f_0\end{cases}$$

Figura 2.42: Función de transferencia filtro ideal paso bajo

Este tipo de filtro no se puede realizar a base de componentes electrónicos como implementación analógica ya que se necesitarían condensadores y resistencias de valor infinito, aunque puede ser simulado en un ordenador.

- **Filtro Butterworth paso bajo**: Corresponde a una función de transferencia con una atenuación transitoria, sin establecer un claro corte entre las frecuencias permitidas y las filtradas. Gobernada por una función exponencial del tipo $1/(1+x^n)$, correspondiendo más con la realidad de implementación a nivel electrónico.

Los filtros de Butterworth de orden n y con una frecuencia de corte f_o vienen dados por la siguiente función de transferencia:

$$H_{pb_{butterworth}}(u,v) = \frac{1}{1+\left[\sqrt{u^2+v^2}\Big/f_0\right]^{2n}} \quad (4.0)$$

Se caracteriza por que la atenuación en f_0 es del 50%, $H(f_0) = {}^1\!/_2$. A medida que aumente el orden del filtro la banda de transición de reduce, acercándose al filtro idea cuando $n \to \infty$.

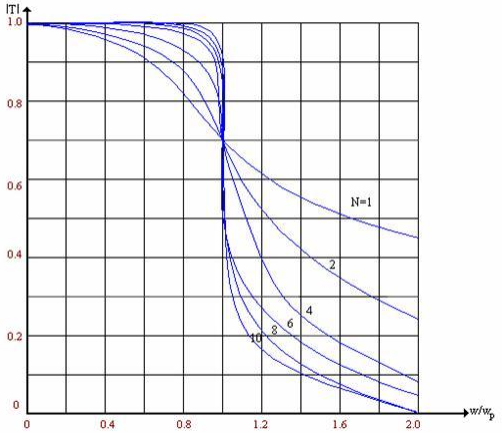


Figura 2.43: Función de transferencia filtro Butterworth paso bajo para diferentes N

- **Filtro Gaussiano paso bajo**: Antes de finalizar el primer apartado de filtros me gustaría detallar este que estará incluido en las opciones de filtrado en frecuencia de Forevid, el **filtro basado en distribución normal o Gaussiana**. Este tipo de función de transferencia está disponible para todas las variantes de filtrado: paso bajo, paso alto, paso banda y tipo ranura o notch. Simula una distribución gaussiana bivariante cuyo

valor máximo 1 aparece en el centro de la campana y luego su valor comienza a disminuir según la función Gaussiana cuya dispersión (o apertura) está dada por la constante σ, que en nuestro caso vendrá definido por la frecuencia de corte f_0. El resultado final en el espectro de frecuencia será un conjunto de valores entre 0 y 1. La ecuación de filtrado paso bajo para calcularla es:

$$H_{pb_{Gaussiano}}(u,v) = e^{-\frac{D^2(u,v)}{2f_0^2}}$$

$$con\ D(u,v) = \sqrt{\left(u - {}^{M}/_{2}\right)^2 + \left(v - {}^{N}/_{2}\right)^2}$$

Su porcentaje de atenuamiento es del 60,7% del máximo para $H(f_0)$.

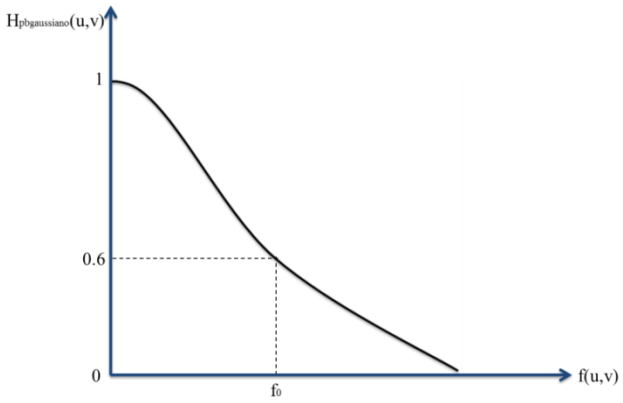
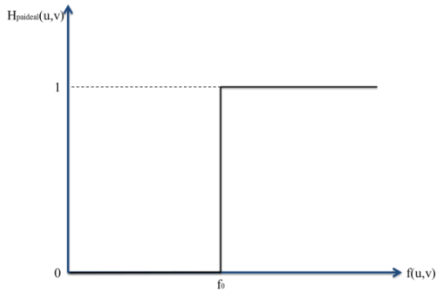


Figura 2.44: Función de transferencia filtro Gaussiano paso bajo

2.4.4.4. Filtro Paso Alto

De manera contraria a los filtros que acabamos de ver, los filtros pasos altos atenúan las bajas frecuencias realzando los bordes y detalles de relieve en la imagen creando un efecto de mayor relieve y evidenciar discontinuidades en áreas homogéneas. Como sucedía en el apartado anterior se van a detallar distintos tipos de filtrados para estas frecuencias. Como complementario del filtro paso bajo se puede crear un filtro paso alto en el dominio de la frecuencia (de manera visual es fácilmente entendible) como diferencia entre un filtro paso todo invariante del espectro de frecuencia, y un paso bajo, como se verá más adelante para todos los tipos de filtros.

- **Filtro ideal paso alto**: Al igual que para el filtro paso bajo tiene un cambio brusco en la frecuencia de corte y con una función de trasferencia complementario al detallado en el filtro idea paso bajo en el espacio de frecuencias:



$$H_{pa_{ideal}}(u,v) = 1 - H_{pb_{ideal}}(u,v) = \begin{cases} 0\ si\ f \leq f_0 \\ 1\ si\ f > f_0 \end{cases}$$

Figura 2.45: Función de transferencia filtro ideal paso alto

Se logra poner a cero los módulos de los coeficientes de Fourier relativos a las bajas frecuencias, dejando sin modificar los relativos a las altas frecuencias

- **Filtro Butterworth paso alto**: Aproximándonos a una función de transferencia más real los filtros Butterworth varían su pendiente en función del orden de la función. El coeficiente entre la frecuencia de corte y la de estudio se debe invertir respecto a la ecuación 4.0 que se define para el filtro paso bajo:

$$H_{pa_{Butterworth}}(u,v) = 1 - H_{pb_{Butterworth}}(u,v) = 1 - \frac{1}{1 + \left[{\sqrt{u^2+v^2}}/{f_0} \right]^{2n}}$$

$$= \frac{1}{1 + \left[{f_0}/{\sqrt{u^2+v^2}} \right]^{2n}}$$

Por su puesto se mantienen las mismas discrepancias y características entre la función ideal y la de Butterworth que indican la aproximación de una a otra cuando n tiende a infinito, así como su implementación en circuito electrónico.

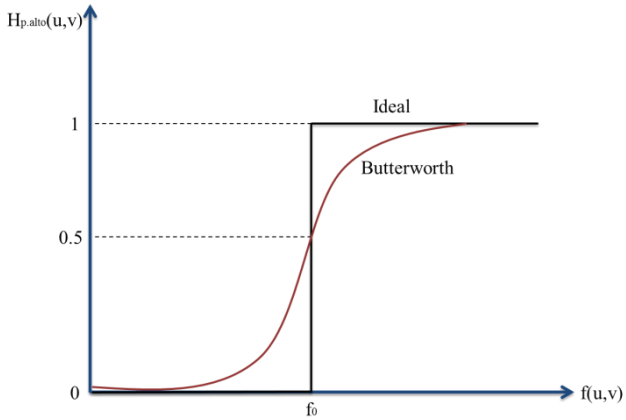


Figura 2.46: Comparativa filtro ideal-butterworth de tipo paso alto

- **Filtro Gaussiano paso alto**: Determinada la dinámica de operación en los anteriores ejemplos de filtros pasos altos hallados, obtenemos mediante la misma metodología su correspondiente gaussiano:

$$H_{pa_{Gaussiano}}(u,v) = 1 - H_{pb_{Gaussiano}}(u,v) = 1 - e^{-\frac{D^2(u,v)}{2f_0^2}}$$

$$con\ D(u,v) = \sqrt{\left(u - {M}/{2}\right)^2 + \left(v - {N}/{2}\right)^2}$$

2.4.4.5. Filtro Paso-Banda

Su composición no es más que una combinación de un filtro paso bajo y otro paso alto en el dominio de frecuencias, siendo $f_{pb}>f_{pa}$.

- **Filtro paso banda ideal**: Al sustraer de un filtro paso todo el efecto de un filtro paso bajo con una frecuencia de corte f_b, y otro filtro paso alto con $f_0=f_a$, obtenemos una

banda de frecuencias que se mantienen inalteradas, siempre que se cumpla la condición anteriormente dicha donde la frecuencia f_b es mayor que f_a.

$$H_{pbanda_{ideal}}(u,v) = 1 - H_{pb_{ideal}}(u,v) - H_{pa_{ideal}}(u,v) = \begin{cases} 0, & si\ f < f_{pa} \\ 1, si\ f_{pa} < f < f_{pb} \\ 0, & si\ f > f_{pb} \end{cases}$$

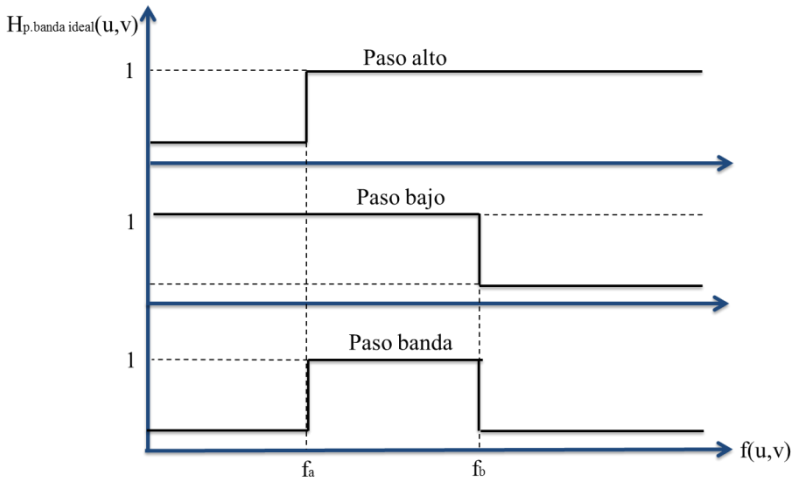


Figura 2.47: Operaciones gráficas para filtro paso-banda ideal

- **Filtro paso banda Butterworth**:
  $H_{pbanda_{Buterworth}}(u,v) = 1 - H_{pb_{Buterworth}}(u,v) - H_{pa_{Buterworth}}(u,v)$

Aparte de los parámetros de frecuencias de corte aparecen otros como:

Frecuencia central: $f_{central} = \sqrt{f_a f_b}$

Ancho de banda: $BW = f_a - f_b$

Factor de calidad: $Q = \frac{f_{central}}{BW}$. Cuanto mayor sea este factor, más estrecho será el ancho de banda de frecuencia inalterada y por lo tanto más selectiva.

2.4.4.6. Filtro Ranura (Notch)

O filtro banda eliminada, realiza la función complementaria al paso banda, poniendo a cero únicamente los coeficientes encerrados en una banda del espectro de frecuencia. Se puede expresar como diferencia de un filtro paso todo y un paso banda anteriormente descrito y, por lo tanto, también puede definirse a partir de filtros paso-bajo y paso-alto.

- **Filtro Notch ideal**:

$$H_{Notch_{ideal}}(u,v) = 1 - H_{pbanda_{ideal}}(u,v) = \begin{cases} 1, & si\ f < f_{pa} \\ 0, si\ f_{pa} < f < f_{pb} \\ 1, & si\ f > f_{pb} \end{cases}$$

Como vemos, este filtro procura eliminar una determinada franja de frecuencia que es provocada por ruido frecuencial que se inserta en la imagen de forma indeseada.

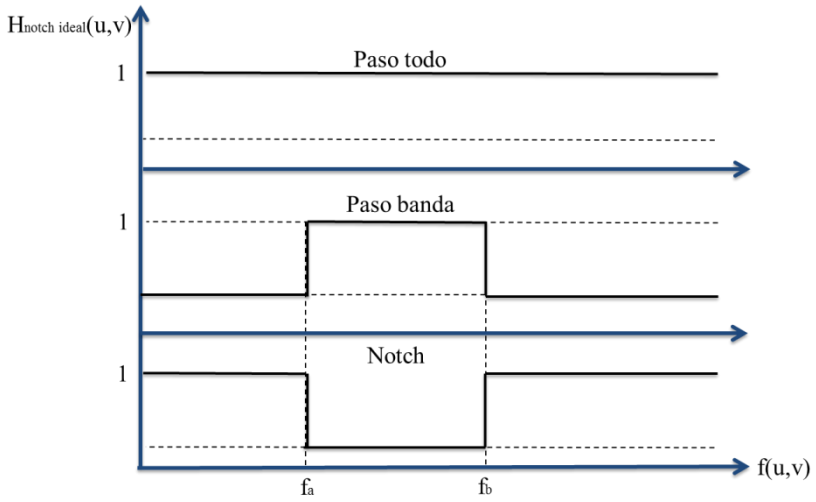


Figura 2.48: Operaciones gráficas para filtro notch ideal

- **Filtro Notch Butterworth**:

$$H_{pNotch_{Buterworth}}(u,v) = 1 - H_{pBanda_{Butterworth}}(u,v)$$

Surgen los parámetros anteriormente definidos para un filtro pasa banda Butterworth.

2.4.4.7. Filtrado en frecuencia

Conocidos los principales tipos de filtrado que se pueden realizar en una imagen, es el momento de llevarlo a cabo. En este apartado vamos a introducirnos en la metodología manejada desde el dominio de frecuencia, utilizando una de las propiedades de la DFT descrita en 2.3.2: la convolución. Esta propiedad permite realizar un filtrado mediante la multiplicación elemento a elemento de dos matrices del mismo tamaño. Como se ha explicado, al realizar la DFT sobre una imagen se obtienen los coeficientes de la transformada asociados a una determinada frecuencia, compuesto módulo y fase, sobre los que es posible interpretar y modificar más intuitivamente la composición de frecuencias de la imagen. En esta representación que utilizamos para el filtrado es muy importante tener en cuenta:

1. La transformada de Fourier de una imagen no se limita a otra matriz del mismo tamaño, sino que es infinita y periódica.

2. Cada valor es un número complejo compuesto por módulo y fase, aunque posteriormente nos centremos únicamente en el valor real para su representación determinando un nivel de brillo para cada coeficiente.

3. Ubicación de las frecuencias en la matriz DFT. Las bajas frecuencias se sitúan en las esquinas, a medida que se alejan de estas posiciones los coeficientes hacen referencia a frecuencias más altas. Al modificar la ubicación (*fftsifth* en MATLAB) de las frecuencias, las bajas quedan ubicadas en el centro de la matriz, donde se concentra la mayor parte de la energía.

El esquema a seguir por lo tanto constará de: una transformación de la imagen de entrada al dominio espectral, obteniendo una matriz de su mismo tamaño que representará la variación de brillos, a continuación se aplicará una máscara de filtrado

compuesta por valores comprendidos entre unos y ceros, realizando una multiplicación de elemento a elemento entre ambas matrices. Obtenida una matriz resultante de coeficientes de Fourier del mismo tamaño que las anteriores, se realizará la conversión para hallar la imagen en el espacio temporal a través de la transformada inversa de Fourier (IDFT), que requiere tanto del módulo como de la fase de cada coeficiente. A continuación se representa un sencillo esquema de los pasos recorridos para el proceso:

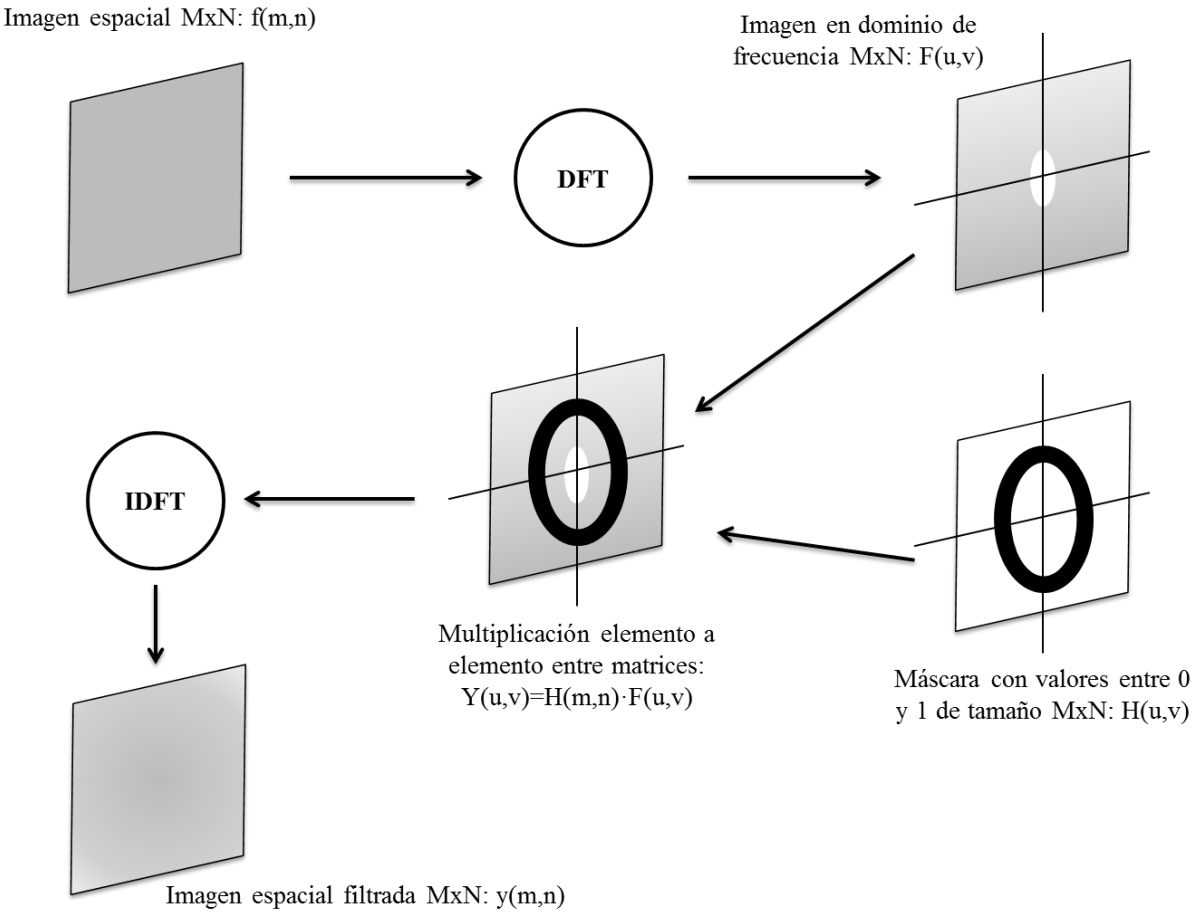


Figura 2.49: Filtrado en frecuencia mediante aplicativo de máscara matricial

Al realizar una DFT bidimensional, lo que supone que los ejes de la matriz DFT representan las frecuencias horizontales y verticales en cada eje x e y respectivamente. En el centro de la matriz se cruzan ambos ejes, referencia desde donde se toma la simetría de valores y se sitúan las frecuencias más bajas que reúnen la mayoría de la energía de la imagen.

Esta forma es la más visual e intuitiva ya que se puede visualizar rápidamente la composición espectral de una imagen y, por lo tanto, actuar en consecuencia para obtener los resultados o eliminar un tipo de ruido periódico visualmente muy apreciable en la DFT de una imagen, ya que se concentra la energía en una zona concreta. La definición de la máscara conforma la principal elección ya que define el tipo de filtro a aplicar, la inclinación de la pendiente de atenuación, el ancho de banda o frecuencia central, por lo que existen multitud de máscaras predefinidas que dependen de estos parámetros y que se deben adaptar al tamaño de nuestra imagen.

2.5. DCT: Transformada discreta del coseno

Su capacidad de compactación de energía hace que sea una herramienta muy utilizada en la compactación de imágenes. Se trata de, al igual que hacíamos con la DFT, aplicar una función sobre la imagen temporal que permite la transformación de la imagen a funciones coseno de distintas frecuencias y coeficientes que posteriormente pueden ser cuantificados sin producir pérdidas de eficiencia de compresión, disminuyendo el ancho de banda en caso de que sea necesario transportarla a través de un medio. Esto aprovecha la respuesta que tiene el ojo humano frente a las frecuencias que componen una señal visual, separándolas y descomponiéndolas para permitirnos posteriormente eliminar o ignorar aquellas que son imperceptibles para nosotros, en general altas frecuencias que quedan desplazadas en la zona inferior-derecha de la matriz de coeficientes tras realizar una DCT a una señal bidimensional como puede ser una imagen.

Es una transformada ortogonal, lo que permite aplicarla sobre matrices o bien con la transformada bidimensional directamente, o bien con dos DCT unidimensionales, una para cada eje, posibilitando la aplicación de algoritmos implementados para una dimensión a imágenes facilitando el cálculo al hardware que ejecuta el proceso.

Habitualmente se aplica sobre bloques de 8x8 o 16x16 sobre la imagen para poder segmentar con mayor precisión cada parte de la imagen reduciendo los coeficientes en mayor medida para zonas de bajo relieve y otorgar una mayor precisión en zonas de detalle. Esto puede provocar efecto bloque en casos en los que el valor DC o de continua sea muy dispar entre bloques adyacentes.

La función bidimensional matemática que define la transformación discreta del coseno es:

$$C(k,l) = \alpha(k)\alpha(l)\sum_{m=0}^{N-1}\sum_{n=0}^{N-1} F(m,n)\cos\left[\frac{\pi(2m+1)k}{2N}\right]\cos\left[\frac{\pi(2n+1)k}{2N}\right],$$

$$con\ 0 \le k,l \le N-1, y\ \alpha(k),\alpha(l) = \begin{cases} \sqrt{1/N}, & k,l = 0 \\ \sqrt{2/N}, & 1 \le k,l \le N-1 \end{cases}$$

Al igual que sucedía con la transformada de Fourier, es necesario una transformada inversa que nos proporcione la imagen en el dominio espacial a través de los coeficientes ponderados de funciones base cosinusoidales, la IDCT:

$$F(m,n) = \sum_{k=0}^{N-1}\sum_{l=0}^{N-1} \alpha(k)\alpha(l)C(k,l)\cos\left[\frac{\pi(2m+1)k}{2N}\right]\cos\left[\frac{\pi(2n+1)k}{2N}\right],$$

$$con\ 0 \le m,n \le N-1$$

La siguiente gráfica puede ayudar a comprender de forma representativa el significado de cada uno de los coeficientes de una DCT de 8x8:

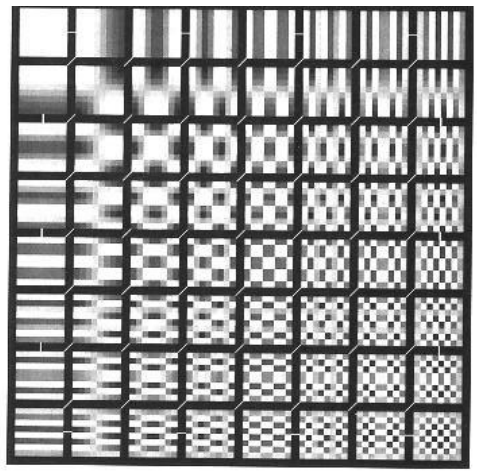


Figura 2.50: Representación de las frecuencias en la DCT para cada coeficiente

Podemos observar que las imágenes orientadas horizontalmente representan las frecuencias horizontales, y lo propio con las verticales. El cuadro superior izquierdo representa el valor de componente continua o frecuencia 0.

2.6. Redimensionado de imágenes

La mayoría de las transformaciones geométricas dan lugar a píxeles que no coinciden con la posición de los de la imagen original debido a la discretización de valores en la imagen y el muestreo sobre ella en el proceso de digitalización o captura. El problema que surge es si se debe asignar el nivel adecuado de gris existente de la imagen original o interpolar dicho brillo para esa "nueva" posición a partir de los píxeles vecinos. Existen varias alternativas que como se intuye darán lugar a peores o mejores resultados pagando el coste de un mayor coste computacional.

A parte del redimensionado de imágenes, existen otras trasformaciones anteriormente expuestas que requieren de este tipo de escalados como las geométricas, que incluyen desplazamientos, rotaciones, giros, cambios de perspectiva, simetrías o inclinaciones. Debemos tener claro que los métodos de interpolación se utilizan en aquellas circunstancias donde se quiere recuperar información perdida por compresiones y métodos de transmisión de imágenes, o directamente se desconoce.

2.6.1. Interpolación lineal

También conocido de vecino más próximo, consiste en asignar el valor de brillo a cualquier posición del espacio escogiendo el nivel de pixel más cercano de la imagen origen geométricamente. Es necesario por tanto calcular las distancias entre el centro del píxel de la imagen corregida y el de los centros de los cuatro píxeles más cercanos en la imagen transformada, tomando aquel que proporcione la menor distancia. En el siguiente ejemplo vemos como el pixel P tendría un valor de brillo idéntico al representado por el pixel rojo, ya que la distancia es la menor respecto a los cuatro más cercanos.

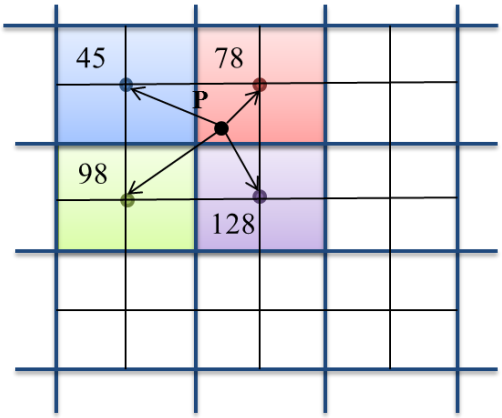


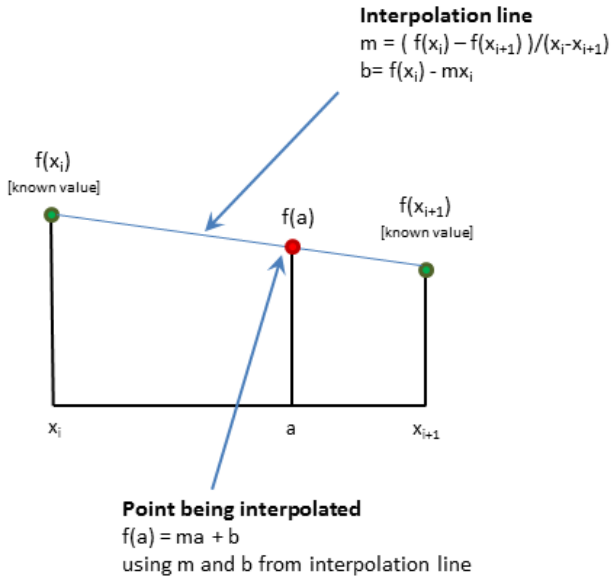
Figura 2.51: Método de interpolación lineal

Su uso se realiza principalmente por motivos de rapidez y sencillez de cálculo, e invariación de niveles de brillo de los pixeles, algo importante en corrección de imágenes con información cualitativa o temática. Se emplea principalmente en cambios de escala o zooms aplicados a una zona concreta de la imagen. Sin embargo, se produce un efecto cuadriculado y escalonado, dando la sensación de poca calidad y resolución.

2.6.2. Interpolación bilineal

Para entender este procedimiento es importante comprender la idea en un eje de dimensiones, y extrapolarlo posteriormente al otro completando el proceso para ambos ejes de direcciones x e y.

Dicho lo cual, la interpolación consiste, de forma sencilla y gráfica como se verá a continuación, ten trazar una línea recta entre dos niveles adyacentes, permitiendo que podamos interpretar cualquier valor ponderado de brillo entre ambos, para lo que que se deben considerar las distancias entre estos puntos y los valores conocidos de las posiciones entre los que se traza.



Se utilizan por tanto los cuatro niveles de brillo de los píxeles más cercanos en la imagen transformada al píxel de la corregida.

Se va a representar, ya en el espacio que ocupa, las posiciones de los centros de los píxeles, interpolando el punto P con posiciones genéricas m y n. Los puntos 1, 2, 3 y 4 se toman como referencias de la imagen original como valores más cercanos. Los puntos A y B son puntos con la misma ordenada que P, perteneciendo A a la recta que une los puntos 1 y 3, y B a la que une los puntos 2 y 3. Δx e Δy son las distancias entre los puntos AP y PB respectivamente, y se han de tomar en valor absoluto. D indica el ancho y alto de la celda, suponiendo píxeles cuadrados. Como se ha comentado anteriormente, en primer lugar se aplica una interpolación lineal en el eje vertical Y para calcular el nivel de brillo en el punto A (ND_A) y B (ND_B) con los que a posteriori servirán de referencia para ponderar el brillo en P:

$$ND_A = \left(\frac{ND_3 - ND_1}{D}\right)\Delta y + ND_1;\;\; ND_B = \left(\frac{ND_4 - ND_2}{D}\right)\Delta y + ND_2$$

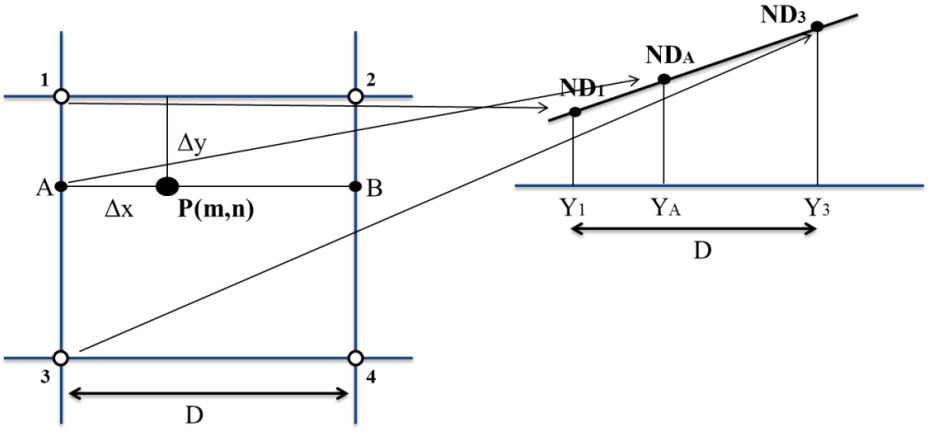


Figura 2.52: Interpolación bilineal: eje Y

Realizando lo propio para el eje X se obtiene:

$$ND_P = \left(\frac{ND_B - ND_A}{D}\right)\Delta x + ND_A$$

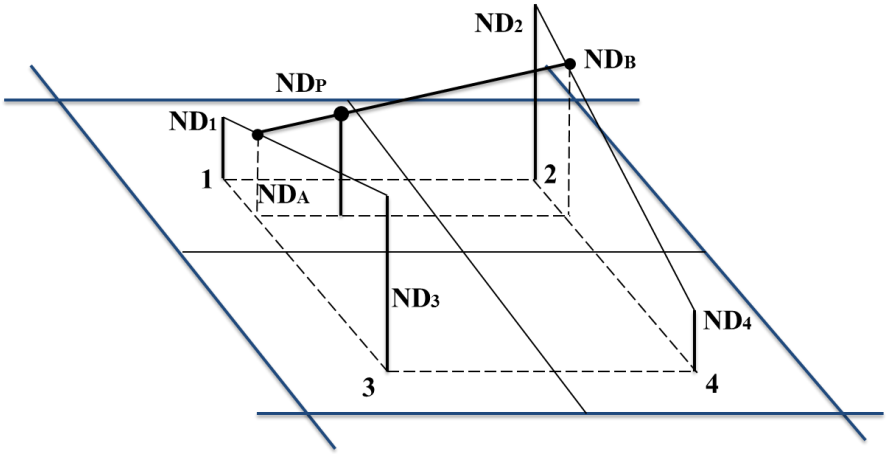


Figura 2.53: Interpolación bilineal: eje X

Este método mejora resultados en comparación con la interpolación lineal o de vecino más próximo con resultados más suavizados y precisiones espaciales mayores. Tiene los inconvenientes de suavizar también los bordes y presentar en algunas zonas efectos rectangulares.

2.6.3. Interpolación bicúbica

Fundamentado en el mismo método que el anterior, pretende dar un paso más allá y considerar los dieciséis valores de brillo más cercanos, aproximando el valor en P con una función cúbica (de tercer orden) por eje: $f(x) = c_1x^3 + c_2x^2 + c_3x + c_4$ que aproxime los valores nuevos con mayor exactitud y cree un efecto de escalado menor entre pixeles adyacentes.

Para procurar este método es necesario resolver el valor de los coeficientes, y para ello se toman 4 valores de la función (los dos valores más cercanos a cada lado del punto) que nos ayuden a resolver las incógnitas: f(m-1), f(m), f(m+1) y f(m+2), creando un sistema que determinan el peso final de cada componente.

En la siguiente figura se exhibe un modelo de vecindad generalizada de interpolación bicúbica en la que se muestran los 16 vecinos que rodean a P(m',n') como posición del pixel a interpolar, y donde la posición P(m,n) de la imagen original es el vecino más cercano. Se puede apreciar la perspectiva del procedimiento de interpolación, donde la tercera dimensión la proporciona el valor del nivel digital, y se observa los cuatro valores interpolados para cada fila. El valor digital que se asigna a la celda en la imagen corregida es el indicado mediante trazos discontinuos y de color negro:

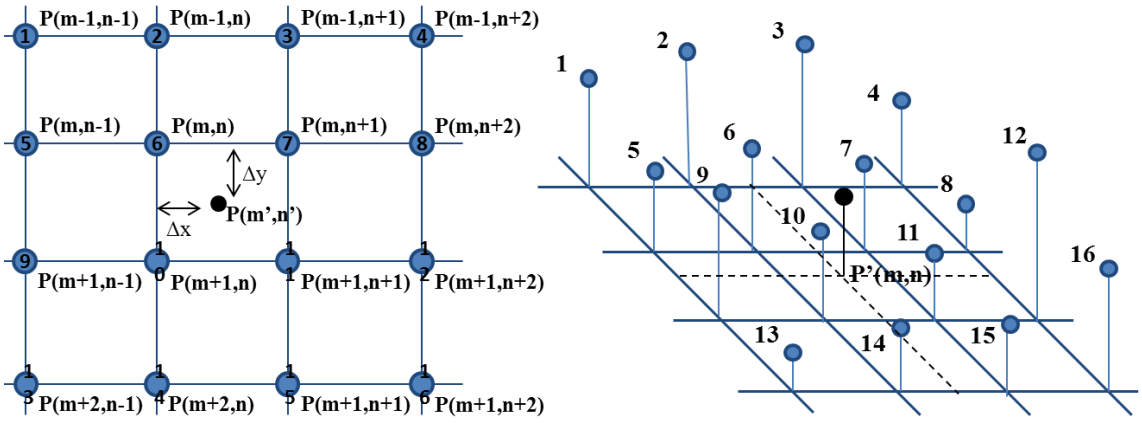


Figura 2.54: Método de interpolación bicúbica

El pixel interpolado se puede expresar en forma compacta como:

$$P(m',n') = \sum_{m=-1}^{2} \sum_{n=-1}^{2} P(m+p, n+q) \cdot H_c[m-a] \cdot H_c[n-b]$$

$$P(m',n') = \sum_{m=-1}^{2} \{P(m+p,n-1)H_c[p-\Delta y]H_c[-1-\Delta x] + P(m+p,n)H_c[p-\Delta y]H_c[-\Delta x] + P(m+p,n+1)H_c[p-\Delta y]H_c[1-\Delta x] + P(m+p,n+2)H_c[p-\Delta y]H_c[2-\Delta x]\}$$

Desarrollando el sumatorio en el eje x obtenemos los 16 términos, que expresados por filas serían:

- **Fila**$(\mathbf{m-1})$: $\{P(m-1,n-1)H_c[-1-\Delta y]H_c[-1-\Delta x] + P(m-1,n)H_c[-1-\Delta y]H_c[-\Delta x] + P(m-1,n+1)H_c[-1-\Delta y]H_c[1-\Delta x] + P(m-1,n+2)H_c[-1-\Delta y]H_c[2-\Delta x]\}$
- **Fila**$(\mathbf{m})$: $\{P(m,n-1)H_c[-\Delta y]H_c[-1-\Delta x] + P(m,n)H_c[-\Delta y]H_c[-\Delta x] + P(m,n+1)H_c[-\Delta y]H_c[1-\Delta x] + P(m,n+2)H_c[-\Delta y]H_c[2-\Delta x]\}$
- **Fila**$(\mathbf{m+1})$: $\{P(m+1,n-1)H_c[1-\Delta y]H_c[-1-\Delta x] + P(m+1,n)H_c[1-\Delta y]H_c[-\Delta x] + P(m+1,n+1)H_c[1-\Delta y]H_c[1-\Delta x] + P(m+1,n+2)H_c[1-\Delta y]H_c[2-\Delta x]\}$
- **Fila**$(\mathbf{m+2})$: $\{P(m+2,n-1)H_c[2-\Delta y]H_c[-1-\Delta x] + P(m+2,n)H_c[2-\Delta y]H_c[-\Delta x] + P(m+2,n+1)H_c[2-\Delta y]H_c[1-\Delta x] + P(m+2,n+2)H_c[2-\Delta y]H_c[2-\Delta x]\}$

El total de la suma de los cuatro números por fila es igual a P(m',n'). Sí tomamos como caso particular la interpolación cúbica $H_c(x) = h(x)$, obtenemos la expresión correspondiente a la interpolación bicúbica, cuyo núcleo más común es el siguiente:

$$h(x) = \begin{cases} 1-2|x|^2+|x|^3, & si\ 0<|x|<1 \\ 4-8|x|+5|x|^2-|x|^3, & si\ 1<|x|<2 \\ 0, & otra\ parte \end{cases}$$

El resultado de interpolación bilineal es muy utilizado y ofrece la ventaja de introducir un suavizado de la imagen menos fuerte que con la interpolación bilineal y, por lo tanto, una alta calidad debido al acercamiento en cuanto a la realidad de la imagen original, algo requerido en algunas aplicaciones donde se requiere de una alta fidelidad de la imagen recompuesta con la original, aunque el precio pagado en cómputo computacional puede ocasionar que no sea el método preferido en procesamiento de videos en favor de otros como el anterior expuesto, la interpolación bilineal del punto 2.6.2, o modificaciones derivadas de esta, ya que Avisynth dispone de múltiples tipos de interpolaciones que difieren entre ellos pero que todas se basan todos en alguna de estas tres expuestas. Además, decir que el empleo de una u otra técnica debe depender, además de la finalidad y requerimientos, el tipo de imagen o video, ya que las diferencias entre una interpolación bilineal o bicúbica se reducen a medida que el contenido frecuencial aumenta.

3. Python. Conocer el lenguaje

A mediados de los años 90 Guido van Rossum creó este lenguaje de programación cogiendo el nombre de un grupo de cómicos mundialmente conocido: los "Monty Python", homenajeando a su manera a los ingleses.

Es, por tanto, un lenguaje de programación de creación reciente que alberga las características propias de lenguaje de alto nivel de computación, pero utilizando una sintaxis sencilla que entre otras características elimina corchetes utilizando las tabulaciones de texto en su lugar, permitiendo una lectura limpia y clara cercana a un lenguaje natural y con facilidad de aprendizaje, adquiriendo un nivel notable en un plazo de periodo corto. Esto hace que sea muy adecuado para gente que se introduce por primera vez en el mundo de la programación, además de que su funcionamiento y estructura orientada a objetos, permite el manejo de una herramienta muy potente que permite la traslación de los conceptos reales al mundo digital, manejando la ejecución de una aplicación como interacción entre objetos creados.

A diferencia de los lenguajes de programación más vistos habitualmente (C, C++, Pascal o Visual Basic) es un lenguaje interpretado o de script, por lo que no es necesario compilarlo previamente a su ejecución. Dentro de las ventajas de esta característica destaca la portabilidad o flexibilidad, sin embargo su ejecución es más lenta y peligrosa, ya que alberga más posibilidades de error durante la ejecución. Ésta se produce a través de otro programa, encargado de comprender y ejecutar este código intermedio, llamado intérprete, que posibilita el hecho de no tener que traducir el código a lenguaje máquina para ir ejecutándolo a medida que se avanza por la aplicación. Detallar que, a pesar de lo dicho anteriormente, Python necesita traducir el código fuente en otro pseudocódigo (bytecode) la primera vez que se ejecuta, generando archivos *.pyc (python compiled) que son los que realmente se leerán en las próximas ejecuciones.

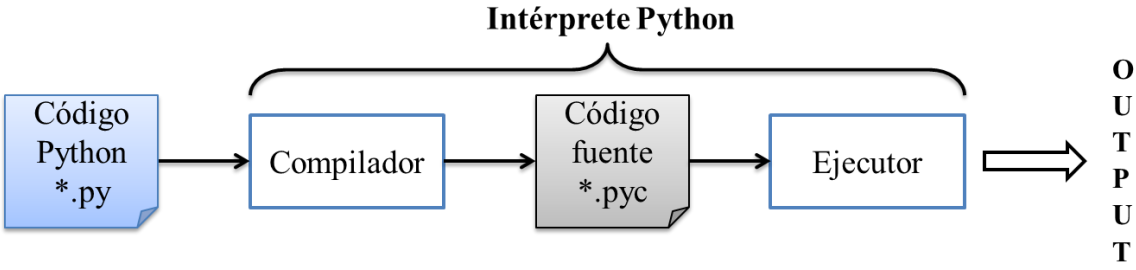


Figura 3.1: Ejecución código Python

Existen diversas implementaciones de Python dependiendo del lenguaje de programación en cual estén basados, otorgándole nombres diferentes, aunque la más comúnmente utilizada es CPython, basada en código C, debido a que su ejecución a bajo nivel proporciona mayor rapidez, está más desarrollado y depurado que las otras implementaciones. Habitualmente cuando la gente habla de este lenguaje se refiere a este tipo de implementación, a pesar de que existen muchas otras como: Jython, implementación en Java de Python, o IronPython la propia realizada en C# (.NET). La gran ventaja de utilizar este tipo de implementaciones estriba en que se pueden utilizar

todas las librerías de las que disponen en el lenguaje Python, esto permite que un conocimiento previo de las funcionalidades de otro lenguaje acerque rápidamente el desarrollo de aplicaciones en Python.

La posibilidad de agregar módulos que amplíen las funcionalidades y posibilidades es otra de las grandes ventajas, y gracias a que se trata de código abierto multitud de extensiones se han ido creando para solucionar carencias y completar funcionalidades que los desarrolladores han ido requiriendo, adaptándose a nuevas necesidades o apariciones de programas que no existían cuando se creó Python, manteniendo actualizada y útil esta herramienta. La utilización de estos módulos en una aplicación es muy sencilla, como se explicará posteriormente en detalle, ya que Forevid utiliza varios de éstos para determinadas funcionalidades: la creación de la interfaz de usuario, desarrollo de efectos gráficos, reporte de errores, creación de documentos automáticamente, etc. Además de que la gran mayoría son de código abierto y por lo tanto el usuario tiene acceso al código para su modificación y uso personal (siempre y cuando no se cierre este código una vez modificado y utilizado, lo que supondría una infracción de la licencia GNU General Public License) también son gratuitos y sin ánimo de lucro.

El intérprete al estar disponible en multitud de plataformas permite crear y ejecutar código Python tanto en sistemas UNIX, como DOS, Windows o Mac OS, además en estos primeros sistemas muchas de sus derivadas distribuciones Linux incluyen el intérprete de CPython instalado de forma predeterminada, haciendo mucho más accesible y rápido comenzar a trabajar en un entorno de trabajo integrado en el SO. La versión de Python utilizada durante el desarrollo de este proyecto ha sido la 2.7, aunque actualmente ya se dispone de la versión 3.

3.1. ¿Por qué Python?

En la introducción de este documento se comentó brevemente la potencia de este lenguaje de programación y la funcionalidad que ocupaba dentro de la aplicación Forevid. Este apartado trata de extender y detallar: el funcionamiento, la finalidad y las posibilidades de Python, además de las herramientas asociadas a este lenguaje que se han utilizado para desarrollar la UI de la aplicación.

Python no es únicamente una forma de crear la plataforma de interactuación entre la persona y el ordenador a través del ratón y teclado, sino que también debe incluir la necesidad de entenderse con las librerías que realizan el tratamiento de imagen desarrolladas en código C++, en este caso la librería ctypes incluida en la instalación de Python. Esto permite que entre ambos códigos haya un entendimiento o "traductor" que proporcione la manera sencilla de facilitar los parámetros de los filtros a través de la interfaz del usuario a dichos plugins y de forma que éstos los entiendan. Además, ya que es un lenguaje orientado a objetos, durante la creación de Forevid se establecieron clases para estructuración y acciones de los filtros (instancias, variables, interfaz…), funcionalidades como creaciones de resúmenes en formato pdf del proyecto, widget visuales para determinadas ventanas dentro de la aplicación, impresión de imágenes o

integración de reproducción de videos, e incluso el manejo de las diferentes traducciones de lenguaje o ventanas que se muestran en la aplicación Forevid. Estas son algunas de las herramientas que permite el lenguaje de programación, además de muchas otras que no se utilizan o integran en esta aplicación.

3.2. Módulos Python

Ahora vamos a entrar en detalle de aquellos módulos externos que se han utilizado en Forevid y el cometido dentro de cada uno de ellos, además de una breve descripción de sus características y posibilidades de uso fuera de la aplicación que nos ocupa. No confundir estos módulos con las librerías internas de Python, las cuales ya vienen incluidas a la hora de realizar la instalación del intérprete de Python como pueden ser: ctype, cPickle, math o subprocess. Las librerías externas que nos ocupan ahora se deben necesariamente descargar e instalar en el SO donde se ejecuta la aplicación si se desea hacer mediante el código fuente.

3.2.1. PyQt

Poco hemos comentado sobre una de las principales herramientas que se utilizan en la implementación de Forevid vinculadas al desarrollo de la UI y ejecutada sobre Python como ya se ha dicho anteriormente, PyQt[4]. Esta biblioteca es una API que incluye sus propias clases de objetos, instancias y funcionalidades gráficas que facilitan las creaciones del programador como se detallará más adelante junto con el resto de módulos incluidos en la ejecución del programa. Por supuesto son todas de código abierto, y existen multitud de páginas, aparte de la oficial detallada en la referencia, donde nos muestran todas las clases incluidas, funciones de cada una de ellas, y ejemplos creados a partir de las mismas, con posibilidad de combinarlas y adaptarlas a nuestros usos.

Incluye una aplicación que acelera el proceso de creación de UI, **QtDesigner**, donde a partir de los objetos de PyQt se va diseñando las diferentes ventanas e interpretación de las acciones del usuario sobre las mismas. La mayoría, por no decir todas, de las clases de objetos que un desarrollador necesita están ya implementadas, con las instancias y parámetros creados, una gran cantidad de creación nueva de código de programación que se ahorra, suponiendo menores costes de tiempo y dejando espacio para otros problemas más complejos que van surgiendo durante la implementación.

[4] http://www.riverbankcomputing.com/software/pyqt/

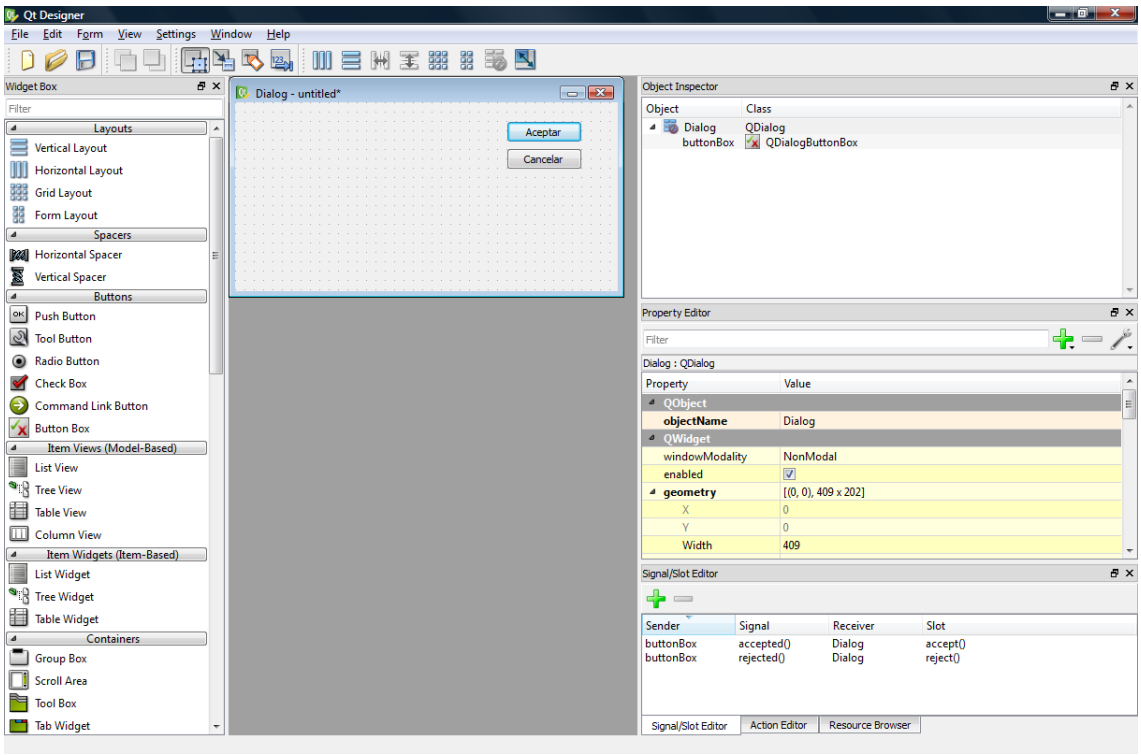


Figura 3.2: Diseñador de UI, Qt Designer

También incluye el manejo de las traducciones de textos del programa con la aplicación QtLinguist. Su funcionamiento consiste en la creación de un archivo que contenga las referencias entre una palabra original escrita en el código del programa directamente, y su correspondiente traducción, permitiendo que, una vez que el archivo de traducciones "se cargue", los textos que coincidan serán sustituidos por el almacenado en el archivo de traducciones, con extensión *.qm. Aunque se detallará más adelante, la forma en la que la aplicación Forevid trataba estas cargas de los textos entre idiomas no permitía la traducción de la aplicación en ejecución, es decir, era necesario aplicar los cambios de cambio de idioma en una variable que servía de guía para poder decidir a Forevid qué archivo de traducciones cargar al inicio de la ejecución, y sólo en este momento, por lo que se debía cerrar y volver a abrir para aplicar los cambios. Gracias a la inclusión de una instancia que "recarga" los textos de traducciones de la ventana principal, que sólo se ejecutaba al inicio, fue posible la traducción dinámica en la totalidad de las ventanas de Forevid.

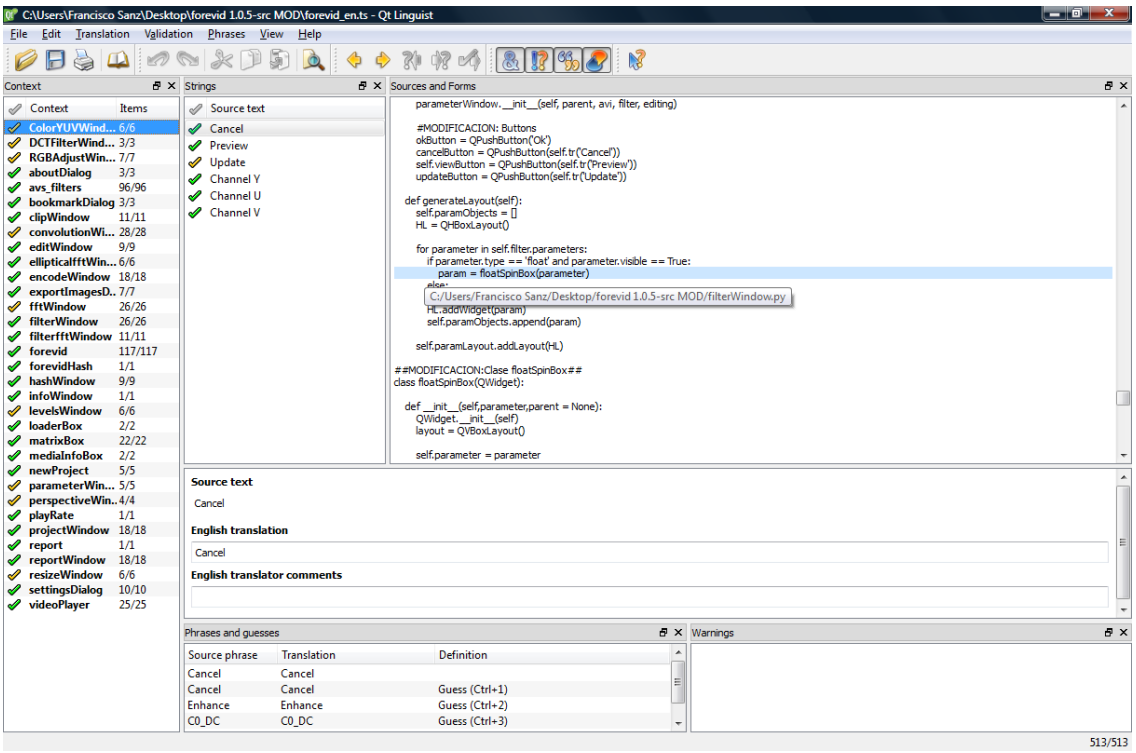


Figura 3.3: Traductor de textos, Qt Linguist

Y por último, otras de las grandes utilidades incluidas por PyQt es un espacio de referencia o documentación donde consultar a través de páginas creadas en formato *.html, todas y cada una de las clases de objetos de las que dispone, métodos de instalación y versiones de PyQt, tipos de signals y slots de los que dispone, o manuales de guía para cada una de las aplicaciones detalladas anteriormente (QtDesigner y QtLinguist), almacenadas en el equipo durante la instalación en los archivos de programa de Python, o bien consultables a través de internet[5].

Con todas estas herramientas conocidas podemos saber más sobre el funcionamiento de Forevid y los procedimientos aplicados para incluir mejoras dentro del mismo.

3.2.2. PyOpenGL[6]

O OpenGL[7] (Open Graphics Library) implementada para código Python, son unas extendidas bibliotecas que proporcionan funcionalidades y ajustes de la API gráfica de una aplicación en la que se requieren efectos 2D o 3D a partir de la creación de geometrías básicas tales como puntos, líneas, cuadrados, triángulos, círculos, etc, con diferentes texturas y colores. Con esta biblioteca de funciones también puede servir para definir la tipografía de la aplicación definiendo su tamaño, forma y formato en una gran variedad de posibilidades.

La unión entre OpenGL y Python para crear la API se crea utilizando la biblioteca estándar ctypes, siendo interoperativa con las otras librerías externas que también se

[5] http://pyqt.sourceforge.net/Docs/PyQt4/classes.html y http://pyqt.sourceforge.net/Docs/PyQt4/index.html
[6] http://pyopengl.sourceforge.net/index.html
[7] www.opengl.org/

utiliza en la aplicación para la API (wxPython, PyQt o PIL) entre otras muchas. Al igual que Python, es de código abierto y multiplataforma manteniendo la compatibilidad entre sistemas operativos.

La versión incluida en Forevid es una implementación de software que no dispone de aceleración de hardware, a diferencia de otros dispositivos que utiliza esta biblioteca de forma dedicada que requieren de mayores prestaciones a nivel de hardware (simulaciones de vuelo, representaciones de realidad virtual o científicas o plataformas para videojuegos).

3.2.3. wxPython[8]

Desarrollado por Robin Dunn como principal autor, es un conjunto de herramientas GUI para Python, implementado como módulo de extensión que incluye todas las librerías multiplataforma wxWidgets[9], las cuales fueron creadas originalmente en lenguaje de programación C++.

Al igual que Python y wxWidgets, wxPython es de código abierto, por lo que es accesible a cualquier persona y cuyo código fuente está disponible para verlo y modificarlo a su uso.

Como se ha dicho es una herramienta multiplataforma, lo que implica que el mismo programa se ejecuta en múltiples plataformas sin necesidad de modificaciones. Las plataformas sobre las que se puede soportar actualmente son: Microsoft Windows, la mayoría de los sistemas Unix, y Macintosh OS X.

3.2.4. PIL: Python Image Library

O comúnmente nombrada PIL[10], que será como se conocerá a partir de ahora en este documento, es un módulo que se puede utilizar tanto para crear, modificar y convertir archivos de imagen en una amplia variedad de formatos utilizando el lenguaje Python. Su biblioteca está diseñada para un rápido acceso a datos almacenados en formatos básicos de píxeles, y contiene algunas funciones básicas de procesamiento de imágenes: operaciones de punto, filtración espacial mediante convolución con matrices integradas, y conversiones de espacio de colores. La biblioteca también puede ajustar tamaños de imagen, rotaciones y transformaciones afines arbitrarias, además de un método de que le permite obtener algunas estadísticas de una imagen a través de su histograma, lo que se puede utilizar para la mejora del contraste automático, y para el análisis estadístico global.

En la aplicación Forevid esta biblioteca se utilizó para creación de widgets a través de elementos básicos que van variando gracias a la actualización de valores paramétricos y que permiten al usuario percibir el resultado en tiempo real de los parámetros que está introduciendo. Con esta herramienta se ha diseñado la imagen incluida en el popup de

[8] http://www.wxpython.org/
[9] http://wxwidgets.org/
[10] http://www.pythonware.com/products/pil/

filtro "*levels*" que ya tenía Forevid, y que se utilizó de ejemplo para diseñar otra imagen de ayuda en la ventana de cambio de perspectiva.

3.2.5. ReportLab[11]

Librerías opensource utilizadas como herramienta para la creación de archivos pdf con Python de forma directa basándose en comandos gráficos y sin pasos intermedios, generando informes rápidamente. Con las funciones y objetos que nos proporciona esta extensión somos capaces de dar formato al archivo diseñado incluyendo toda la información que se desee de la aplicación que estamos manejando.

En Forevid se ha recurrido de esta herramienta para elaborar resúmenes o informes acerca de un proyecto, incluyendo el nombre del video o videos que se han tratado, los filtros utilizados, parámetros en cada uno de ellos, además del autor del proyecto, fecha, descripción y otros datos del mismo.

[11] http://www.reportlab.com/opensource/

4. Avisynth

4.1. Introducción

Se trata de una organización sin ánimo de lucro mantenida y ampliada por gente anónima que se rige a partir de unas bases y parámetros que marcan la forma y tratamiento de los objetos que se pasan como parámetros, en este caso videos entendidos como conjuntos de imágenes compuestos por píxeles o muestras de brillo organizadas en un determinado formato. A partir de estas premisas, cualquiera que las conozca puede comenzar a crear plugins en lenguaje C++ creando librerías en formato *.dll que acepten un video de entrada junto con otros parámetros, modifique ciertos valores o características con un criterio o algoritmo particular y se obtenga otro salida modificado. Su metodología permite el uso de varios filtros en cascada con un lenguaje sencillo de scrips que llaman a las funciones de los plugins pasando como parámetros el video y valores enteros, decimales, cadenas de textos, etc. Esto ha hecho que AviSynth[12] sea probablemente una de las herramientas libres más extendidas utilizada para la edición de vídeo basada en scripts, incluso puede ser usada como herramienta base para otros programas de edición o conversión de espacios de colores de un vídeo.

Las posibilidades son muy amplias, ya que aparte de la edición de video, desarrolladores han creado incluso los recursos necesarios para llevar a cabo una postproducción completa y efectiva. Soporta diversos formatos y contenedores de audio-video: *.avi, *.mkv, *.mpg, *.mov… con los que puede trabajar e interpretar el formato de de video bien sea en componentes YUV o RGB en sus diferentes versiones que veremos más adelantes con más detalle. Avisynth lleva integrados multitud de filtros internos a los cuales recurrir sin necesidad de incluir una gran cantidad de archivos en el programa de edición, ya que todos se encuentran en la misma librería.

Por sí misma no tiene ninguna interfaz gráfica, lo que ha hecho proliferar multitud de aplicaciones que cubren esta carencia como Forevid, haciendo que el sistema de scripting sea totalmente trasparente y abriendo la posibilidades de uso para usuarios con conocimientos limitados en lenguaje de scripts, pero manteniendo las posibilidades y sin limitar el potencial del procesamiento de video. Para nuestro caso se ha utilizado la versión de Avisynth 2.58.

Existe una comunidad oficial[13] que se encarga de actualizar y seguir desarrollando la herramienta de forma desinteresada y compartiendo los conocimientos entre usuarios, aparte de la cantidad de ejemplos que se puede encontrar en todo tipo de foros para crear efectos únicos mediante combinaciones de filtros y los scripts que lo hacen posible utilizarlos en un solo documento de ejecución. Como ejemplo, los trabajos del autor V.C. Mohan[14] han sido de gran utilidad debido a que la mayoría se centran en transformaciones enfocadas al aprendizaje y temas tratados teóricamente en este

[12] http://avisynth.nl/index.php/Main_Page
[13] http://forum.doom9.org/forumdisplay.php?s=&forumid=33
[14] http://www.avisynth.nl/users/vcmohan/

documento. En muchos sitios web podemos hallar tutoriales con toda la información necesaria para que cualquier usuario que se plantee desarrollar nuevas librerías sea capaz de hacerlo empezando desde cero con ejemplos detallados paso a paso.

Con todo esto, vamos a ir detallando características y principales puntos a tener en cuenta a la hora de utilizar esta herramienta en edición de video.

4.2. Espacios de colores

Como todos sabemos, la representación de una imagen viene determinada por los valores de brillo en cada posición de la matriz, y para las imágenes en colores son necesarios tres valores por píxeles que determinen los valores para cada componente R, G y B como se explicó en el punto 2.1 de este documento. Sin embargo, buscando técnicas de optimización para el almacenamiento de documentación gráfica y radiotransmisión se hizo común el empleo de otros formatos o espacios de colores, también originados por los comienzos monocromáticos. Es sencillo representar y almacenar imágenes de un único plano colorimétrico, plano Y de luminancia, sin embargo la necesidad pedía realizar lo mismo con tres veces más de información para imágenes en color manteniendo la retro-compatibilidad. El espacio de colores YUV permite reducir la cantidad de información respecto del espacio RGB mediante técnicas de cálculos entre planos y transformaciones, además de conservar el plano Y inalterado. Esto se logra considerando una medida de brillo general el plano Y', hallado mediante la suma ponderada (otorgando distintos pesos a cada plano) de RGB, a partir del cual se obtienen los planos U y V calculadas como diferencias de escalas entre Y' los planos B y R respectivamente, también conocidas como componentes de crominancia:

$$W_R + W_G + W_B = 0{,}299 + 0{,}587 + 0{,}114 = 1$$

$$Y' = W_R R + W_G G + W_B B = 0{,}299R + 0{,}587G + 0{,}114B$$

$$U = 0{,}492(B - Y') = -0{,}14713R - 0{,}28886G + 0{,}436B$$

$$V = 0{,}877(R - Y') = 0{,}615R - 0{,}51499G - 0{,}10001B$$

Las matrices de transformación entre ambos espacios de colores halladas a partir de las anteriores ecuaciones son:

$$\begin{bmatrix} Y' \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0{,}299 & 0{,}587 & 0{,}114 \\ -0{,}14713 & -0{,}28886 & 0{,}436 \\ 0{,}615 & -0{,}51499 & -0{,}10001 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1.13983 \\ 1 & -0{,}39465 & -0{,}5806 \\ 1 & 2{,}03211 & 0 \end{bmatrix} \begin{bmatrix} Y' \\ U \\ V \end{bmatrix}$$

Existen otras ponderaciones dependiendo del formato de video o espacio de colores al que queramos convertir RGB, por ejemplo, estas matrices no valdrían para pasar a un formato en componentes digitales YC_bC_r, CMYK, utilizado para procesos de impresión, o YIQ, usado en sistemas de televisión NTSC (EEUU y Japón).

Perceptualmente los seres humanos somos más sensibles a la luminancia, distinguiendo con mayor facilidad la pérdida de información en los tonos entre el blanco y negro, algo importante en la búsqueda de técnicas que reduzcan la cantidad de datos por imagen sin alterar la percepción sobre el ojo humano de la imagen, ¿qué supone esto?, la respuesta es prácticamente obvia sabiendo que Y representa el plano de blancos y negros, y tanto U como V los planos colorimétricos. Esta limitación y otras del ojo humano son evaluadas a la hora de crear los formatos de video, donde también entraría en juego la frecuencia de refresco de la imagen, resoluciones o cantidad de movimiento relativo entre un cuadro y el siguiente. Por lo tanto, un parámetro recurrido es la cantidad de muestras por pixel que se utilizan y en qué cantidad para componente (Y, U y V)

En entornos de trabajo informáticos como en el que se desarrollan todos los elementos de la aplicación Forevid, no sólo debemos considerar la cantidad o número de pixeles por imagen sino también el espacio que ocupa cada valor de brillo a nivel de bits y su organización, ya que cada brillo hay que transformarlo en un valor binario que pueda manejar un ordenador con una determinada longitud y almacenarlo en un array de valores. Habitualmente se utiliza un byte por valor, es decir, 8 bits o números binarios con los que se pueden representar un total de 2^8 o 256 niveles. Como métodos de organización podemos diferenciar dos grandes grupos:

- **Packet formats**: Las muestras se almacenan en un único array de valores divididos en macropixeles, cada uno de los cuales se compone por un número determinado de muestras de YUV.
- **Planar formats**: Cada componente se almacena en un array o matriz separada, obteniendo la imagen como fusión de los tres planos.

Lo más común es utilizar el primer tipo de organización, manejando una única matriz o parámetro de valores, y conocido el formato en cuál se desarrolla manejar los datos y valores correctamente. En los diagramas que se van a representar el sufijo numérico indica la posición de muestreo de la imagen, por lo que Y_0 sería la primera muestra de luminancia hallada de la imagen comenzando desde la esquina superior izquierda. El submuestreo de la imagen tanto en vertical como horizontal determinan las muestras que servirán para el tratamiento de la imagen. Por ejemplo, el formato UYVY tiene un periodo de submuestreo horizontal de dos para las componentes colorimétricas, indicando que se toma un valor de U y otro de V por cada dos muestras a través de una línea. En cambio su período de submuestreo vertical es uno, lo que indica que las muestras de U y V se toman en cada línea de la imagen.

En los siguientes diagramas se van a representar las muestras de cada plano Y, U o V numeradas por un sufijo que indican la posición de muestreo del plano, empezando desde la esquina superior izquierda y siguiendo hacia la derecha en cada línea de imagen y de arriba abajo. Por ejemplo, V_0 indica la muestra más a la izquierda y en la primera línea de V, ya que la numeración indica la posición n+1.

A continuación se va a explicar cada uno de los tres formatos que se han utilizado en Forevid.

4.2.1. YUY2

Todas las muestras van en un único array sin ordenar por planos (packet format) intermezclando las muestras, de las cuales la luminancia es almacenada con todo detalle a diferencia de la componente cromática, que divide en dos su tamaño matricial horizontalmente ($(M/2)xN$) al submuestrear con una frecuencia mitad a la de la luminancia en este eje espacial. Este formato de submuestreo está dentro de la familia 4:2:2, al igual que UYVY descrito en el ejemplo introductorio del apartado 4.2. Si calculamos, el número de bits efectivos por pixel suman un total de 16: 8 de luminancia, 4 de crominancia U y 4 de crominancia V (1 muestra cada dos pixeles de U y V).

	Horizontal	Vertical
Y Sample Period	1	1
V Sample Period	2	1
U Sample Period	2	1

Su organización de forma gráfica se podría interpretar de la siguiente manera partiendo desde las muestras ya digitalizadas de 8 bits cada una:

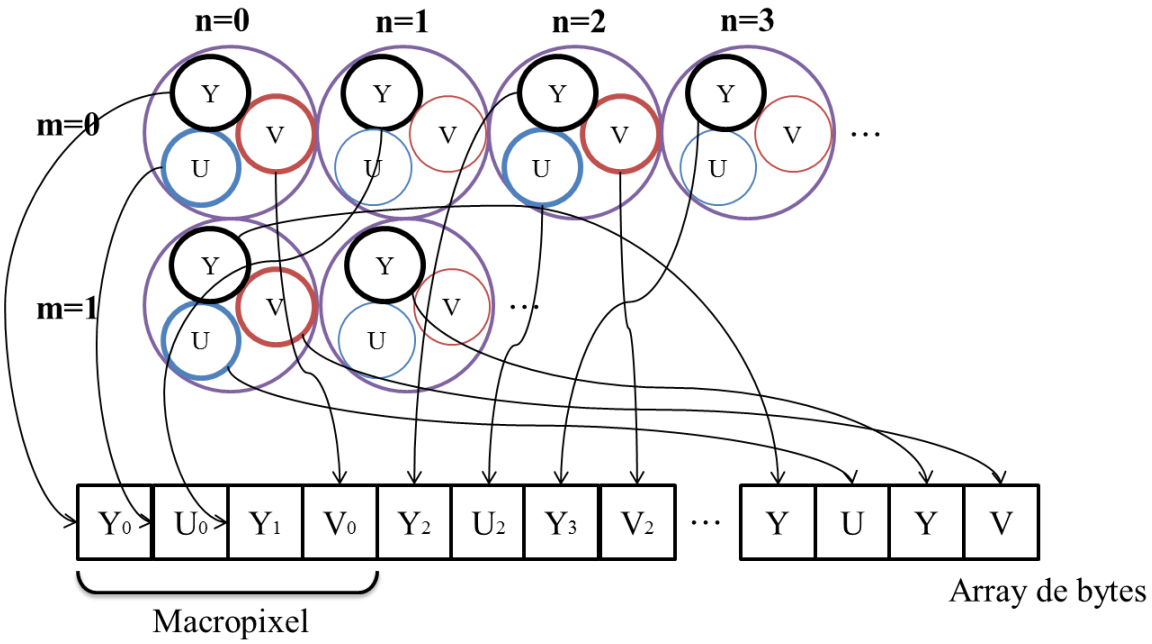


Figura 4.1: Organización de los bytes de datos en array para formato YUY2

Cada macropixel contiene la información de dos pixeles, compuesto por dos muestras de luminancia y una para cada plano de crominancia.

Se utiliza en soportes de video profesional o semi-profesional, ya que guardan una alta calidad de imagen: Betacam Digital, DVC-pro 50 o DVC-pro 100 (HD)

4.2.2. YV12

Este formato almacena los datos por planos (planar format) por lo que en la cadena de array se separan los planos sin inter-mezclar los campos. La luminancia se transporta con todo detalle, a diferencia de la componente de crominancia que se reduce a la mitad tanto en el muestreo vertical como el horizontal obteniendo un valor de V y otro de U para cada bloque de 2x2 de luminancia. Se compone por tanto de un plano Y de (MxN) bytes seguido de otros dos de tamaño $(MxN)/2$ para los planos V y U respectivamente.

	Horizontal	Vertical
Y Sample Period	1	1
V Sample Period	2	2
U Sample Period	2	2

Su estructura organizativa gráficamente:

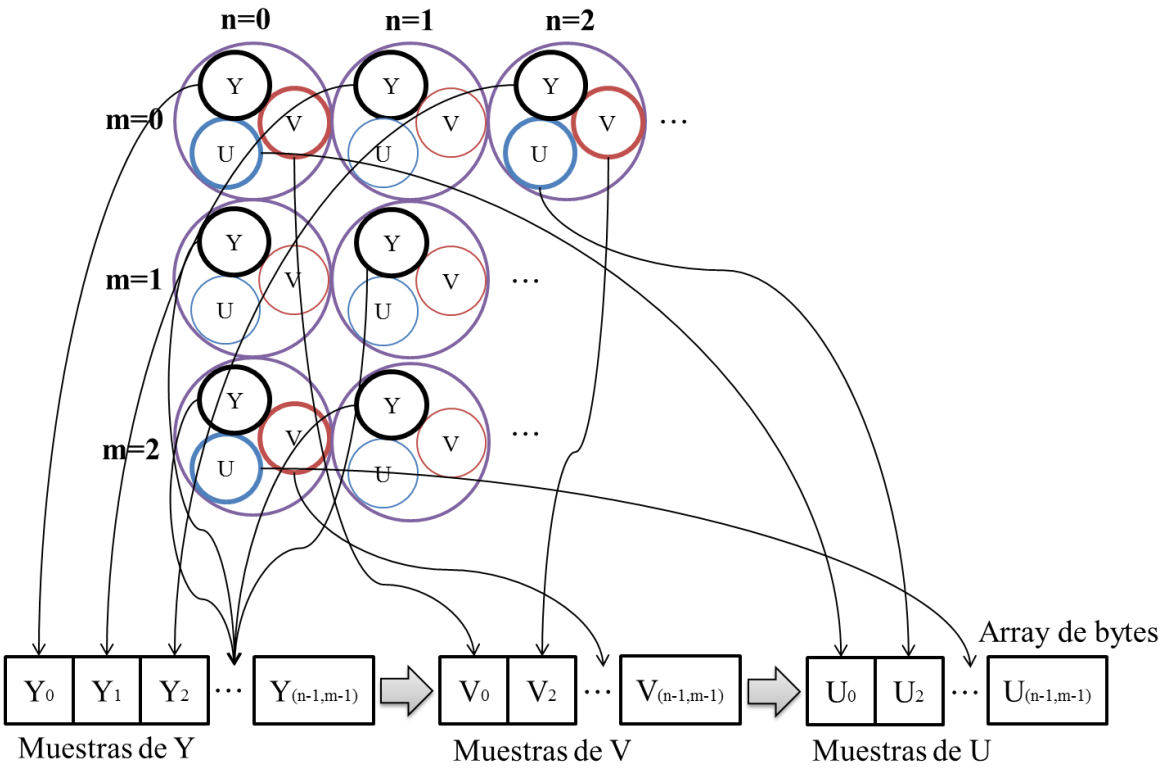


Figura 4.2: Organización de los bytes de datos en array para formato YV12

Este submuestreo utilizado por YV12 se denomina 04:02:00, indicando la periodicidad de muestreo en caja eje y por componentes, muy utilizado comúnmente para digitalización de señales PAL o NTSC.

Muchos códecs importantes almacenan el video en YV12: MPEG-4 (x264, XviD, DivX y muchos otros), MPEG-2 en DVD, MPEG-1 o MJPEG.

4.2.3. RGB32

Este formato es el utilizado para la representación de los videos en Forevid, por lo tanto cada vez que se requiere previsualizar o mostrar un video en la aplicación es necesario convertirlo a este espacio de colores.

Como su nomenclatura dice, es un formato de video RGB donde cada pixel contiene 4 bytes de información, uno para cada componente, añadiendo una componente adicional para la máscara de transparencia Alpha. El número total de bits utilizado por pixel son 32 (nombre del formato) que se ordenan en formato array siguiendo el patrón:

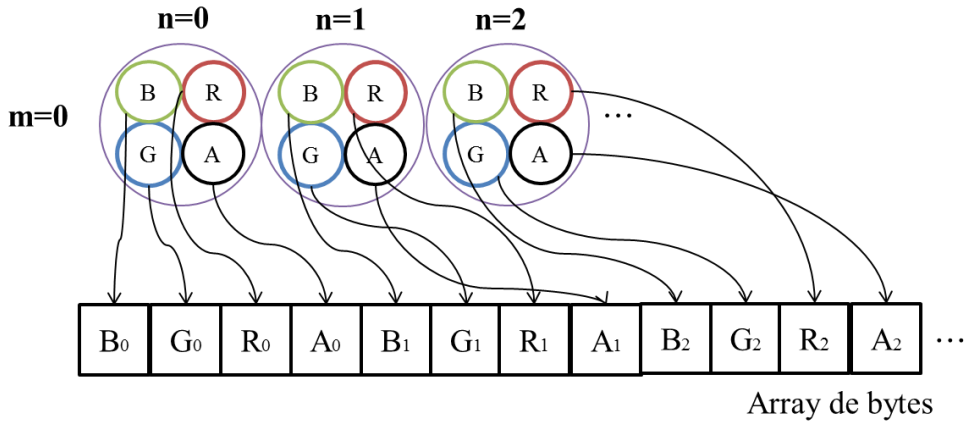


Figura 4.3: Organización de los bytes de datos en array para formato RGB32

Este formato de video es muy adecuado para los procesadores de entorno informático que trabajan con palabras de esta misma longitud, e incluso muchas aplicaciones que no necesitan una máscara descriptiva Alpha para sus imágenes rellenan con ceros los últimos 8 bits correspondientes a este plano en lugar de utilizar otro formato RGB24 ya que la velocidad de procesamiento es mayor.

Este tipo de imágenes o videos se tratan en entornos informáticos considerablemente potentes de tratamiento de imágenes donde no importa reducir el tamaño del archivo excesivamente ni transmitirlo por un medio hostil.

4.3. Frameserver

AviSynth trabaja como un frameserver proporcionando edición instantánea a otros programas sin la necesidad de crear ficheros temporales cuando se requiere transferir archivos de video desde éste a otro en el mismo PC, abriendo el archivo fuente de video y enviándolo a la aplicación. Este proceso ahorra espacio temporal en el disco evitando crear archivos intermedios o temporales en el ordenador y procesando directamente el archivo de video en la aplicación de destino según va llegando desde el frameserver. Además, aumenta la compatibilidad entre aplicaciones al evitar las compresiones de video para transferir el archivo y enviar el fichero fuente original. Si por ejemplo sucediese que la aplicación receptora no acepta archivos comprimidos en formato MPEG-1, pero se recurrió a este método para la transferencia entre aplicaciones, aumentarían los problemas para poder descomprimir e interpretar el video.

4.4. Funciones

Para poder ejecutar los filtros, Avisynth trabaja con dos tipos de filtros:

- **Filtros internos**: Todos ellos están incluidos en un único archivo ubicado en los directorios de instalación del software. Incluye multitud de filtros utilizados en Forevid que iremos detallando más adelante.
- **Filtros Externos**: No están incluidos al realizar la instalación de Avisytnh. Estos filtros se obtienen a través de archivos compilados en librerías dinámicas. Para poder invocar un filtro externo es necesario ubicar el archivo en una carpeta por defecto de Avysinth o bien llamar a una función en el script en cual se va a utilizar que cargue el plugin, indicando su ruta de sistema y nombre.

Un mismo script puede utilizar tantos plugins como se desee realizando la llamada correspondiente para cargarlos y posteriormente invocar las funciones correspondientes. Es evidente por tanto que el script no puede filtrar por sí mismo y que depende de estas llamadas a los filtros internos y externos.

4.4.1. Plugins externos

Se deben manejar varios conocimientos previos antes de comenzar a crear código en C++ para la edición de video. El video se pasa como un parámetro de la función, al igual que el resto de valores, con la característica de estar asociado a una clase predefinida por Avisynth que permite el desarrollo de código siguiendo unas reglas y utilizando constantes, clases de objetos y funciones básicas a las que se pueden llamar desde la cabecera "*avisynth_c.h*", necesaria para compilar cualquier plugin en su creación. Ejecutarse en lenguaje C++ permite encontrar una gran variedad de compiladores gratuitos como *MinGW*, integrados en entornos de desarrollo como *CodeBlocks*, además de ser de uso sencillo para programadores principiantes y aprender rápidamente.

Cada función puede trabajar sobre unos determinados espacios de colores limitados, e incluso, sobre un único formato, ya que la extensión de código se hace mayor cuantos más tipos de espacio de colores soporte sobre los que poder ejecutarse, añadido al hecho de que Avisynth posee filtros internos que permiten cambios entre formatos de video de forma sencilla.

Como se ha dicho anteriormente, las funciones definidas dentro de un plugin (puede haber más de una función incluida dentro de una misma librería) trabajan mediante los parámetros que se les proporciona, caracterizados por un nombre y, sobre todo, un tipo. Se puede interactuar con parámetros de cualquier tipo, los creados por los propios desarrolladores u otros más simples: enteros (*int*), decimales (*float*), cadenas de texto (*string*), o booleanos (*bool*). Estos datos servirán para modificar la imagen con los ajustes requeridos por el usuario actuando en el nivel más bajo, pixel a pixel, y devolviendo un valor de retorno de tipo *clip* si no ha ocurrido ningún error durante la ejecución, aunque en la enorme mayoría de los filtros los parámetros ya vienen con un valor asignado por defecto, en cuyo caso con solo aplicar el comando del filtro el script

es funcional. Cada función recibe la llamada para su uso identificándola por un nombre o texto que se define intrínsecamente en el código del plugin, e incluso, ciertos programas que trabajan a través del desarrollo de scripts para Avisynth, son capaces de identificar los parámetros que requiere esa función (tipo de parámetro y nombre) a medida que se van introduciendo los valores (AvsP[15] o VirtualDub). 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, obviamente conociendo que representa cada parámetro (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 puede asignar su valor en cualquier orden mientras se los nombre. Esta es una forma mucho más cómoda y versátil, ya que permite sólo asignar valores a aquellos parámetros que se quieren modificar por su valor por defecto.

4.5. Carga de archivos fuente

Para que Avisynth pueda trabajar con los ficheros de video/audio es necesario previamente cargarlos. Mediante esta operación la aplicación reconoce cada uno de los frames del video de forma exacta para realizar posteriores ediciones sobre la fuente correctamente.

Dependiendo del formato contenedor en el que el archivo fuente esté empaquetado se debe utilizar una librería u otra. En Forevid se han utilizado tres funciones de Avisynth para esto, algunas están incluidas en sus filtros internos (Avisource, DirectSourceShow), y otra externa que es necesario cargar a través de su plugin (FFmpeg).

Ya que Forevid no realiza edición de audio, a la hora de cargar los ficheros existe la posibilidad de ignorar los datos relativos a los mismos colocando *audio=false* en la línea de script donde se realiza la llamada a la función.

4.5.1. AviSource

Permite cargar archivos *.avi, *.wav. *.avs o *.vdr sin problemas siempre y cuando alberguen formatos de video estándar y no otros más avanzados como H264. Incluye tres funciones de carga (AviSource, AviFileSource y OpenDMLSouerce), y dependiendo del tamaño del archivo que se desee cargar es conveniente utiliza una u otra. Para realizar la carga correctamente es imprescindible facilitar la ruta o directorio completo del archivo (a no ser que éste se encuentre en la misma carpeta donde se está ejecutando el script) y el formato de pixel en la salida del decodificador, pudiendo ser: YV12, YV411, YV16, YV24, YUY2, Y8, RGB32 y RGB24 (si se omite Avisynth usa el primer formato soportado respetando el orden mostrado).

[15] http://www.avisynth.nl/users/qwerpoi/

Notar que archivos *.avs se pueden importar, scripts en los cuales se basa Avisynth directamente para ejecutar los comandos de edición de video explicados al inicio del apartado Avysinth en este documento.

4.5.2. DirectShowSource

Permite cargar otros formatos haciendo uso de la interfaz DirectShow, demultiplexando y decodificando cualquier formato que posea un Decoder o Splitter directshow instalado en el sistema, incluyendo la mayoría de los códecs que el sistema de reproducción de Windows utiliza.

Uno de los problemas al utilizar esta función es que necesita que se le proporcione el framerate del video en la mayoría de los casos, por lo que necesita el parámetro fps que fije este valor. En videos con una frecuencia de cuadros variable, VFR, se puede utilizar la opción *convertfps=true* para convertir a un framerate constante mediante duplicación de cuadros. Las opciones que ofrece para el formato de salida del clip incluye: YV12, YUY2, ARGB, RGB32, RGB24", YUV y RGB.

4.5.3. FFmpegSource2

A diferencia de los anteriores, se puede utilizar tras la carga del consecuente plugin externo, el cual incluye un conjunto de filtros que hacen uso del proyecto FFmpeg para la carga de los archivos, ya sean de audio y/o video.

De las múltiples funciones que posee para importar determinadas características del video (*FFIndex*, *FFAudioSource*, *FFmpegSource2*, *FFFormatTime* o *FFInfo*), ya que únicamente se desea importar el video la que se va a invocar será FFVideoSouce en la aplicación de Forevid.

La indexación del video se realiza automáticamente al cargar el archivo multimedia, aunque puede ser controlada por el usuario mediante la aplicación interna *ffmsindex.exe* que proporciona esta librería o realizarse de forma individual separada de la carga de video mediante *FFIndex*.

Los formatos de video soportados son: AVI, MKV, MP4 y FLV como aquellos cuyo framerate es detectado, WMVy OGM, en los cuales puede ser detectado pero no se asegura, y VOB, MPG, M2TS y TS donde no se garantiza la localización del framerate, ocasionando posibles deviaciones ligeras en la numeración de frames.

Algunos de los parámetros que se deben definir para la correcta carga del archivo son:

- **Track**: sirve para eligir que pista de las indexadas debe ser cargada. Esta función sólo es capaz de cargar una pista simultáneamente.
- **fpsnum y fpsden**: Fija el valor del framerate del video cuando se busca aplicar una conversión desde un video con frecuencia de imagen variable (VFR → CFR). Especifica el valor mediante la relación: fpsnum / fpsden. Por ejemplo: para 29,97: fpsnum=30000 y fpsden=1001. Esta información se puede obtener con las funciones que incorpora Avisynth.

Los videos que son importados mediante el arrastre de los archivos de vídeo directamente desde el Explorador de Windows a la lista de los vídeos importados utilizan este método de FFmpeg al ser el que mayor cantidad de formatos soporta y más flexibilidad permite.

5. Forevid

Forevid[16] es una aplicación software de código abierto desarrollada a partir de los elementos y conocimientos anteriormente descritos, implementada para el uso del análisis de imágenes forenses a partir de grabaciones fijas de video. Sus autores son el laboratorio forense del centro nacional de investigación de Finlandia, y ha servido de uso para la policía de dicho país, debido a la cantidad de posibilidades que ofrece y las expansiones que agregan nuevas funcionalidades que se van sumando. Como ejemplo claro del avance de esta aplicación, en la versión utilizada para el desarrollo de nuestro proyecto (Forevid 1.0.5) ha quedado atrás respecto a la última versión publicada del software (Forevid 1.2.1) en cuanto a funcionalidades, añadiendo entre otras cosas: captura de pantalla del dispositivo donde se ejecuta con CamStudio, o edición de imágenes fijas como cambios más significativos. Entre medias de ambas versiones se han publicado otras (1.1.0 y 1.2.0) que han ido añadiendo nuevas funciones de tratamiento de imagen, en su mayoría ya incluidas dentro de los filtros internos de Avisynth:

- Filtro *Loop*: Agregado de filtro para recorrer permanentemente de vídeo para una determinada cantidad de veces.
- Filtro *AssumeFPS*: filtro añadido para cambiar la velocidad de reproducción asumiendo un nuevo framerate de vídeo.

Otras mejoras se han centrado en el aspecto y accesibilidad de la UI:

- Opción para recorrer los vídeos en bucle realizándolo ya sea para una parte seleccionada desde un fotograma a otro o para todo el video.
- Se han añadido traducciones tanto para el idioma alemán como para polaco.
- Al añadir filtros, el video de vista previa donde se representa del resultado busca la posición del video original.
- Opción para localizar la ruta de un archivo de video en el disco duro y abrir en una ventana nueva la misma.
- Con el fin de facilitar la unión de los videos combinándolos y añadiendo títulos o barras, éstos se escalan automáticamente al mismo tamaño y se cambia su velocidad de fotogramas para hacerlos denominador común a todos ellos e impedir erores de compatibilidad.
- Nueva opción para añadir al documento de resumen del proyecto en pdf que enumera todos los vídeos incluidos en el proyecto y las operaciones de tratamiento que se aplican a ellos.
- Compresión UPX eliminado de los archivos binarios.

Como vemos, la mayoría de las mejoras han sido centradas en mejorar la accesibilidad de la UI en la reproducción de los videos, además de ampliar otras funcionalidades que ya tenía. En nuestra aportación a este proyecto, en cambio, las mejoras se han centrado

[16] http://www.forevid.org/

en añadir nuevas funciones y plugins que permitan nuevos procesos de tratamiento de imagen enfocados en su mayoría a un punto de vista didáctico.

La aplicación se desarrolla y despliega mediante llamadas a otros módulos o scripts desde el principal que se van ejecutando en tiempo real. Todas las acciones relacionadas con la reproducción de video, marcaciones de fotogramas, exportación de los mismos, cortes de video, o aplicación de filtros conlleva realizar llamadas a las funciones de Avisynth, por lo que esta parte de interactuación entre Forevid (Python), y las librerías de edición de video es muy importante y debe permanecer sólida ante errores y aplicación de varios filtros en cadena. En el siguiente punto se verá con detalle la estructura de esta parte de la aplicación tan importante, tanto a nivel interna de Forevid como a nivel de accesibilidad del usuario, pero antes de eso vamos a repasar el funcionamiento general a través de un esquema-resumen que engloba todas las partes y usos de la aplicación, además del camino recorrido por un archivo multimedia desde que se importa hasta que se almacena en el disco de nuevo tras haber sido modificado y exportado en alguno de los formatos habilitados.

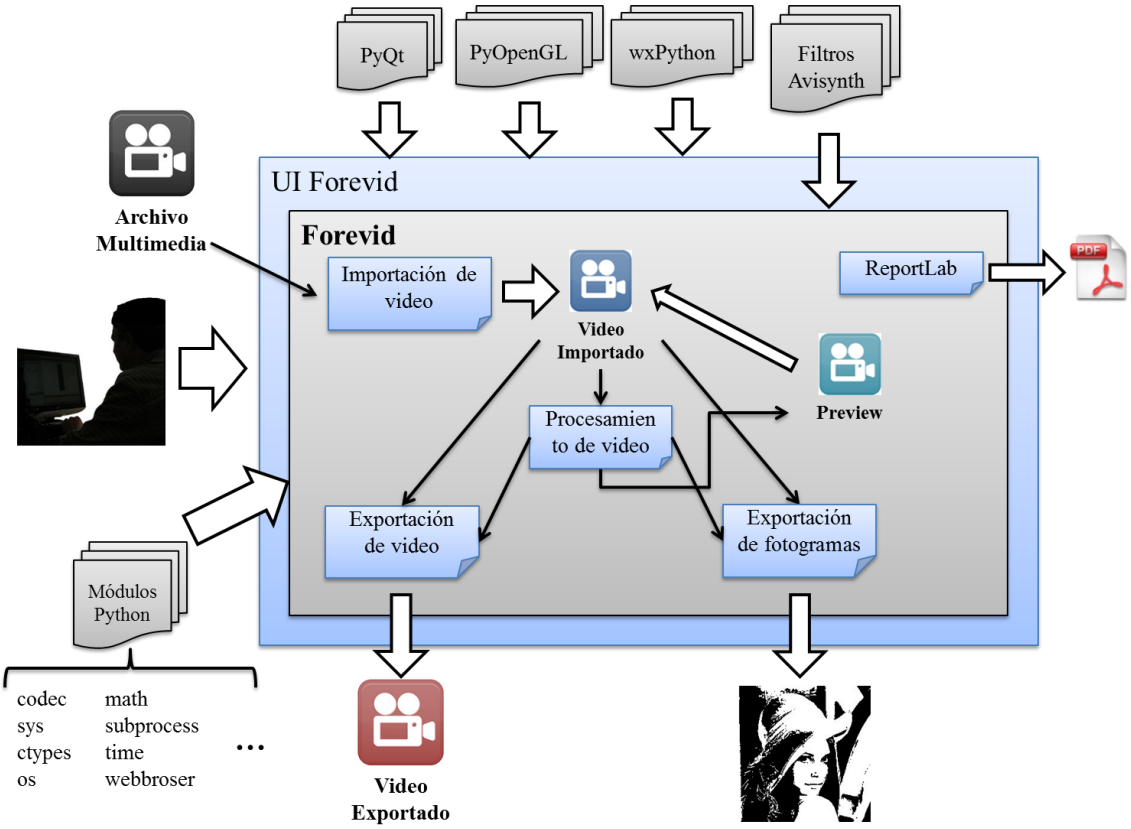


Figura 5.1: Esquema de las principales funcionalidades de Forevid

Se han omitido muchos de los módulos para facilitar la lectura del recorrido y finalidad principal que no es otra que la edición de video y análisis del mismo a través de la exportación de los fotogramas que la componen. Como se puede observar, muchos de los pasos que se muestran ya han sido detallados con anterioridad como es: la importación del video a través de librerías Avisynth, uso de APIs que implementan parte de la UI o el módulo de creación de resúmenes de proyecto en formato *.pdf.

De aquí en adelante se tratará de exponer las partes más relevantes sobre la cuales se ha desarrollado este proyecto en su mayoría, funciones ya creadas que han servido de modelo y estudio para el aprendizaje, y pasos a seguir para completar la ampliación que se ha llevado a cabo. Empezaremos por el estudio en el funcionamiento de la aplicación para aunar las acciones necesarias para la edición de video y cómo se recopilan dichos datos desde el usuario a través del dispositivo que ejecuta: los filtros de edición y procesamiento de video.

5.1. Estructura de filtros en Forevid

Es importante que ambas partes, tanto la de interactuación con el usuario como la definición interna de Forevid para cada filtro, estén en sintonía y sean recurrentes una con la otra, en este caso la parte definida de filtros sirve para la UI de la aplicación. Para ello se ha creado una única clase en Python definida en *avs_filter.py* que contiene todos los filtros, y que se declara en el módulo *filterWindow.py* para mostrarlos en un widget y poder añadirlos. Ambas partes, aun siendo recurrentes y necesarias bidireccionalmente, tienen sus propias características y organización ya que, evidentemente, cada parte tiene finalidades diferentes. A modo de resumen general hemos elaborado un esquema del recorrido de un archivo multimedia desde que es importado a Avisynth hasta que es editado, pasando a través de los recursos explicados en puntos anteriores de este documento y los elementos externos necesarios que utiliza la aplicación:

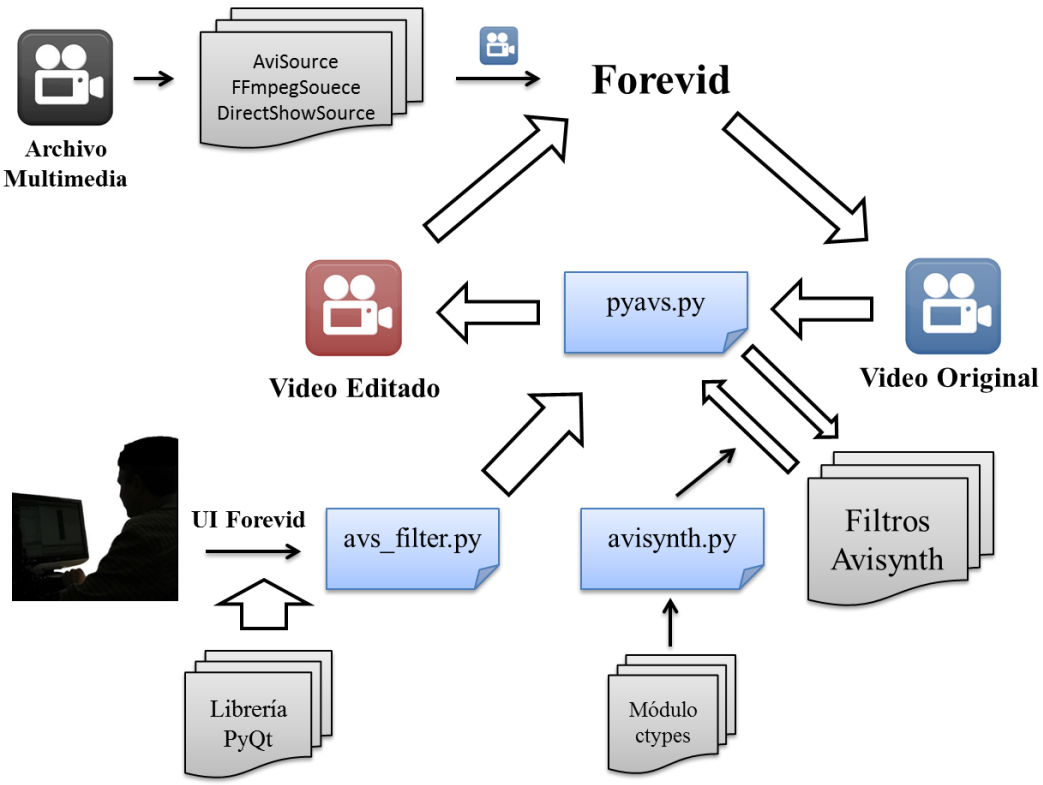


Figura 5.2: Recorrido de la edición de un clip de video en Forevid

Como podemos ver, la pieza que reúne todos los elementos aunándolos y utilizándolos entre sí es el archivo pyavs.py, que recibe como argumentos dentro de su clase AvsClip

el video o videos originales que se van a tratar en el array *source*, y los filtros en otro parámetro del mismo tipo llamado *filters*. Además de estos dos elementos, otro fundamental es la importación del módulo avisynth.py que incorpora el módulo ctypes, proporcionando las funciones y parámetros necesarios para comunicarse con los filtros de edición de video, los cuales únicamente entienden lenguaje C++. Todos los filtros externos que se quieren utilizar en la aplicación se deben introducir en el directorio AVS_Plugins de la aplicación. Este podría ser el recorrido o video básico de un video a través de la aplicación Forevid sin entrar en mayores detalles (posteriormente se desarrollará por partes cada una de las fases).

5.1.1. Interactuación con Avisynth

Para este objetivo se ha creado un objeto de clases en el módulo *avs_filter.py* compuesto por otros sub-objetos que definen todos los datos y parámetros necesarios para que se pasen a un entorno de creación que pueda ejecutar Avisynth. Una cadena de estos objetos de filtros Avisynth se encarga de recoger las peticiones de edición para cada uno de los videos abiertos por la aplicación, incluyendo posibles cambios de espacio de colores necesarios entre filtros, ya que ciertos filtros sólo pueden ejecutarse en un determinado espacio de colores, y puede suceder que los siguientes a aplicar no correspondan con éste.

No todos los filtros que están definidos en el script son los utilizados por la aplicación, sino los que el usuario puede añadir directamente desde la ventana de edición de la aplicación, ya que hay otros muchos que se utilizan indirectamente al realizar otro tipo de operaciones que no implica el uso de filtros como tal: exportación de fotogramas, recorte de fotogramas del video, importación de archivos multimedia a Forevid, transformación de espacios de colores, creación de pequeños clips como cabecera de otros…e incluso, en diversas ocasiones se hace uso de la función "*eval*", la cual funciona como ejecutor de código script puramente dicho, cuyo único parámetro de funcionamiento es un *string* que se evalúa como texto de un archivo *.avs, en el que sus comandos se van ejecutando siguiendo las ordenes de la cadena de texto.

Volviendo a los filtros que el usuario puede manipular, todos tienen la misma estructura y se recurre a la misma clase python, tanto para mostrarlos y que el usuario pueda introducir los valores deseados, como para importarlos al módulo que recogerá esos valores y los pasará a C para que puedan ser entendidos por los filtros de Avisynth. La composición de éstos es la mostrada en la siguiente figura:

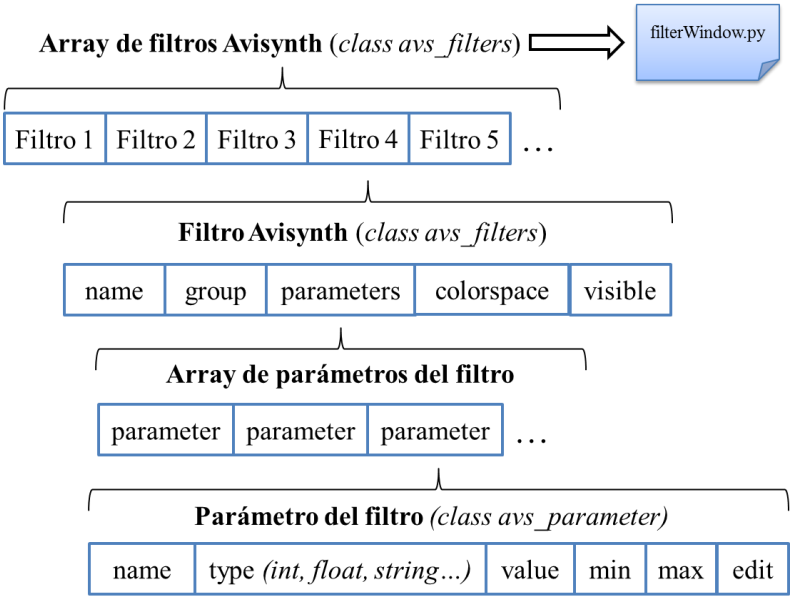


Figura 5.3: Estructura de objetos Python para definición de los filtros

El objeto que engloba toda la cadena de filtros se exporta al módulo filterWindow.py que se encargará de dar salida a los mismos de cara al usuario, poder añadirlos, y elegir los valores de cada uno de sus parámetros. Cada filtro se compone de un identificador o nombre que servirá tanto para el usuario como para las librerías dinámicas Avisynth que reconocen y ejecutan el comando correspondiente. El grupo sirve para la organización en tipo árbol desplegable de los filtros que se muestra en la ventana de acceso al procesamiento de video. También se compone de un determinado número de parámetros, espacio de colores en los cuáles se puede ejecutar el filtro y si será accesible para el usuario de forma directa. Un filtro puede tener varios parámetros o ninguno, y son los encargados de dar la posibilidad de actuación al usuario de forma subjetiva y personalizada. Se identifican a través de un nombre, un tipo, y un rango de valores posibles entre los que puede moverse.

5.1.2. Interactuación con el usuario

Para acceder a la ventana que permite añadir filtros se ha creado un módulo que recoge y gestiona toda la parte de UI: *filterWindow.py*. Desde aquí se ejecuta una ventana que muestra todos los filtros disponibles importados desde *avs_filter.py*, organizados en forma de árbol dentro de un widget. Otro cuadro indica los que se han añadido:

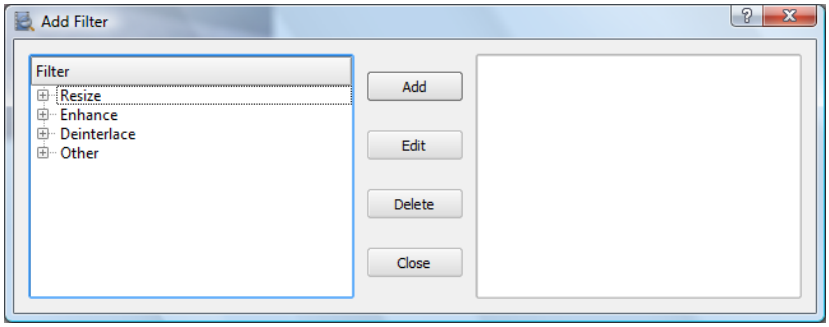


Figura 5.4: Ventana de filtros disponibles

Cada vez que se selecciona un filtro con doble click o la opción “Add”, una ventana emerge se ejecuta para que el usuario pueda introducir los valores de los parámetros de ese filtro con el nombre del mismo en el título de la ventana, además de la posibilidad de abrir una vista previa del video con el filtro aplicado, otro de actualización de valores de parámetros y otros botones para, o bien aceptar el filtro e introducirlo en la cadena, o bien cancelar la ventana y volver a la ventana previa sin realizar cmabios. Inicialmente existen tres tipos de ventanas emergentes u objetos para este cometido definidos en *filterWindow.py*, y dependiendo del tipo de filtro que se ejecute recurrirá la llamada a:

- *ParameterWindow*: Se utiliza para la mayoría de filtros. En su ventana se van introduciendo clases de objetos que permiten seleccionar el valor de cada parámetro mediante deslizadores (sliders) de tipo enteros o decimales que marcan los valores posibles entre el mínimo y el máximo para dicho parámetro, También se muestra el nombre de cada parámetro encima del slider para identificar cada parámetro.

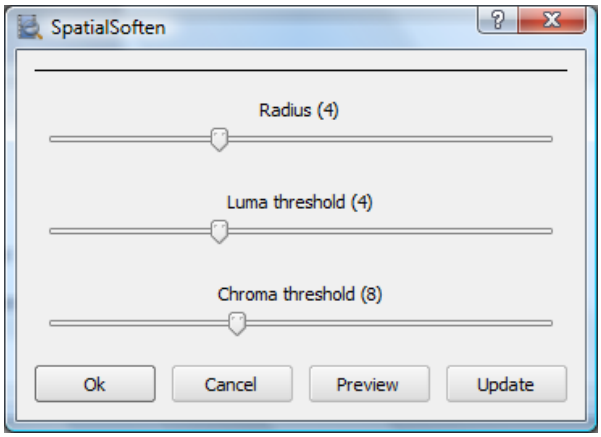


Figura 5.5: Ejemplo de ParameterWindow. Spacial Soften

- *LevelsWindow*: Únicamente se ejecuta para el filtro *levels* de Avisytnh, mostrando una gráfica con el histograma de la imagen actual del video para cada canal de color (R, G y B) y sobre ella la repercusión de los valores de los parámetros introducidos.

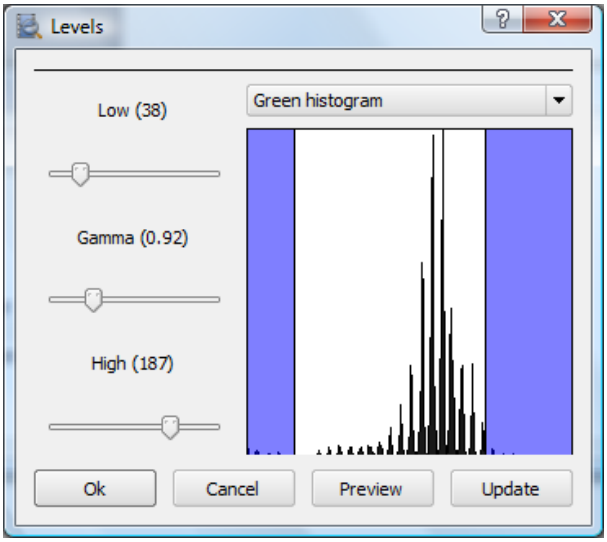


Figura 5.6: Ventana filtro Levels

- *ResizeWindow*: Sirve para los filtros dedicados al redimensionado del video: Point resize, bilinear resize, bicubic resize, spline36 resize y lanczos4 resize. Su particularidad reside en poder introducir en cuadros de texto las dimensiones que queramos que tenga el video manteniendo o no la relación de aspecto original.

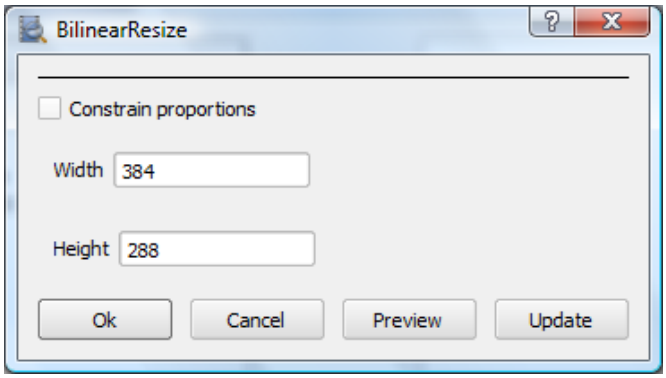


Figura 5.7: Ejemplo ResizeWindow. Bilinear Resize

Para introducir los valores otras tres clases de objetos se han desarrollado en función del tipo de valor: intSlider, floatSlider e intBox. Como su propio nombre indica, los dos primeros se muestran gráficamente como un deslizador para seleccionar un valor determinado entre un mínimo y un máximo, y un nombre que sirve de identificación del mismo. Toda esta información va incluida en la clase *avs_parameter* descrita en el punto anterior y que se pasa como parámetro para mostrarse. *intBox* se utiliza únicamente para los filtros de redimensionado de imágenes *ResizeWindow* que se puede ver en la captura anterior correspondiente a esta clase.

Durante el desarrollo de ampliación de programas hubo que crear otras clases de parámetros e interfaz asociada que se detallarán más adelante, ya que con estos tres tipos explicados no completaban el abanico necesario para hacer más fácil el acceso al usuario la introducción de ciertos parámetros en los nuevos filtros.

5.2. X264, exportación de videos

Aunque ya se ha tratado el tema de importación a Forevid y se puede pensar que el proceso puede ser similar, no debemos equivocarnos. Si en el primer paso se utilizaban filtros tanto internos como externos de Avisynth para tratar con diferentes formatos, en este caso se recurre a un programa de encodificación de streams que ofrecen diferentes perfiles dependiendo de la calidad deseada (que influirá en el peso final del archivo), características propias específicas de reproducción de video y el tipo de contenedor de video.

Se recurre pues a un sotware ya creado, que funciona sobre comandos, y que está de implementado por VideoLan[17], un proyecto y organización sin ánimo de lucro integrada por desarrolladores que ofrecen soluciones para archivos multimedia de código abierto y libre acceso, entre otros productos se encuentra el reproductor VLC, multicat o VLMC.

[17] http://www.videolan.org/

En este caso hablamos de **x264**, una librería que permite las funcionalidades requeridas para la aplicación Forevid distribuida bajo los términos de la GNU GPL como sucede con el resto de los componentes y herramientas empleadas en el proyecto. Permite codificaciones en formatos de video de los estándares H.264 y MPEG-4 para los sistemas operativos Windows, Linux y MacOSX mediante formato de línea de comandos donde se van ejecutando las órdenes y parámetros necesarios. Existe otra variante como codificador de video tipo "*Video for Wondows*", un framework multimedia no utilizado en Forevid, pero sí como reproductor en los sistemas operativos de Microsoft Windows para operar con ficheros de video.

La versión utilizada en nuestra aplicación ejecutada a través de línea de comandos, los cuales se crean desde el módulo *encodeWindow.py*, se mantiene más actualizada y no requiere de una interfaz de usuario para su uso. Las principales características a destacar que permite este software son:

- De los pocos codificadores de tipo "High Profile AVC" a disposición del público
- Todo tipo de predicción para macro-bloques I y P de tamaño 16x16, 8x8 (parte del perfil *High* integrado en AVC) y 4x4.
- Predicción de macro-bloques para frames tipo B desde 16x16 hasta 8x8.
- Permite orden adaptativo de fotogramas B, posibilitando tener es tipo de cuadros como referencia de otras.
- Permite modo *lossless* (o sin pérdida de calidad) con el perfil "*High 4:4:4*".
- Característica "*adaptive quantization*" que permite utilizar diferentes valores de cuantificación para diferentes partes del frame asignando de forma más efectiva los bits en áreas más complejas mejorando la compresión del video.
- Las versiones base del códec en su inicio no eran compatibles con los estándares Blu-ray/HD-DVD por falta de metadatos que posteriormente se crearon y parchearon.
- Constituye el núcleo de muchos servicios web de vídeo como Youtube, Facebook, Vimeo o Hulu

Los contenedores disponibles en x264 para la codificación de videos incluyen Matroska (*.mkv), MPEG-4 Parte 14 (*.mp4) y Flash (FLV). Además de esto, el vídeo codificado también se puede exportar envuelto dentro de un reproductor de vídeo de extracción automática (SFX player) que automáticamente reproduce el vídeo incluido en un paquete ejecutable (*.exe).

Este es uno de los módulos ampliados en posteriores en versiones posteriores a Forevid1.0.5 permitiendo mayores perfiles de codificación y exportación de videos editados.

5.2.1. Exportación de archivo AVS

Como se vio en la importación de ficheros, los archivos *.avs propios de scripts Avisynth pueden ser utilizados como órdenes de ejecución dentro de Forevid, por lo que no es de extrañar que también se pueda realizar la acción propia de exportación, creando

un archivo del mismo formato que contenga todas las acciones aplicadas a un clip de video a través de los comandos y llamadas a funciones internas o externas Avisynth.

Por supuesto, para que estos archivos exportados puedan ser ejecutados en su totalidad fuera de la aplicación, deberá tenerse en cuenta que las librerías necesarias deben estar referenciadas e incluidas en las rutas del sistema.

Nos hemos centrado en detallar el funcionamiento de la aplicación a la hora de realizar su principal cometido de procesamiento de video y todo lo relacionado con ello. A pesar de esto, muchas funcionalidades se han omitido ya que se consideran secundarias o se han descrito de forma breve porque se ha considerado necesario para comprender otras partes mayores de Forevid que la engloba.

6. Nuevas funcionalidades en Forevid

En este punto vamos a conocer el desarrollo y trabajo principal del proyecto, analizando los puntos sobre los que se ha trabajado y cómo se ha hecho, cómo se investigó el problema que se pretendía resolver y cuál ha sido la forma de subsanarlo con las herramientas de las que se disponían, algunas de ellas ya explicadas en anteriores puntos y que es importante conocer para entender por qué se recurrió a esta metodología y no otra.

6.1. Traducción de la aplicación Forevid

Como toma de contacto con la aplicación y conocer el entramado y funcionamiento del mismo, se comenzó por realizar la tarea de incluir una traducción a la lengua castellana para todos los textos que lo componen. El proceso se realizó en varias fases necesarias y consecutivas debido a que no todos los strings incluidos en el mismo derivaban del mismo proceso, o por las características propias del idioma castellano, que incluía caracteres alfabéticos no soportados por Python en un inicio. Pero veamos los pasos uno a uno sin adelantar acontecimientos.

6.1.1. Cambio de tabla alfanumérica

Por defecto, el compilador de Python utilizaba una tabla alfanumérica de caracteres que no reconocía vocales con tilde o la letra ñ propias del idioma castellano, por lo que al introducir en los archivos de código Python alguno de estos caracteres, se mostraba un carácter no correspondido o erróneo.

Para solucionar este problema, sencillamente se incluyó una sentencia al inicio de cada archivo que indicaba al compilador que tipo de tabla utilizar para transformar o normalizar los textos Unicode por defecto a otros iso-8859-15, a partir de los cuales no volvieron a surgir este tipo de caracteres sin sentido en mitad de un texto de la UI de Forevid respetando el idioma y la compresión total de los usuarios.

6.1.2. Traducir y cargar textos

Esto se realiza a través de la anteriormente citada aplicación contenida en PyQt, QtLinguist. Con ella podremos crear un archivo que indica a la aplicación qué texto se debe suplir por el original escrito en el código, pero antes de esto debemos indicar con una instancia ofrecida por la librería QtCore llamada QTranslator, cuáles son esos textos. Para ello, todos y cada uno de los textos que se sustituirán están pasados como parámetros por una función citada por “tr” o “translator”.

Algunos de los textos que se han querido incluir como traducibles, y que en la versión sin modificar no se encontraban, eran entre otros los nombres de los filtros y sus parámetros definidos, o los cuadros de texto que se mostraban dentro de las ventanas de cada filtro que no venían definidas en la clase *parameterWindow*, es decir, en la ventana *levelsWindow* y *resizeWindow* cuyas capturas se han mostrado. En el archivo *avs_filter.py* hubo que incluir las instancias necesarias heredadas que sirven para indicar

estos textos ya que no se encontraban dentro de este módulo, a diferencia de filterWindow que ya poseía estas instancias heredadas.

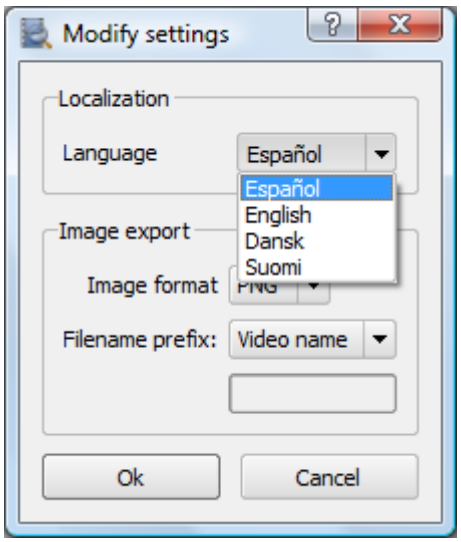
Una vez referenciados todos los textos que se desean traducir, mediante el comando *pylupdate4 -verbose* seguido del nombre de todos los archivos python de la aplicación que contienen strings a traducir y el nombre del archivo *.ts sobre el que trabajaremos, se crea el archivo de datos ejecutándolo por consola sobre el mismo directorio en el que están los archivos de la aplicación.

Utilizando QtLinguist abrimos el fichero creado *.ts y podemos asignar a cada fraso o palabra la traducción deseada, incluso opciones adicionales como creación de un diccionario de traducciones que permite realizar traducciones instantáneas para una palabra ya guardada si se repite en varias ocasiones. Cuando guardamos el archivo, exportamos un fichero *.qm que finalmente cargaremos en la aplicación a través del objeto *QTranslator*.

Con este proceso realizado, y tras cargar las traducciones en la aplicación, se observa que quedan popups y otros widgets cuyos botones se siguen mostrando en el idioma estándar, el inglés. Este problema se debía a la aplicación de este leguaje como el lenguaje local por defecto, y todos estos elementos que se mostraban en la aplicación se llamaban directamente de la API proporcionados por PyQt. Para corregirlo hubo que recargar de nuevo, mediante el mismo objeto *QTranslator*, la ruta local del sistema que indica a la API el idioma que se utiliza en el sistema operativo que está corriendo la aplicación. También se tiene que cargar el objeto en la app. como sucedía con el anterior archivo creado *.qm a través de la instancia *installTranslator*.

Con esto ya tenemos traducida la aplicación completa, e incluso ciertas ventanas que anteriormente no estaban integrados como parte del total a traducir inicialmente.

6.1.3. Añadir botón en la interface para elegir el idioma español



Para que el usuario tuviera acceso a aplicar esta nueva versión castellana se tenía que proporcionar, junto con el resto de ajustes de idiomas que ya tenía, una nueva pestaña en el widget de lista desplegable ubicado en la venta de modificación de ajustes de la aplicación que se muestra ala izquierda, desarrollada dentro del módulo *dialogs.py* y que aparte de esta ventana ejecuta otras que se muestran como por ejemplo cuando se añaden fotogramas a los marcadores, se desea ver la información sobre Forevid, o ver la información de un clip importado. Si observamos dentro del mismo, nos damos cuenta que la clase que ejecuta la ventana que nos interesa es *settingsDialogs*, y el objeto que contiene las opciones de idioma *language* de tipo *QComboBox*. Únicamente debemos crear una item

de texto ‘Español’ que se añada al widget caja. La opción elegida en el mismo se recogerá en un array declarado para toda la aplicación y que también recoge el resto de opciones de esta ventana de ajustes para su verificación desde el módulo principal *forevid.py*, en la instancia *changeLanguage* de la clase *forevid*, y decidir que archivo de traducciones cargar.
La ventana que se mostraba informando al usuario del cambio de idioma en el próximo reinicio cuando se cambiaba el idioma se eliminó desde la instancia *settingsModify* de *forevid* ya que, como se verá en el siguiente apartado, se ha implementado un cambio mediante el cual no será necesario esta acción para que los cambios de idioma se hagan efectivos en la misma ejecución de la aplicación.

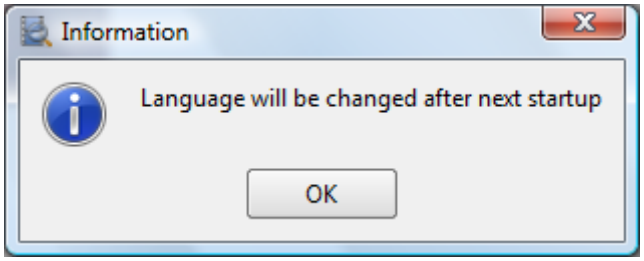


Figura 6.1: Mensaje informativo eliminado tras las mejoras aplicadas

6.1.4. Traducción dinámica de Forevid

Se persiguió insistentemente esta mejora, considerando un déficit notable el tener que reiniciar la aplicación cada vez que los cambios aplicados en el idioma se viesen reflejados.

De un inicio se trató de implementar un reinicio automático y lo más trasparente y rápido posible de Forevid cuando se detectaba el cambio de lenguaje que aplicase el cambio de forma forzada, pero esto implicaba un inicio manual de la aplicación y complicaciones de guardado del proyecto abierto antes de cerrarse. Cambiando de estrategia y buscando otras formas, se cayó en cuenta que no era necesario reiniciar la aplicación completa sino, en lugar de esto, recargar la UI o ventana principal de la aplicación, y únicamente ésta, ya que el resto de ventanas aplican el archivo de traducciones cargado cuando se ejecutan y no al inicio del programa, por lo que el problema se centraba en esta ventana main desde donde se accede a toda la aplicación.

Para esto se introdujeron dos instancias nuevas dentro de *forevid.py*, además de otros cambios en *settingsModify*:

- ***changeLanguage(language)***: Se ejecuta dentro de *settingsModify* y carga en la aplicación los strings traducidos con el idioma ***language*** que es pasado como parámetro de tipo texto de la instancia determinando el archivo *.qm creado con QtLinguistic a cargar. En la versión original esta acción se realiza antes de ejecutar la ventana principal, pero para poder llamarla una vez que se ha aceptado la ventada de ajustes es necesario introducir la instancia dentro de la ejecución de la aplicación.
- ***retranslateMainWindow***: Actualiza todos los textos que se muestran en la ventana principal cargando la última traducción seleccionada en la aplicación. No es necesaria esta instancia en el resto de archivos *.py de la aplicación que ejecutan las

ventanas que se llaman desde la ventana principal como se ha comentado anteriormente, ya que al ejecutarse muestra los strings cargados del último archivo de traducciones. Se ejecuta a posteriori de la carga del archivo de traducción.

Por desconocimiento de los idiomas finés y danés, los textos que se han ido añadiendo han quedado sin traducción en su idioma correspondiente al seleccionarlo, por lo que se mostrarán los textos originales que se han escrito en el código, el inglés, y sólo se mostrarán traducidos aquellos textos que ya estaban incluidos en la versión sin modificar. En las versiones castellana y anglosajona la traducción es completa en todas las ventanas, ya sean emergentes, popups, ventana principal, botones del reproductor de clip implementado desde *videoPlayer.py*, barra de estado en la parte inferior de la ventana que además de información extra sobre el elemento señalado sobre la aplicación, incorpora el indicador de fotograma y tiempo del clip.

6.2. Descriptores en puntero y barra de estado para elementos

Muchos de los elementos de la ventana principal tenían una descripción al seleccionarlos con el puntero del ratón, tanto en la barra de estado como sobre el propio ratón al mantenerlo brevemente sobre el elemento.

Los widgets de reproducción no poseían estas características, y es que estos elementos se cargan desde otro módulo *videoPlayer.py* que genera la funcionalidad de reproducción sobre el video en el cual se trabaja, por lo tanto para corregirlo se ha trabajado sobre dicho módulo, tanto para añadir estos detalles como para su correspondiente traducción sin necesidad de reiniciar la aplicación en concordancia con el resto de la aplicación.

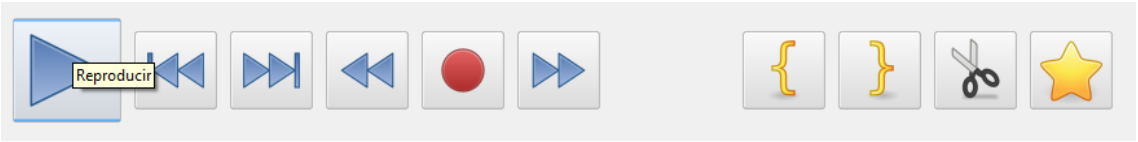


Figura 6.2: Descriptor de puntero para iconos de reproductor de video

Para esto último, al igual que ocurrió en el módulo *forevid.py*, se crea una instancia ***retranslateVideoPlayer*** que carga de nuevo los textos de estos mismos botones en *videoPlayer.py*, y que introducimos en la instancia ***retranslateMainWindow*** del módulo principal ejecutado a través del objeto *self.player* declarado para tal funcionalidad.

Para que apareciese texto descriptivo de puntero y en la barra de estado sobre estos iconos hubo que utilizar dos instancias proporcionadas por la API: *setStatusTip* y *setToolTip*. Para cada uno de los objetos que sirven de manejo del reproductor e interactuación con el usuario se han especificado ambas instancias con sus correspondientes textos, todos ellos incluidos en la instancia de creación de los botones en el módulo *videoPlayer*, ***generateButtons***.

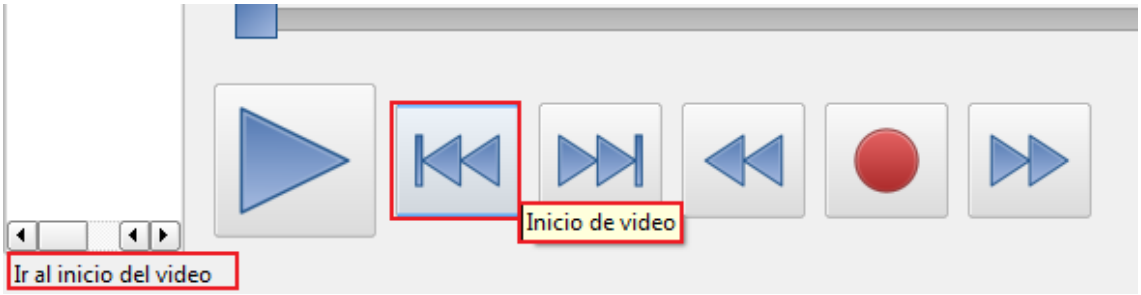
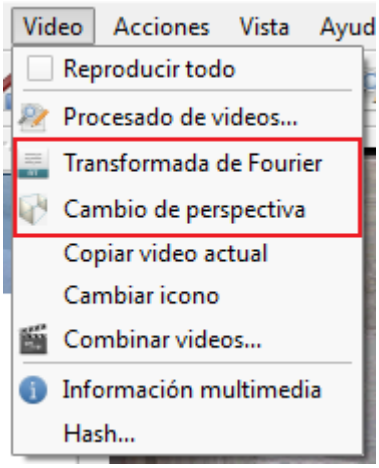


Figura 6.3: Descriptor en barra de estado para los iconos de reproductor de video

6.3. Iconos de acceso directo



En la barra de herramientas de la aplicación y menú desplegable 'Video' se han añadido los enlaces con los iconos correspondientes a los dos nuevos procesamientos de video más importantes, transformada de Fourier y cambio de perspectiva, para lo que se ha utilizado un recurso anteriormente también usado en la creación del archivo de carga de traducciones, que nos proporciona la posibilidad de utilizar fuentes externas a python dentro de la aplicación a través de una ruto o referencia.

Si en la anterior ocasión usamos el comando *$~ pylupdate4* para dicho cometido, en esta ocasión primero debemos crear un archivo *.qrc donde se especifique el nombre del archivo imagen que define el icono y su ruta, además de cómo vamos a referenciarla posteriormente dentro de la aplicación para poder utilizarla. Ahora sólo nos queda ejecutar el comando *$~pyrcc4 *.qrc –o *.py* que nos servirá para la creación del script *.py que debemos integrar en los archivos de nuestra aplicación y que servirá de carga de los iconos. En nuestro caso hemos nombrado este archivo como *newIcons_rc.py*, y dentro del archivo *icons.py* referenciamos ambos iconos a través de las constantes *ICON_FFT* y *ICON_PERSP* que llamaremos desde el módulo *Forevid* e incluirlas en la ventana principal como se aprecia a continuación.

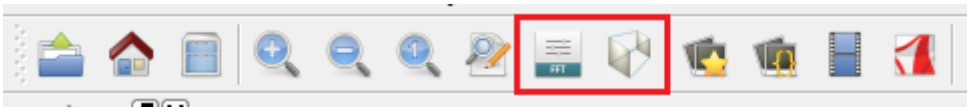


Figura 6.4: Iconos añadidos en la barra de herramientas

Asociamos a estos iconos las acciones que deben realizar, o realmente instancias a ejecutar dentro del módulo principal de Forevid, cuando son seleccionadas. Éstas no son otras que las de llamar a los módulos correspondientes para cada filtro que se detallará más adelante en los puntos 6.5.9 (cambio de perspectiva) y 6.5.10 (DFT), pasándoles como parámetro el clip de video seleccionado y la cadena de filtros Avisynth que hayamos podido incluir anteriormente como procesado previo a estos filtros como principales argumentos, todos ellos necesarios para posteriormente ser enviados al módulo pyavs.py en caso de ser aceptados los procesamientos de video deseados.

6.4. Editar y Eliminar cualquier filtro añadido

Otros de las limitaciones detectadas en la aplicación a la hora de trabajar con el procesamiento de video sobre los clips importados, era que no se permitía eliminar o modificar los filtros previos al último, lo que se convertía en un problema mayor a medida que se introducían más filtros en la cadena y se deseaba realizar alguna acción sobre los primeros añadidos. Esto restaba enormemente la flexibilidad de la aplicación en lo que debe ser una de sus principales funcionalidades como es la edición y análisis de video.

Inicialmente se centraron los esfuerzos en conocer la estructura y organización de estos filtros que se aplicaban para comprender por qué se limitó de forma explícita, y si dicha razón se podría evitar o hallar una solución que tuviese como solución final poder realizar las acciones que se permitían sobre el último filtro añadido en cualquiera de ellos independientemente de su posición.

De inicio, se crea una instancia en *pyavs.py*, *printFiltersChain*, para poder visualizar durante la ejecución de la aplicación los filtros de la cadena, observando que se incluyen posibles conversiones del espacio de colores necesarias para aplicar el siguiente filtro. Este filtro oculto en la interfaz de manejo de Forevid provoca que se decidiera bloquear acciones sobre filtros previos al último y únicamente analizar si previo al último filtro ha sido necesario añadir otro para convertir el formato de espacio de colores y eliminarlo en caso afirmativo. Otro de los problemas iniciales surgidos fue la devolución del número que indica la posición del filtro dentro de la cadena de filtros por la API, ya que en la lista que se muestra, como ya se ha dicho, no se corresponde con todos los filtro reales que alberga la cadena, dando lugar a error en los índices devueltos desde la ventada de procesamiento de videos.

6.4.1. Obtención del índice real

Por el motivo antes mencionado de la introducción de filtros “ocultos” que sirven para la conversión del espacio de colores, se debe implementar una instancia que nos permita obtener el índice real del filtro que tenemos seleccionado en el cuadro de procesado de vídeo teniendo en cuenta la conversión de colores. Pongamos un ejemplo, si añadimos los siguientes filtros desde Forevid: DCT, Difuminar y Convolución, en este orden, los índices que devolverá el widget desde la ventana será:

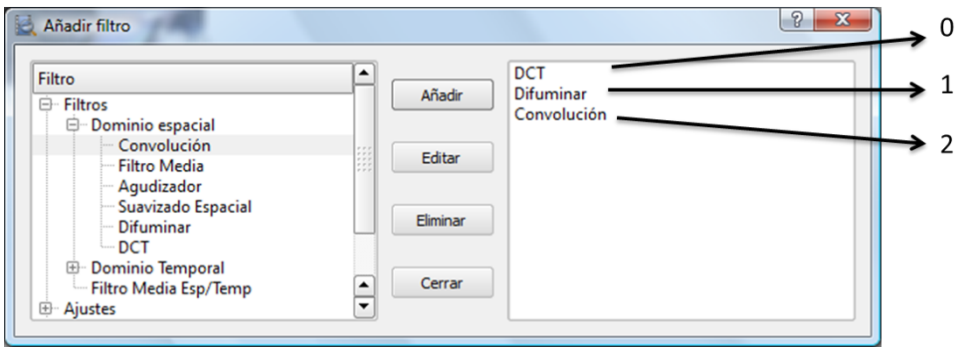


Figura 6.5: Índices según la API de cada elemento

Sin embargo, los índices reales para cada uno de ellos será otro, descubriendo todos los procesamientos de video que se producen:

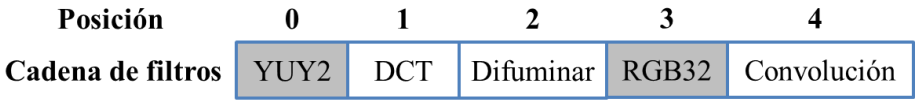


Figura 6.6: Índices reales en el array de filtros

Esto provoca una descoordinación entre la interfaz y Avisynth que se ha resuelto mediante la implementación de la instancia ***realIndexFilter*** que haya el índice real del filtro seleccionado dentro del array, y no el índice que se obtiene al seleccionarlo desde la interfaz de ventana de filtros. Con esta instancia devolvemos el índice correcto dentro de la cadena de filtros pasados a pyavs.py (*self.filterChain*), ignorando los posibles filtros de conversión del espacio de colores que pudiese haber antes del filtro seleccionado en la interfaz de la ventana de filtros.

Dentro de este mismo módulo se debe editar otras instancias relacionadas tanto con la edición como la eliminación de algún filtro: *editFilter* y *removeFilter*, pasándole como parámetro el número de dicho índice que le indica cuál será la posición del filtro a modificar.

6.4.2. Eliminación de espacio de colores indebido

Tras la modificación anterior observamos un problema secundario derivado del funcionamiento establecido y limitado inicial de la aplicación del que no se era consciente. Al poder eliminar cualquier filtro, únicamente se comprobaba si el filtro anterior al mismo era una conversión de espacio de colores, y en caso de que así fuese, se eliminaba también. Esto no trae ningún problema si el filtro analizado es siempre el último de la cadena, en cambio provoca problemas de espacios de colores si el siguiente filtro al eliminado necesita de la conversión de colores previamente realizada. Lo explicamos con el ejemplo anterior:

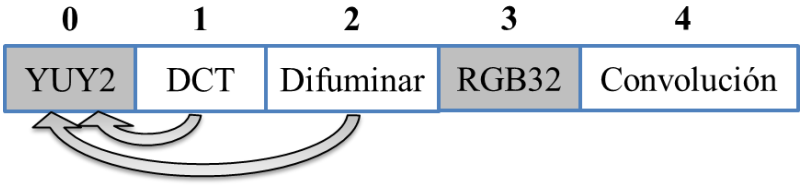


Figura 6.7: Recursión de filtros a la conversión del espacio de colores

Como vemos, tanto DCT como Difuminar necesitan del espacio de colores YUY2 para ejecutarse, por lo tanto, si deseamos eliminar DCT se eliminará también YUY2, dejando a Difuminar sin el filtro necesario y provocando errores al aplicar la cadena sobre el video fuente:

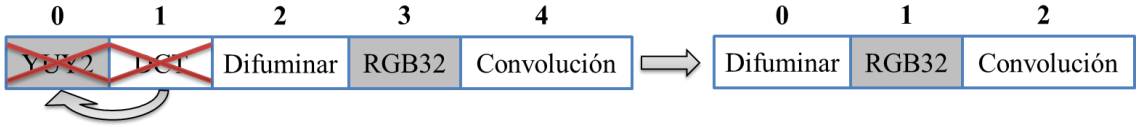


Figura 6.8: Problemática de eliminación de filtro precedido de espacio de colores

Por ello se ha ampliado el algoritmo de eliminación comprobando que ningún otro posterior necesita la conversión de colores situada antes, y únicamente en el caso de ser así eliminarlo.

6.4.3. Cambios de la UI

Completando la mejora había que cambiar ciertas cosas en la capa de la interfaz de usuario que estaban establecidas desde el módulo *filterWindow.py*, como por ejemplo la deshabilitación de los botones editar y eliminar cuando no se seleccionaba el último filtro añadido, realizada desde la instancia ***clickedB***, o incluir las instancias creadas de *pyavs.py* que se han descrito anteriormente:

- En ***editFilter*** y ***removeFilter*** usamos la instancia ***realIndexFilter*** para obtener el índice correcto.
- En la instancia que carga los valores de los parámetros (***updateValues***) de la clase ***parameterWindow*** hemos introducido el parámetro ***editing*** que indica el número de posición dentro de la cadena a la función para crear el script Avisynth temporal antes de aceptar los datos de la ventana.

Además de esto, y para evitar duplicación de procesados de video en un mismo clip, se ha restringido desde la capa de la UI la introducción del mismo filtro en más de una ocasión, mostrando un popup informativo en caso de que esto sucediese:

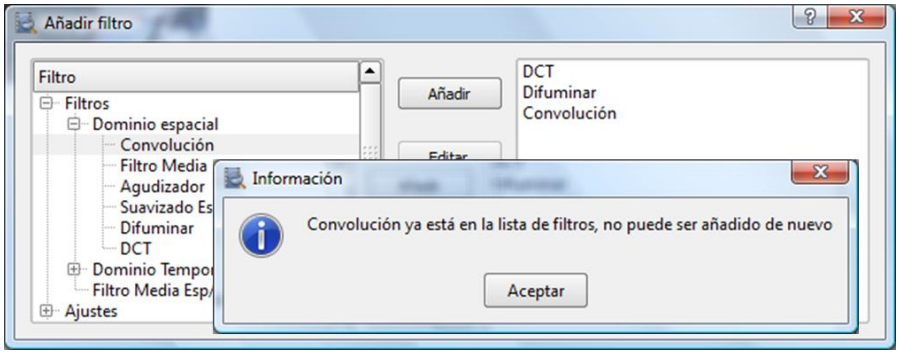


Figura 6.9: Pop-up informativo para evitar duplicación de filtros

6.5. Filtros añadidos

Vamos a listar los plugins añadidos de Avisynth que se han recopilado para esta aplicación, en su mayoría filtros internos que no disponían de la interfaz de usuario necesaria. Por lo que el principal trabajo se centra en la comunicación entre el cuadro de diálogo creado y el paso de estos parámetros introducidos a los plugins que los interpretan y utilizan. Siguiendo la estructura implementada en la aplicación, fue necesario antes implementar nuevas clases de parámetros que posteriormente se han incluido como parte de los filtros:

- **floatSpinBox**: Se utiliza para introducir los valores de la función DCTFilter
- **matrixBox**: Conjunto de 5x5 spinBoxs para introducir valores enteros que indican el valor de la matriz a convolucionar en el filtro Convolution.

- **sliderPainter**: Se crean para la función Perspective, ya que, a diferencia de los sliders que ya existían, están vinculados a una acción de pintado y actualización del widget gráfico mostrado en la misma ventana.
- **fft_filter**: Por necesidad hubo que implementar estos deslizadores para la ventana de filtrado en frecuencia, uno para valores enteros y otro para decimales, similares a los ya existentes en la versión original de la aplicación.

Vamos entonces a describir cada uno de los filtros con detalle.

6.5.1. Convolución

Se trata de un filtro incluido en la librería interna de Avisynth (GeneralConvolution[18]) que se consideró de utilidad para comprobar los resultados al realiza una convolución en el tiempo sobre una imagen tomando una matriz que el usuario a diseñado. Para ello este filtro proporcionaba la posibilidad de:

- Matriz de convolución de tamaño 3x3 o 5x5 con valores enteros en cada índice entre -255 y 255. Estos valores se facilitan con una cadena de texto de 9 o 25 números enteros, por lo que hubo que implementar instancias de conversión entre los valores enteros recogidos en los spinBox para darles el formato requerido.
- Ajustar la intensidad de la imagen salida en casos en los que la matriz utilizada para la convolución produzca una reducción de brillo sumando un valor a cada pixel. Después de sumar o restar este valor a cada uno de los brillos de la imagen, se recortará el valor a 0 en caso de obtener un resultado menos que este, o a 255 si es superior.
- Introducir un divisor que afecta a toda la matriz. Este valor habitualmente es la suma de todos los índices de la matriz para impedir variaciones o modificaciones del valor brillo medio de la imagen.

En los bordes de la imagen se ha utilizado la repetición de pixeles para de la imagen para poder completar la convolución de la matriz por todas las posiciones como se detalla en el apartado 2.4.1 de este documento. Este plugin únicamente se efectúa en el espacio de colores RGB32.

La ventana que se diseñó para el usuario, al igual que el resto de ventanas, se basó en las ya integradas siguiendo una plantilla similar que el usuario considerase propia a la aplicación y reconociese. El clip se muestra en la parte izquierda de la ventana y a la derecha los parámetros a introducir:

[18] http://avisynth.nl/index.php/GeneralConvolution

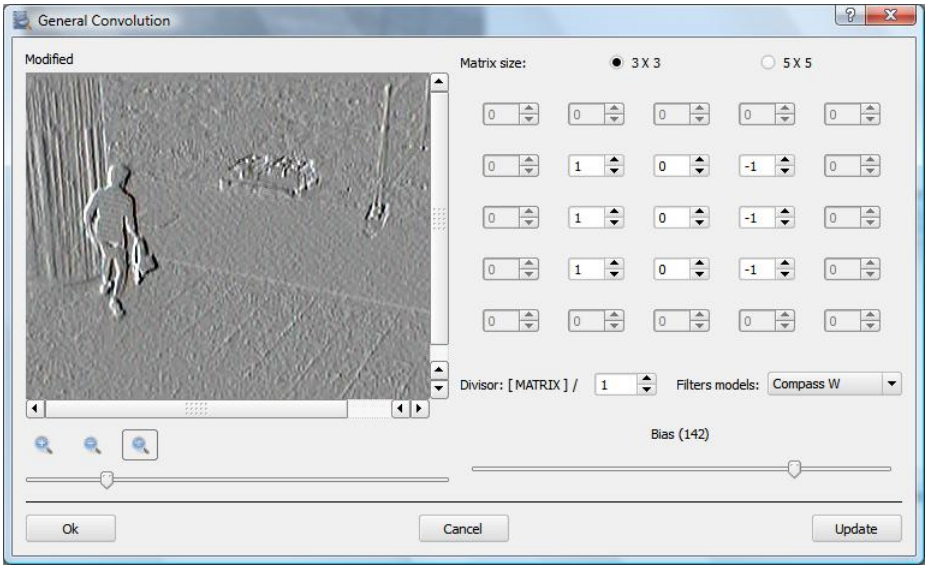


Figura 6.10: Ventana filtro convolución espacial 1

Se ha introducido la posibilidad de seleccionar el tamaño de la matriz evitando cálculos innecesarios que incrementen el tiempo de procesamiento, asignando 9 valores a la matriz en caso de seleccionar 3x3 e ignorando el resto.

Ya que hay muchas matrices típicas utilizadas en el procesamiento de video, se ha creado una caja que permite seleccionar alguna de éstas cargando automáticamente los valores correspondientes:

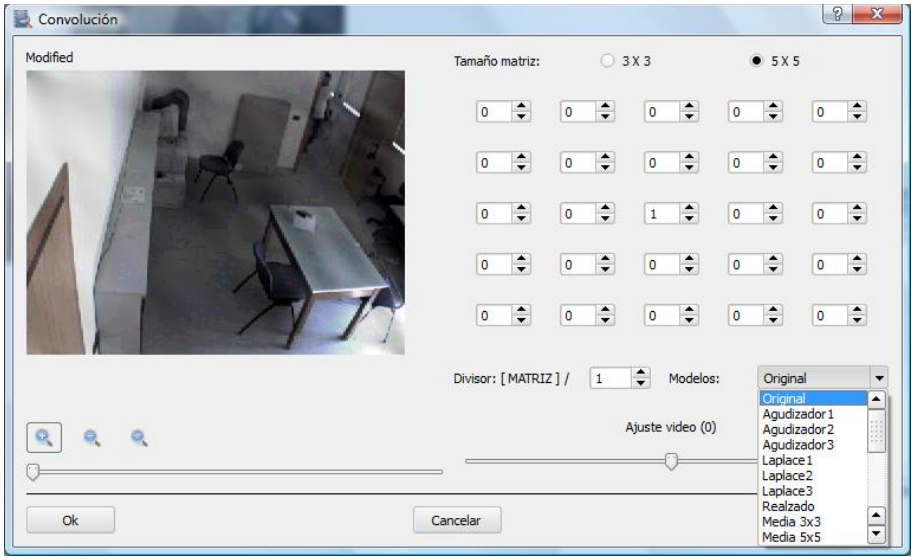


Figura 6.11: Ventana filtro convolución espacial 2

6.5.2. Filtro de media adaptativa

Este filtro[19] externo es similar al filtro de media explicado en el apartado 2.4.2.1 con algunas modificaciones que mejoran su comportamiento, aunque su objetivo es el mismo: tratar de eliminar el ruido de tipo impulsivo (salt&peeper) usando estadísticas

[19] http://www.avisynth.nl/users/vcmohan/AdaptiveMedian/AdaptiveMedian.html#

locales dentro del kernel o matriz, de un tamaño determinado seleccionado por el usuario, que va recorriendo la imagen completa.

Para ello se calcula en primera instancia el mínimo (MIN), máximo (MAX) y media (MED) de los valores situados dentro de la rejilla, evaluando:

1. Si la media es mayor que el mínimo y menor que el máximo:
    a. El valor central de la rejilla se mantiene igual en caso de que su valor sea también mayor que MIN e inferior a MAX.
    b. En caso contrario, el valor central es sustituido por MED
2. Si la condición 1. no se cumple es debido a que la matriz se encuentra en los bordes de la imagen, por lo que se ignora el proceso y pasa a la siguiente posición de la imagen.

Con estos sencillos pasos de evaluación se consigue respetar los detalles de la imagen y presentar mejores resultados que un filtro de media habitual cuando la densidad de ruido impulsivo es alta, además de suavizar otro tipo de efectos de ruido no impulsivo o reducir la distorsión excesiva de adelgazamiento o engrosamiento de los bordes de los objetos.

Aunque este filtro puede trabajar sobre los tres espacios de colores: RGB32, YUY2 e YV12, se ha limitado su uso para únicamente funcionar sobre el primero de ellos, ya que de esta forma el filtrado se ejecuta sobre los tres planos de colores de forma automática.

6.5.3. Transformada discreta del coseno, DCT

El DCT filter[20] realiza la transformación para bloques de 8x8 de la imagen a través de una librería externa aplicando la teoría señalada en 2.5, obteniendo una matriz del mismo tamaño con los valores de los coeficientes de esta transformada, y posteriormente multiplicándolos por los valores de escala que le proporciona el usuario elemento a elemento. Para completar el proceso se realiza la IDCT devolviendo la imagen al dominio espacial.

Para introducir los valores de escala que determinaran los coeficientes, y por lo tanto peso final de los rangos de frecuencia, de la matriz es necesario introducir 8 números decimales que darán lugar a los 64 coeficientes, ¿cómo? posicionando los arrays en una columna y una fila, juntamos los C0 de ambos elementos creando la esquina superior izquierda de la matriz, y multiplicando los coeficientes correspondientes para cada posición de la matriz. Gráficamente es más sencillo de comprender el mecanismo utilizado para evitar introducir cada uno de los 64 valores que serían necesarios:

[20] http://avisynth.org.ru/docs/english/externalfilters/dctfilter.htm

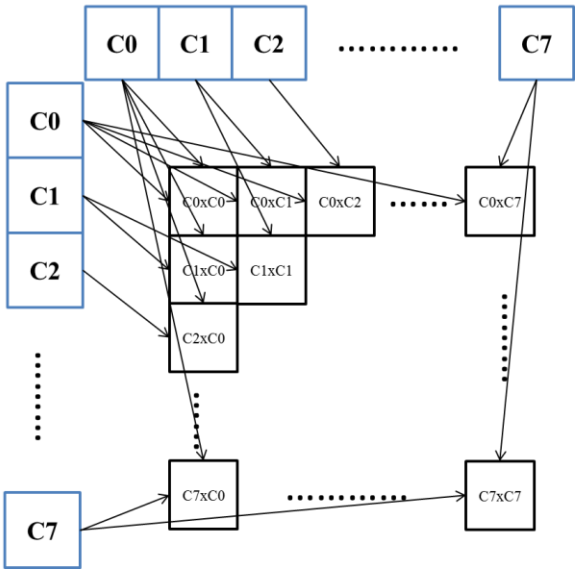


Figura 6.12: Cálculo de coeficientes para DCT

Los coeficientes de baja frecuencia se sitúan en la parte superior izquierda de la matriz, y por lo tanto el valor de componente continua o DC es la posición V(0,0), la cual debe mantenerse a 1. A media que nos desplazamos a la parte inferior-derecha de la matriz los valores se asocian a frecuencias mayores y por lo tanto tendrán menos efecto de pérdida de información en la imagen si sus valores son 0 o cercanos a 0.

Todos los valores de escala están comprendidos entre 0 y 1, y aunque la resolución de bits para cada uno de ellos es de 3 bits y por lo tanto los valores a todo efecto serán de: 0, 1/8, 2/8, 3/8,…1, es posible introducir cualquier valor decimal que será redondeado al más cercano de los posibles.

La interfaz de usuario se ha implementado mediante spinBoxs que recogen el valor para cada uno de los coeficientes excepto para el valor C0_DC, el cual se ha bloqueado y mantenido a 1 de forma necesaria:

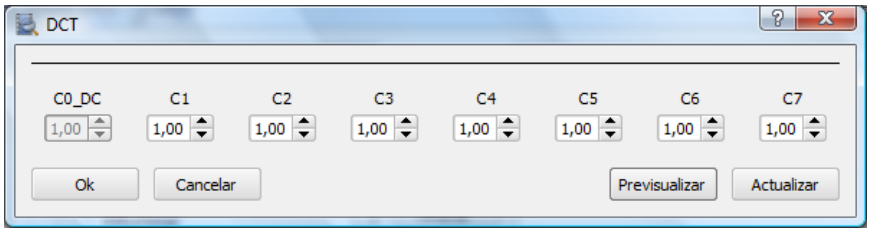


Figura 6.13: Ventana de DCT

6.5.4. Ajustes de video

Incluído como uno de los filtros internos de Avisynth[21], se utiliza como ajuste de parámetros generales de la imagen dentro del espacio de colores YUV de Avisytnh que permite la aplicación, tales como:

[21] http://avisynth.nl/index.php/Tweak

- Tono: Ajusta el color de la imagen otorgando mayor peso a los canales de crominancia dependiendo del valor del parámetro, aceptando desde +180 hasta -180. Cuando se ajustan valores positivos dentro de este rango el canal U adquiere mayor importancia entre los demás canales de luminancia y V. Si son negativos, en cambio, V toma mayor peso en deprimento de Y y de U.
- Saturación: Su valor oscila entre el 0 y el 10, variando el peso de los canales Y, U y V en función del valor del parámetro. Se establece el 1 como valor por defecto e inalterable de la imagen, y a medida que su valor asciende por encima de este cobra mayor importancia los canales U y V que trasportan el color de la imagen. Si por el contrario se establece un valor por debajo de 1 el canal Y o luminancia tiene mayor peso. Para el valor 0, la imagen se muestra en una escala de grises.
- Brillo: Suma su valor a cada uno de los brillos de los pixeles de los tres canales, recortando a 0 en caso de que dicha suma sea negativa, o a 255 si sobrepasa este valor. Permite valores entre -255 y +255.
- Contraste: Varía el canal Y de luminancia, ampliando el rango o diferencia entre el máximo y el mínimo de dicho canal cuando este valor es mayor de 1 y hasta 10, y estrechándolo cuando está entre 0 y 1.

Para la interfaz de la ventana se utilizaron objetos ya creados, específicamente deslizadores de tipo decimal, ajustando los valores máximos y mínimos permitidos para cada uno de los parámetros descritos anteriormente:

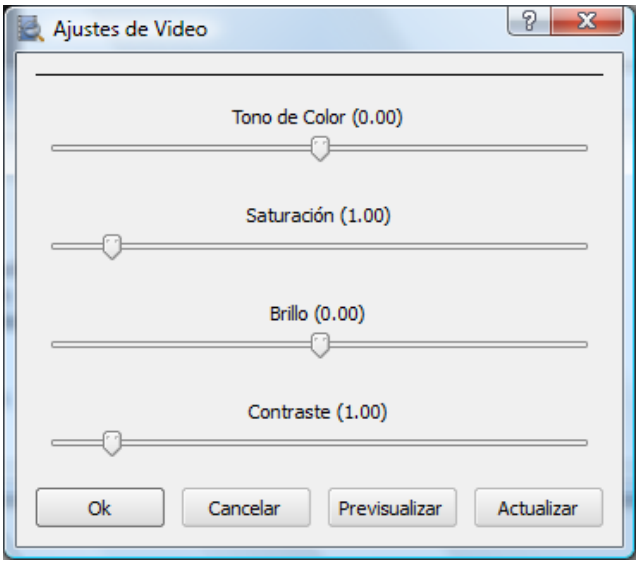


Figura 6.14: Ventana Ajustes generales de video

6.5.5. Ajustes YUV

Su cometido es similar al ajusto de video descrito en el punto anterior, aunque éste trabaja sobre cada canal de manera más específica e individual. Este filtro[22] pertenece a los filtros internos de Avisynth y, evidentemente, funciona en el espacio de colores YUY2 o YV12. Los posibles ajustes que se pueden efectuar sobre cada canal son:

[22] http://avisynth.nl/index.php/ColorYUV

- Ganancia: Factor de multiplicación aplicado a los valores de brillo de dicho canal, de manera que se siga la relación siguiente, poniendo de ejemplo el canal de luminancia:

$$\begin{matrix} G = k * 256 \\ Y' = Y(k+1) \end{matrix} \rightarrow Y' = Y(\frac{G}{256} + 1)$$

Vemos que el valor de G permite el ajuste muy fino del canal permitiendo valores amplios, a diferencia de las técnicas que habitualmente se utilizan de aplicar directamente el factor de multiplicación sobre los brillos. Vemos algunos ejemplos: G=0; Y'=Y, G=256; Y'=2*Y, o G=512; Y'=3*Y. Se ha habilitado un rango de G entre 0 y 1000. Esta fórmula se aplica de la misma forma para U y V.

- Valor añadido: o de offset, que se suma a los brillos del canal de forma directa. Permite introducir tanto valores positivos como negativos con un rango de 128 valores, desde -64 hasta +64.
- Gamma: Ajusta el gamma del canal de luminancia pero no tiene efecto sobre U y V, por lo que se ha deshabilitado el deslizador para estos planos. Su rango es de -256 a +256, siendo el mínimo (-256) equivalente a gamma'=0, y el máximo (+256) igual a gamma'=2gamma.
- Contraste: Expande y aumenta los niveles de brillo del canal de forma similar a ganancia, pero el nivel final de salida es: $Y' = Y + k(Y - 128)$, por lo que:

$$Y' = Y + \frac{C}{256}(Y - 128)$$

Se ha establecido un rango desde 0 hasta 1000.

Para el acceso al usuario de este filtro se ha utilizado un objeto de la API que hasta ahora no se había visto, *QTabWidget*, organizando en pestañas los ajustes para cada canal de manera que quede ordenado y sin saturar las opciones que se presentan a la vez. En cada pestaña se han añadido los ya explicados deslizadores de tipo decimal utilizando el código que se nos proporcionaba desde el inicio. Estos son los valores por defecto de los parámetros para la luminancia:

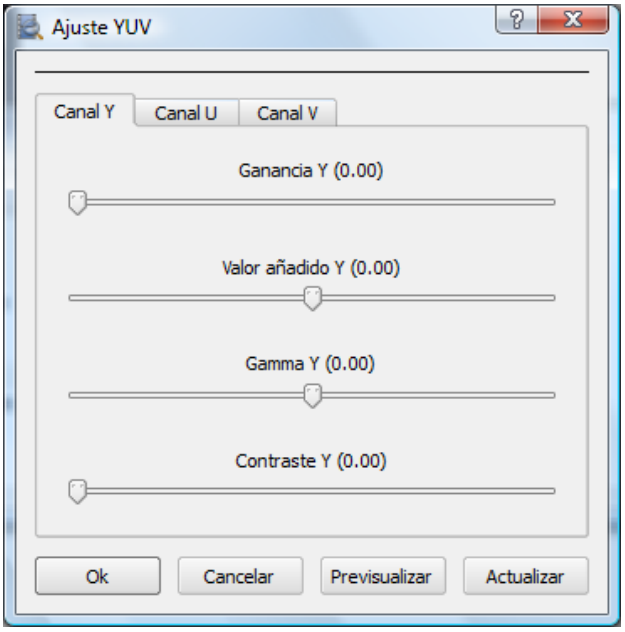


Figura 6.15: Ventana Ajuste YUV

6.5.6. Ajustes RGB

Ya que podemos encontrarnos varios espacios de colores con los que trabajar, este filtro también perteneciente a la librería interna Avisynth, ajusta los mismos parámetros que la función anterior exceptuando el contraste, pero en el espacio de colores RGB32: ganancia, offset y gamma. Como se dijo en el punto 4.2.3., RGB32 define un canal alpha extra aparte de los canales rojo, azul y verde para completar cada palabra de pixel con un total de 32 bits, 8 por canal, aunque no permite modificaciones del mismo por lo que no se ha integrado dentro de las opciones disponibles para el usuario.

Los rangos de valores que se pueden seleccionar no son los mismos que en el punto anterior, por ejemplo, la ganancia admite valores entre 0 y 10, pero su valor se utiliza directamente como factor de escala multiplicador por lo que 1 no produce variación en la imagen, y 0 elimina el canal del conjunto poniendo a este valor todos los brillos.

El valor de offset admite un rango desde -32 hasta 32, y para gamma se introduce el valor exponencial en la función.

Se utiliza la misma herramienta para organizar visualmente los parámetros por canales:

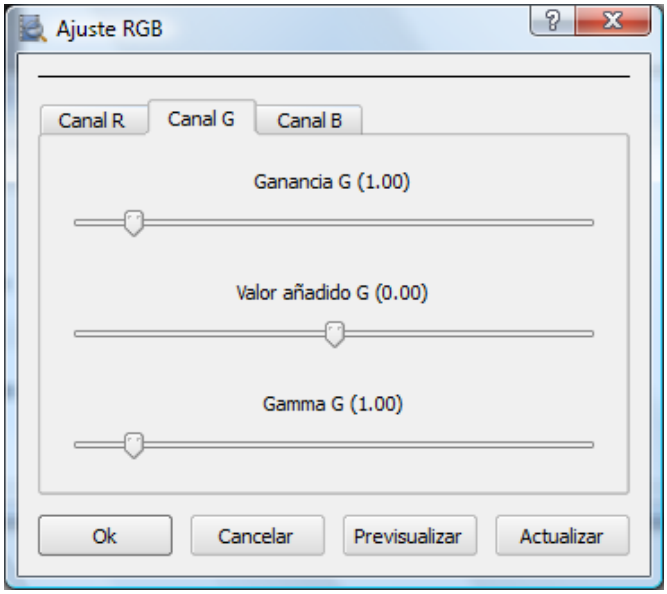


Figura 6.16: Ventana Ajuste RGB

6.5.7. Rotación

A pesar de que se ofrecían transformaciones geométricas de este tipo realizando giros de 90º hacia derecha e izquierda, se quiso incluir este plugin con la posibilidad de seleccionar el ángulo concreto de giro entre los 0 y los 360º. El tamaño del video de salida total es el mismo que el original, lo que provoca recortes en las esquinas del video perdiendo información de imagen, ya que no se analiza el tamaño horizontal del video una vez que se ha rotado, siendo éste siempre mayor. Se utiliza una interpolación lineal para adaptar el efecto de rotación a la imagen de salida y respetar el tamaño a escala de la imagen y relación de aspecto original.

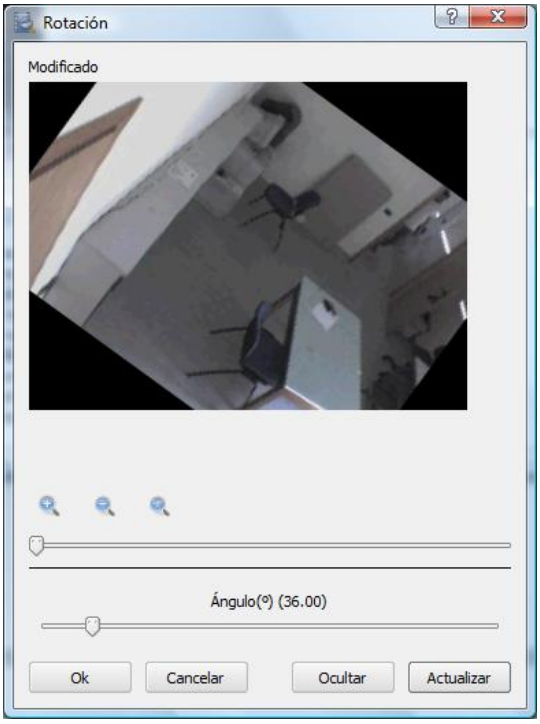


Figura 6.17: Ventana Rotación

6.5.8. Características del clip de video

En ocasiones puede resultar útil conocer toda la información acerca del clip tras realizar diversas modificaciones. Para ello sirve este filtro, que nos muestra información general del mismo incrustada en la parte superior izquierda del video. Hay que reseñar que la información del mismo se muestra hasta la situación anterior a la aplicación de este filtro, por lo que si posteriormente se añaden otros que hacen modificar algunos de los parámetros, éstos no serán los reales del video evidentemente:

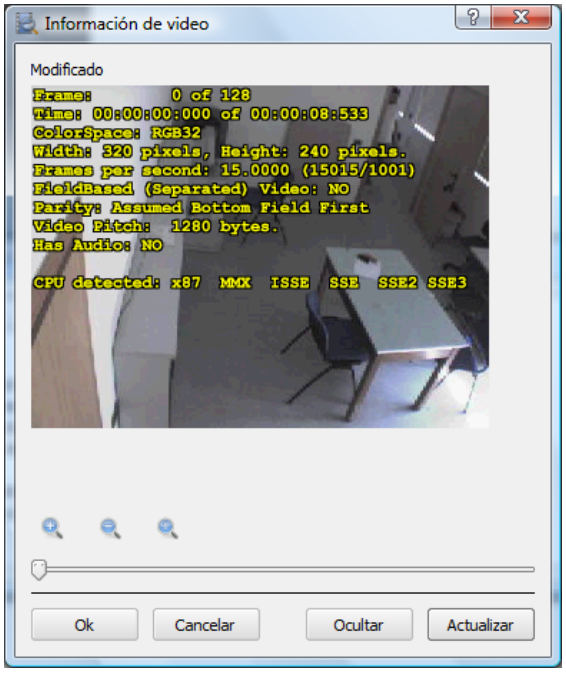


Figura 6.18: Ventana Información de video

6.5.9. Cambio de perspectiva

Éste es el primero de los dos filtros que se ha tratado de forma especial al crearse un acceso desde la ventana principal y aplicándolo al video en un modo aparte de la cadena de filtros que se aplica desde el procesamiento de video. Esta transformación geométrica es compleja y requiere de una interacción con el usuario directa y gráfica para que se sepa en todo momento cuál va a ser el resultado aproximado final incluso antes de aplicarlo. Los plugins externos que facilitan esta transformación se basan en las coordenadas pasadas como parámetros dentro de la imagen, tomando el origen la esquina superior izquierda del video. Son cuatro puntos que determinan la perspectiva de la imagen y el ángulo del observador virtual, pudiendo transformar videos de captura fija con un ángulo o posición incorrecta en otro donde los detalles que interesan sean mostrados desde una observación más directa.

Hay dos maneras de representar los cambios de perspectiva, de una forma rectangular a cuadrilátera, o viceversa. De forma que se pueda entender, la primera de ellas marca los cuatro puntos posicionados de referencia e interpola la imagen completa para que sus esquinas coincidan con los mismos, completando el resto con un fondo negro que mantiene las dimensiones originales del clip de video. Para la segunda el proceso es "inverso", desplazando las coordenadas establecidas hacia las esquinas de las dimensiones originales del video, realizando un efecto "zoom" y de deformación de la imagen más notable y, quizás, menos intuitivo.

Muchos de los plugins analizados traían ambos procesamientos de video, pero incurrían en errores de cálculo a la hora de aplicar uno u otro, por lo que finalmente se decidió el uso de un filtro distinto para cada tipo de procesamiento. Para el primero, el nombre del filtro es Reform[23], y el de la función *skew*, incluida dentro de la propia librería. *Deskew* era otra de las funciones que incluía para el proceso de cuadrilátero a rectangular, sin embargo tras varias pruebas se descartó por su pésimo efecto. Reformer[24] en cambio poseía la función *q2r* que mejoraba con diferencia esta transformación respecto de *deskew*, y ya que la otra función dentro de Reformer, *r2q*, empeoraba los resultados de *skew*, se decidió la inclusión de ambos:

	Librería	
	Reform	**Reformer**
Rectangular a cuadrilátero	**skew**	r2q
Cuadrilátero a rectangular	deskew	**q2r**

La similitud de parámetros requeridos, e incluso su orden, facilitaron la combinación entre ambas funciones, totalmente trasparente para el usuario. Esto se debe a que el

[23] http://www.avisynth.nl/users/vcmohan/Reform/Reform.html
[24] http://www.avisynth.nl/users/vcmohan/Reformer/Reformer.html

autor de ambos procesamientos es el mismo, ya que de no ser así la forma de trabajar entre ambos no tiene porqué parecerse en tantos aspectos.

El ajuste de estas cuatro coordenadas bidimensionales implica introducir ocho parámetros, dos por coordenada: x e y. Hay que tener en cuenta que las dimensiones de cada clip de video es una característica propia, hecho importante si tenemos en cuenta que cada coordenada se tiene que encuadrar dentro de una porción de la imagen total, dividiendo por la mitad la imagen tanto verticalmente como horizontalmente resultando cuatro partes, una para cada punto:

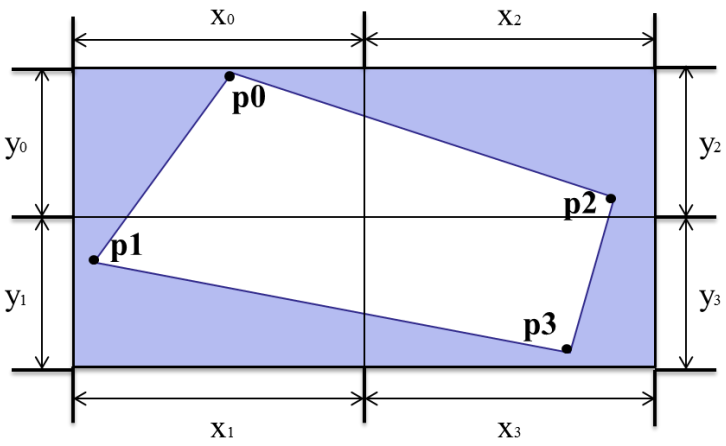


Figura 6.19: Parcelas de división para cada punto

Los valores de las coordenadas de cada uno de ellos por lo tanto quedan delimitados para garantizar un buen funcionamiento del filtro y resultados coherentes.

La ubicación de las coordenadas se determina tomando la posición de la imagen x,y=0,0 como origen de coordenadas como se ha comentado anteriormente. Para recoger el valor de los mismos se ha diseñado un widget gráfico que simula la imagen, pero de un tamaño determinado e invariante. Alrededor del mismo 8 deslizadores que sirven para ilustrar el resultado final en el widget, y el valor de la coordenada previo escalado entre el valor recogido por el deslizador y el que realmente se debe asignar a la función teniendo en cuenta las dimensiones reales de la imagen original del clip de video.

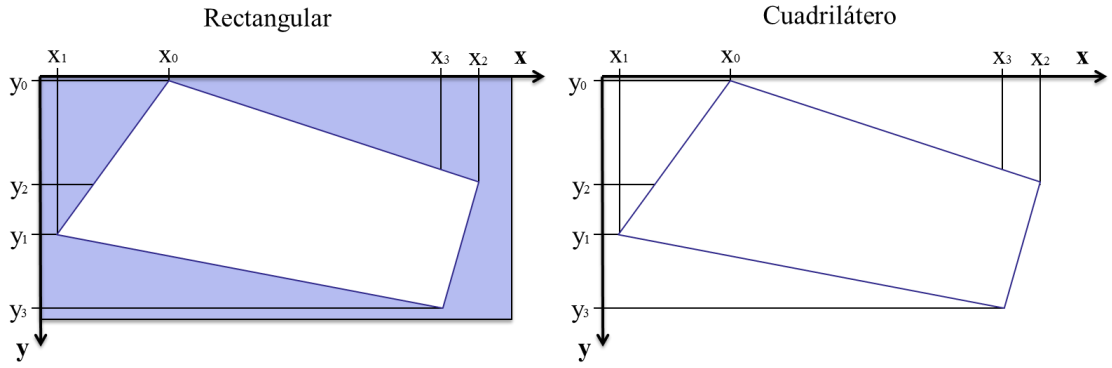


Figura 6.20: Obtención de las coordenadas para cada punto

Como vemos, para reconocer de forma mas sencilla el efecto de ambas funciones un checkbox, además de recoger la petición del usuario, provoca un cambio dentro del

dibujo de forma instantánea eliminando el fondo azul que indicaba el relleno que se añade al realizar el cambio de perspectiva en el video:

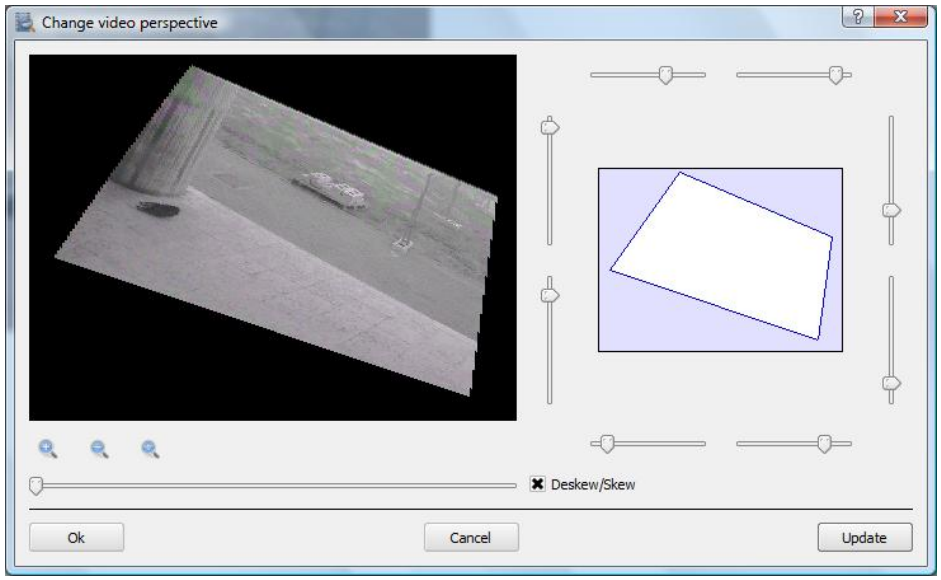


Figura 6.21: Efecto de pintado para transformación cuadrilátera

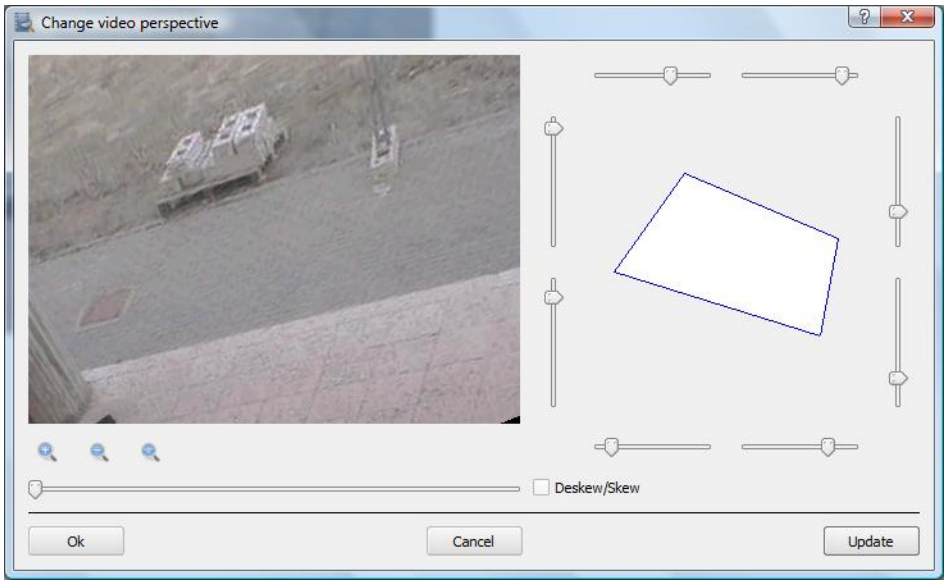


Figura 6.22: Efecto de pintado para transformación rectangular

El formato cuadrilateral en ciertas ocasiones produce errores mostrando zonas negras en parte de la imagen. Esto se debe al redimensionado e interpolación bilineal de la imagen realizado para el cambio de perspectiva según las posiciones de los parámetros seleccionados.

Con todo ello se considera una manera correcta de guiar al usuario en el momento de elegir la aplicación de este filtro.

6.5.10. Filtro en frecuencia

Para esta tarea se ha seleccionado una librería que permitiese el máximo número de posibilidades y flexibilidad que otorgasen al usuario una amplia gama de opciones

disponibles. El filtro F2Quiver[25] creada por V. C. Mohan aúna una gran cantidad de filtros en el dominio de la frecuencia que se ha procurado que sean accesibles al cien por cien por el usuario de forma sencilla e intuitiva.

En ocasiones una librería depende de funciones y cálculos más básicos que se realizan en otra librería, y en este caso eso es lo que ocurre con esta librería que necesita de las funciones que proporciona FFTW[26] para realizar la transformada de Fourier. Esto implica que debemos incluir estos archivos en alguna de las paths por defecto del sistema operativo, que en el caso de Windows para el entorno de trabajo donde se ha desarrollado el proyecto es: "*C:\windows\system32*".

Además de esto, y debido a que el filtrado en frecuencia únicamente se realiza para el canal Y de una imagen en el espacio de colores YUV, es imprescindible que antes de ejecutarlo se introduzca otro que realice la conversión del espacio de colores al citado anteriormente. Esto traerá varios problemas que se detallarán en la parte final dl documento, y ejecución de líneas extras dentro del módulo de ejecución principal desde donde se juntan y realizan las llamadas al resto de módulos que completan este filtro, *fftWindow.py*.

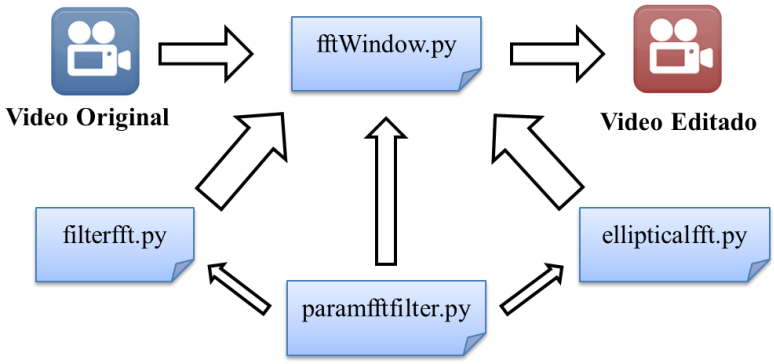


Figura 6.23: Esquema de llamadas de módulos para DFT

Como se observa la organización principal que se ha elaborado para la ejecución de este filtro podemos dividirla en dos partes, una primera para filtros simétricos de diferentes formas, tamaños y tipos que podemos introducir al clickar sobre "*Añadir filtro*", y otra parte que permite incluir hasta un máximo de dos filtros elípticos simétricos. Para cada una de estas partes se muestra en la parte central de la ventana principal las tablas correspondientes que informan al usuario sobre los filtros añadidos y que se aplican al actualizar los valores. Vamos a detallar cada una de estas partes por separado más adelante.

Existe una tercera parte, no tan crucial y más de complementación añadida, que permitía el filtro Avisynth para determinar si es necesario un re-escalado de la imagen modificada para impedir que se pierda resolución de brillos y quede con poco contraste, y otra que ajusta el valor de gamma del contraste de la imagen de la transformada de Fourier como se vio en el apartado 2.4.4.1. de teoría de este documento. Y es que este

[25] http://www.avisynth.nl/users/vcmohan/FFTQuiver/F2Quiver.htm
[26] http://www.fftw.org/

filtro permite visualizar a mitad de pantalla el espectro de frecuencia del video (e incluso la posición de esta información, si a la izquierda o a la derecha) por lo que se ha aprovechado para crear otro clip independiente del mostrado como resultado de las modificaciones aplicadas en el dominio real del espacio en la parte izquierda de la ventana, y coger la mitad de la imagen que nos proporciona el espectro de frecuencia y mostrarlo como información en tiempo real hacia el usuario en la parte derecha:

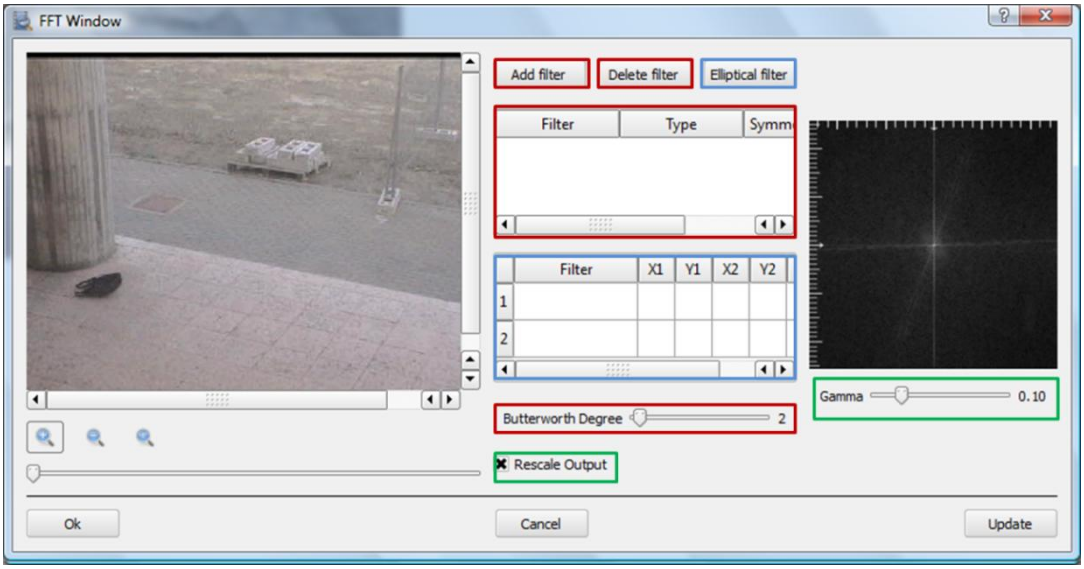


Figura 6.24: División de zonas en ventana DFT

6.5.10.1. Filtros FFT generales

Ya que ésta es la parte principal del filtro y de la que se espera mayores posibilidades y opciones, el número de parámetros que se introducen para definir cada uno de ellos es mayor al del resto y por lo tanto se debe tener claro qué significa cada uno y los valores que pueden tomar. Vamos a desarrollar cada uno de los parámetros y opciones que nos permiten de forma esquemática y a modo de resumen general gráficamente, ya que son varios y de distintos tipos:

Primer parámetro: string de entre 2 y 4 caracteres

- **1er carácter:** Tipo de atenuamiento
  - '**b**' → Butterworth
  - '**g**' → Gaussian
- **2ndo carácter:** Tipo de simetría
  - '**p**' → de punto → **3er carácter:** % ancho de banda, entre 0 y 9
  - '**l**' → lineal
  - '**v**' → vertical
  - '**h**' → horizontal
  - '**c**' → circular
  - • **3er carácter:** Tipo de filtro
    - '**h**' → filtro paso alto
    - '**l**' → filtro paso bajo
    - '**b**' → filtro paso banda
    - '**n**' → filtro ranura 'notch'

    4rto carácter: % ancho de banda, entre 0 y 9, **sólo para simetría lineal**

Segundo parámetro: número entero

	Simetría	Tipo de filtro
Frecuencia de corte	Vertical	Paso alto Paso bajo
	Horizontal	Paso alto Paso bajo
Frecuencia central	Circular	Paso banda Ranura ‘notch’
Frecuencia horizontal	De punto Lineal	

Tercer parámetro: número entero

	Simetría	Tipo de filtro
Ancho de Banda (BW) en %	Vertical	Paso alto Paso bajo
	Horizontal	Paso alto Paso bajo
	Circular	Paso banda Ranura ‘notch’
Frecuencia vertical	De punto Lineal	

Incluir el ancho de banda para los filtros de tipo paso bajo o alto no es un dato relevante que modifique en exceso el efecto final del filtrado, sin embargo sí puede ser más relevante para filtros paso banda o ranura ‘notch’. Ciertas combinaciones no se pueden ejecutar ni introducir como texto en el primer parámetro por razones obvias de contrasentido del tipo de filtro que se quiere y su simetría lineal (gll, glh, bll y blh). El siguiente parámetro sigue haciendo referencia al filtro especificado anterior, determinando el grado del mismo en caso de tratarse de un filtro con atenuamiento tipo Butterworth.

Mostramos una ventana que se muestra al usuario para añadir un filtro de este tipo divido en pestañas que seleccionan la simetría, y dentro de cada una de ellas todos los parámetros aplicables:

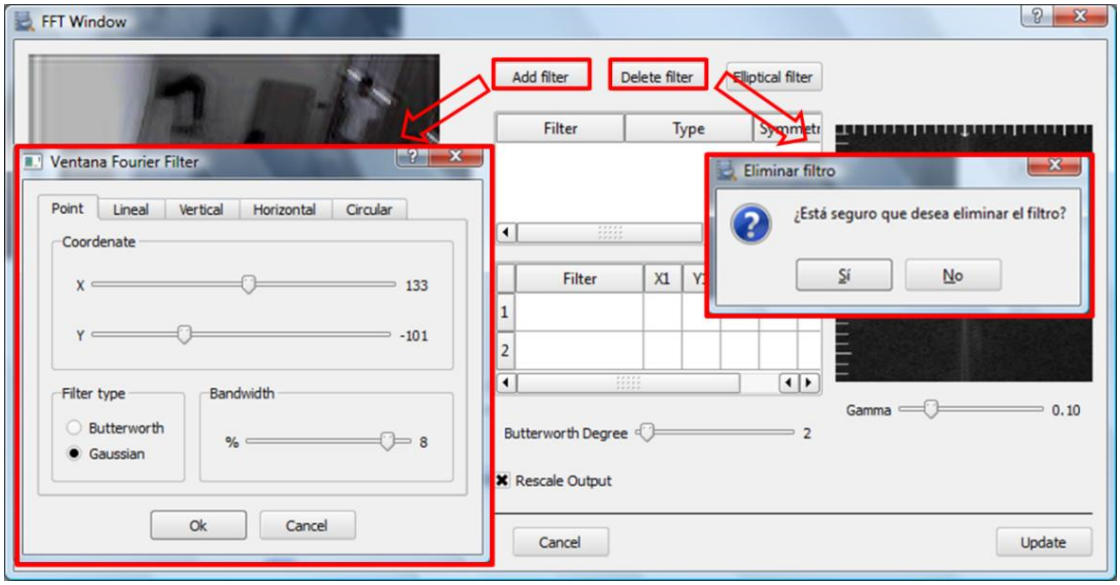


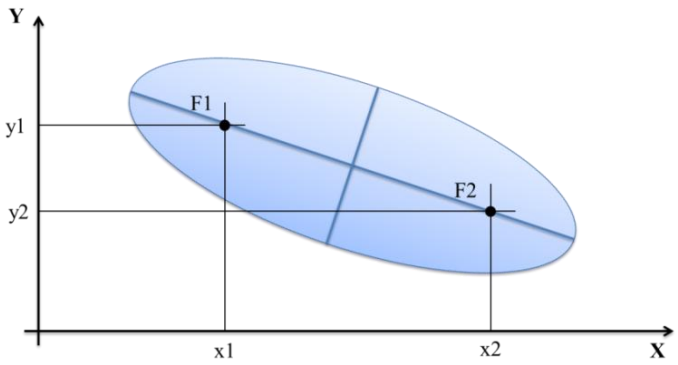
Figura 6.25: Ventanas emergentes para filtros generales

Todas las funcionalidades que controla esta ventana emergente se desarrollan a través del módulo *filterfft.py*, tanto el acceso al usuario a través de la ventana mostrada organizada en pestañas para cada una de las simetrías disponibles, como los parámetros (cadena de texto y otros valores enteros de BW y posición) a entregar al filtro finalmente una vez se aceptan los valores introducidos y se muestran en la tabla de la ventana principal.

Es posible incluir varios de estos filtros en cascada como en una cadena interna que procesa cada uno de los filtros añadidos, aunque el proceso se ralentiza a medida que se añaden más procesos al video original. Todos ellos se van listando en un array y mostrando en el cuadro superior de la zona central de la ventana. Para eliminar alguno de estos filtros sólo es preciso seleccionar con el ratón aquél que se desea y presionar el botón "*Eliminar filtro*".

6.5.10.2. Filtros FFT elípticos

Considerando este filtro como uno de los más importantes incluidos en cualquier aplicación que se desenvuelva en el terreno de la edición de video se ha decidido aprovechar al máximo todas las posibilidades que tiene esta librería en el filtrado frecuencial, incluyendo la posibilidad de añadir dos filtros elípticos simétricos independientes entre sí y del anterior, especificado a través de los parámetros de ubicación espacial necesarios para una elipsis en un plano de dos dimensiones:



Estos parámetros son accesibles y personalizables por el usuario a través de una ventana emergente mostrada al clickar el botón "*Filtros elípticos*" incluido en la ventana FFT, y que se ejecuta a través del módulo *ellipticalfft.py*. Además de definir la posición de la elipse y su forma con la ubicación de los dos focos de la misma a través de sus coordenadas x e y correspondientes, también podemos determinar el tamaño y nitidez con los parámetros *mf* y *sh*.

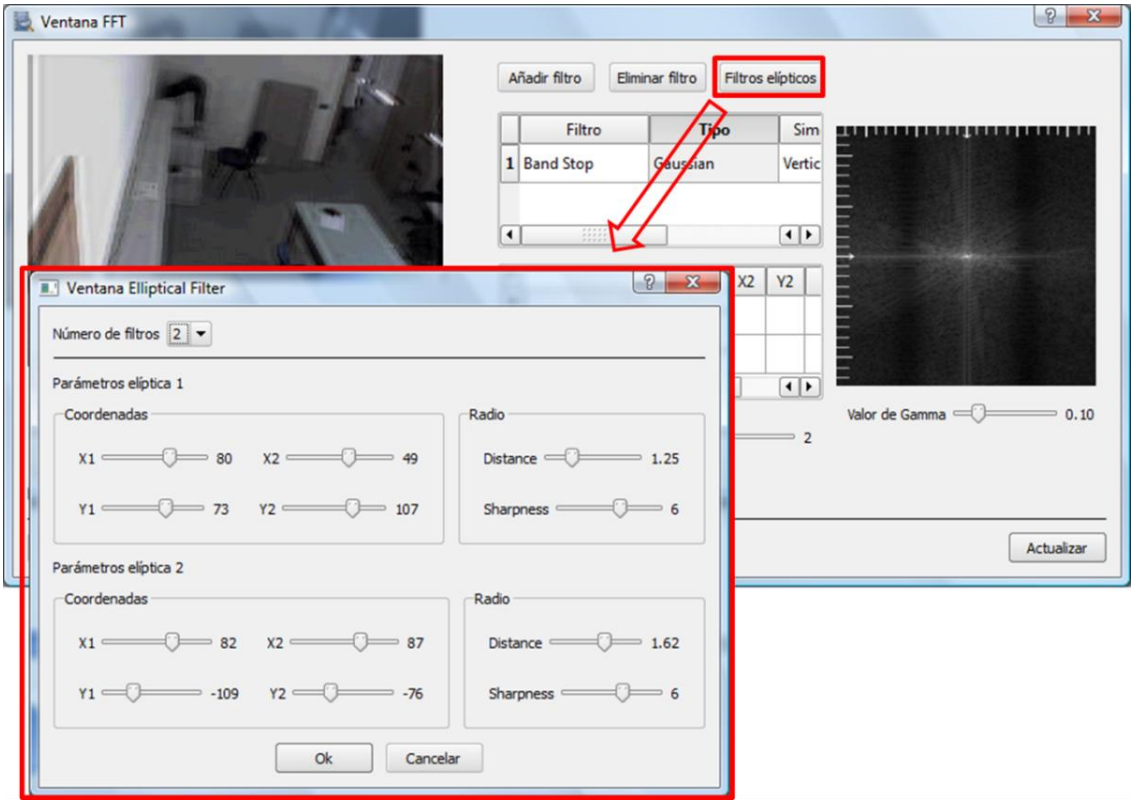


Figura 6.26: Ventana emergente para filtros elípticos

6.6. Redistribución organizativa de filtros

Una vez que se añadieron acciones de edición nuevas, se establecieron nuevos criterios de organización para los videos, creando nuevos niveles de grupos que facilitan la búsqueda de un determinado filtro tomando como ejemplo la organización del programa PhotoEditor y AmpedFive.

En la versión original se encontraba la siguiente organización:

- Resize (redimensionado tamaño de cuadro)
    - Point resize, bilinear resize, bicubic resize, spline36 resize y lanczos4 resize.
- Enhance (Realzado)
    - Sharpen, blur, deblock, spatial/temporal median filter, levels, temporal soften, spatial soften e invert.
- Deinterlace
    - Separate fields, complement parity y bob.
- Others
    - Turn left, turn right, turn 180, flip horizontal, flip vertical, reverse, show red, show blue y show green.

Tras la incorporación de los antes citados filtros, sumado a la nueva estructura de grupos el resultado es:

- Filters
    - Spatial Domain
        - General convolution, adaptative median, sharpen, spatial soften, blur y DCT filter.
    - Temporal domain
        - Temporal soften.
- Adjustements
    - Enhance
        - Deblock.
    - Resize
        - Point resize, bilinear resize, bicubic resize, spline36 resize y lanczos4 resize.
    - Deinterlace
        - Separate fields, complement parity y bob.
    - Tweak
    - Reverse
- Operations
    - Point operations
        - Invert.
    - Geometric operations
        - Rotate, turn left, turn right, turn 180, flip horizontal y flip vertical.
- Color space
    - RGB space
        - Show red, show blue y show green
    - RGB Adjust
    - Color YUV
    - Grey scale
    - Levels

- Others
    - Video Info
    - Gaussian noise

Aparte de estos filtros destacados en negrita como novedades incluidas respecto de la versión original de Forevid, lo ya citados filtros de cambio de perspectiva y transformada de Fourier. Ambos se consideraron más importantes dentro de las operaciones de tratamiento de imagen disponibles por lo que, al igual que otras acciones como extracción de fotogramas o marcadores, se ha realizado un acceso directo a estos procesos de video desde el panel superior de la ventana principal de Forevid y el menú de acciones desplegable del mismo.

7. Distribución de la aplicación

Para concluir el proyecto, y ya que se enfocan las mejoras de la aplicación para el aprendizaje y uso práctico de la teoría vista, se estimó el poder instalar la aplicación como cualquier otra desde un ejecutable o archivo como forma de distribución para los usuarios. Desde un inicio, para dicha tarea se trató de estudiar y aprender a partir del ejecutable *.exe que se ofrecía desde la página oficial de Forevid ya que utilizaba este formato de distribución, además de ofrecer el código fuente como posibilidades de descarga. El excrutamiento del archivo no dio resultados positivos o reutlizables para el que se pretendía crear desde este proyecto, incapaz de hallar la forma a través de la cual se creó a partir del código fuente, y cómo se utilizaba el intérprete Python necesario que ejecutaba el código a través de una librería portable que se localizaba dentro de los archivos de instalación de la aplicación. Además de esto, se debía incluir el resto de librerías externas que se ejecutan durante el funcionamiento (PyQt, PyOpenGL, wxPython, PIL y reportlab), y reproductores de video junto con los plugins necesarios de Avisynth.

Gracias a la colaboración de Andrés Romeu Hernández, un amigo informático al que desde aquí quiero agradecer su tiempo e interés prestados, y cuya experiencia laboral le ha permitido ayudarme en la maquetación y compilación del instalable ofreciéndome varias vías de desarrollo para el cometido que ocupa. Todas estas nuevas propuestas partían desde cero ignorando los anteriores métodos fallidos comentados. Una tarde completa y gran parte de la noche del día siguiente con apenas una parada para la cena fueron suficientes para encarrilar y plantear la solución de creación del ejecutable a medida que se iban presentando diferentes problemas que se detallarán más adelante, completando una instalación limpia y lo más similar posible a la original ofrecida por Forevid.

7.1. Entorno de trabajo y acciones del script

En primer lugar se creó una máquina virtual en VirtualBox sobre nuestro sistema operativo para las pruebas y que posibles fallos o acciones que ejecutásemos no afectasen a nuestro equipo. También se utilizó para comprobar la compatibilidad del ejecutable Forevid compilado sobre diferentes versiones de Windows.

Debido a la necesidad de realizar una instalación personalizada que incluyese los módulos sobre los que trabaja la aplicación además del intérprete que los ejecuta, librerías requeridas y acceso directo desde el escritorio se ha utilizado ejecución de línea de comandos de Windows (cmd), creando un archivo de ejecuciones *.but mediante el cual se han realizado pruebas de ejecuciones de comandos que nos realizasen este trabajo:

- Intérprete Python: No se ha requerido la instalación desde su ejecutable *.msi, logrando incluir en el propio compilado los archivos necesarios para su integración en Windows, realizando una instalación limpia que no implicase la modificación del sistema operativo instalando el intérprete como tal.

Sin embargo, en el momento de la ejecución del módulo principal de Forevid faltaba la asociación de la extensión del archivo (*.py) al intérprete de código Python de forma predeterminada. Esto se ha solventado mediante la incorporación de forma manual en los registros de Windows, accediendo al registro y exportando las claves del mismo que asociaban la acción de ejecutar el intérprete de nuestro código al abrir un archivo con la extensión *.py. Una vez hecho esto se obtiene un archivo *.reg que se incluye en el proceso de instalación y que hace referencia a estos registros exportados, ejecutando los cambios en el SO para realizar estas acciones de registro.

- Código fuente de la aplicación: El código del programa Forevid se ha copiado a la carpeta del sistema operativo "*C:\Programs Files\Forevid*", dentro del cual se incluyen las librerías del intérprete Python y sus módulos correspondientes para la ejecución.
- Librerías en el sistema operativo: Dos archivos se deben incluir en las rutas del SO para el registro de las funcionalidades incluidas en ellas y poder usarlas:
    - *libfftw3f-3.dll*: copiado a "*C:\WINDOWS\SYSTEM32*" para todas las plataformas desde WINDOWS 2000.
    - *python27.dll*: copiado a "*C:\WINDOWS\SYSTEM32\"* para Windows XP/2000, ó "C*:\WINDOWS\SYSWOW64*" para Windows Vista/7/8.
- Acceso directo: Por último se lleva a cabo la creación de un acceso directo al escritorio del usuario del módulo principal de Forevid para su ejecución.

Estas acciones durante la instalación de Forevid conllevaron diversos problemas de permisos de usuario en un inicio, pero se corrigieron otorgando permisos de administrador durante la ejecución, además de "ocultar" en la ventana de ejecuciones los comandos que se ejecutaban al usuario siendo esta silenciosa.

El escenario de pruebas ejecutado desde línea de comandos sirvió para fijar todos estos procesos y acciones, pero realmente este método únicamente servía para esto, la finalidad era crear un proceso de instalación con las ventanas y cuadros de diálogos típicos para cualquier aplicación.

A la hora de incluir todos estos procesos en un ejecutable se utilizó la herramienta Advanced Installer v10.9.1, el cual permite una personalización del proceso e inclusión de aceptación de los términos generales de licencias públicas a través de la cual se distribuye la aplicación.

7.2. Ejecutable final para distribución

Advanced Installer ofrece una interfaz sencilla para la creación y personalización el proceso de instalación de la aplicación e incluir todos los archivos y comandos a ejecutar que hemos detallado anteriormente. Tras lanzar Advence Installer, tenemos que completar algunas de las opciones que las pestañas en la parte izquierda de la pantalla nos ofrece la aplicación:

- **Product details**: Definimos el nombre de la aplicación, versión, nombre del autor, URL´s del producto, email de contacto, y descripción general del producto.

- Add or remove Programs: integración del producto en el panel de control de Windows.
- Icono del paquete *.exe y en la lista del panel de control para quitar o cambiar programa.

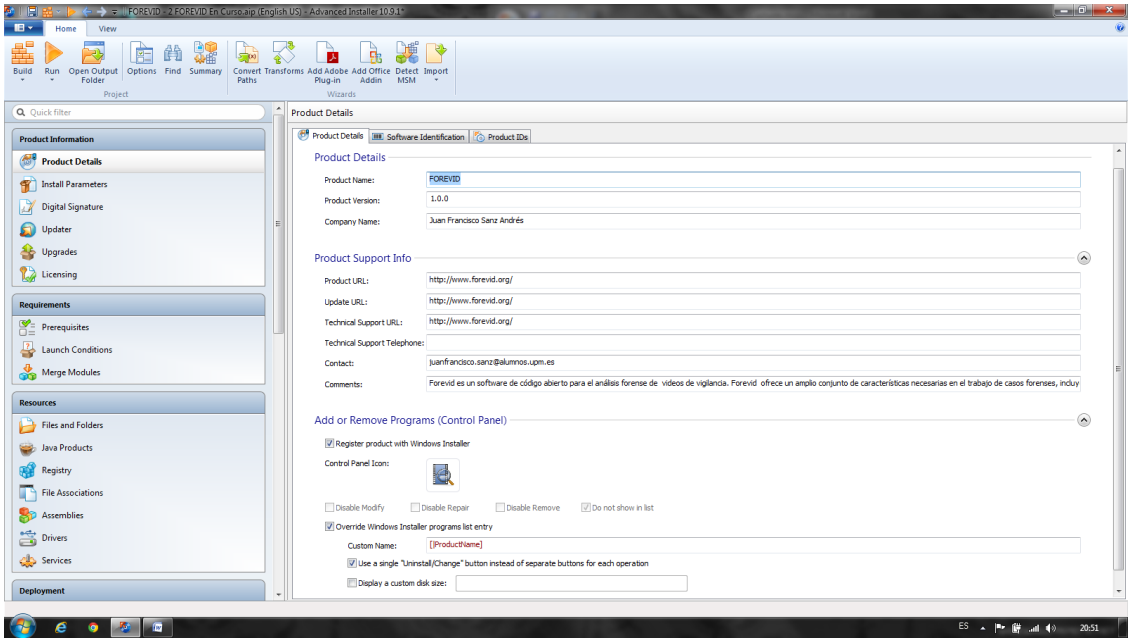


Figura 7.1: Advanced Installer - Product Details

- **Install parameters**: Indicamos la ruta donde se van a copiar los archivos de la aplicación, y a que archivo hace referencia el acceso directo que se creará en el escritorio.
  - **Package type**: se realiza sobre paquetes de 32 bits para compatibilizar con máquinas tanto de 32 como de 64 bits.
  - **Installation type**: *Per –machine only*, instalación por máquina y no por usuario, de manera que la aplicación será disponible para cualquier usuario del sistema operativo.
  - **Reboot behaviour**: *Promp for reboot when requiered*. Solicita el reinicio del SO si es fuese necesario para terminar de completar la instalación. No necesario en nuestro caso.
  - **Run as administrator**: ejecutar la instalación con los máximos permisos de administrador, de esta forma nos aseguramos la correcta copia de las librerías dlls en las carpetas del sistema.

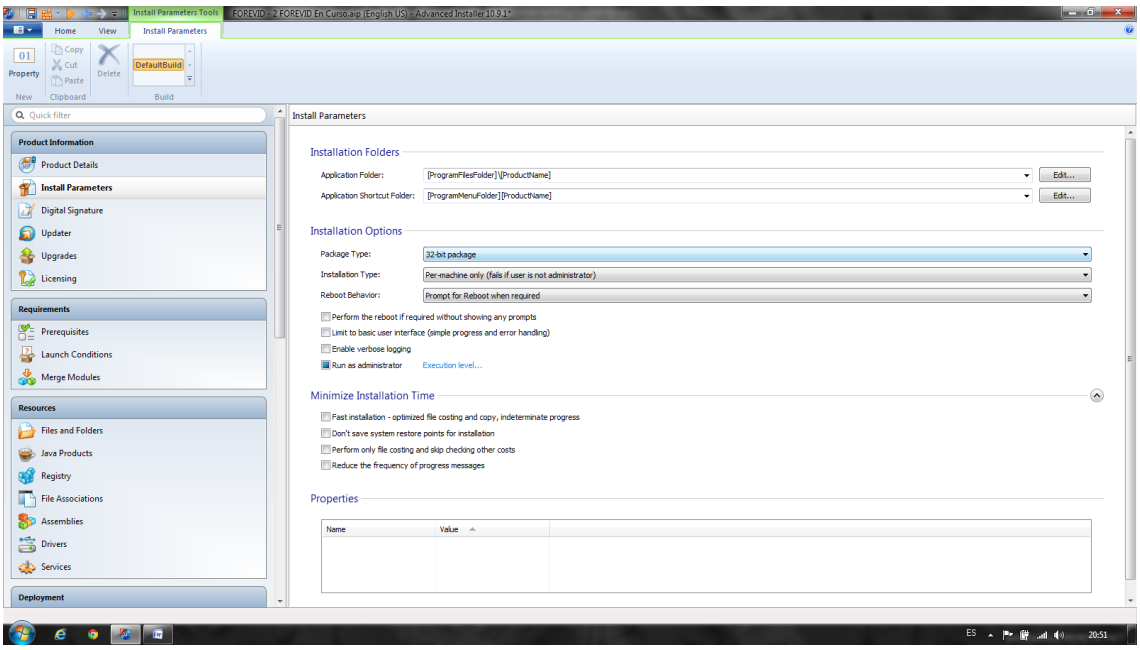


Figura 7.2: Advanced Installer - Install Parameters

- **Upgrades**: Indicamos que la instalación pueda ser una actualización de versiones anteriores, y que se instale en la ruta de la versión anterior. Antes de tal acción, se desinstalaría la versión antigua del sistema.

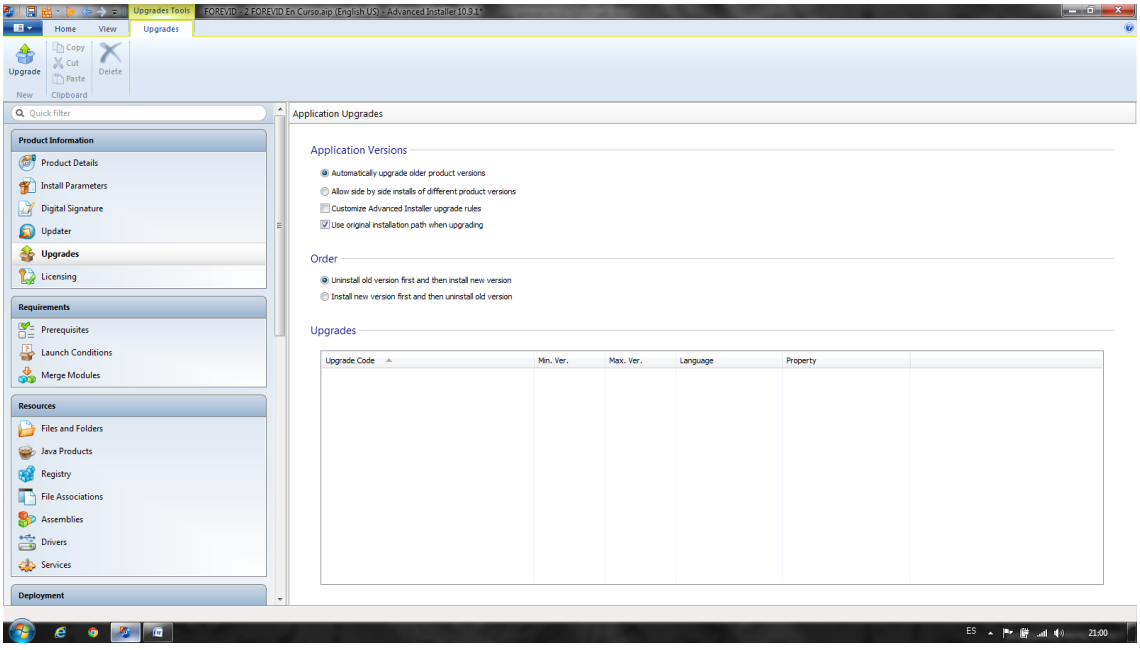


Figura 7.3: Advanced Installer - Upgrades

- **Launch Conditions**: Añadimos los sistemas operativos de Windows en los que podrá correr la aplicación.

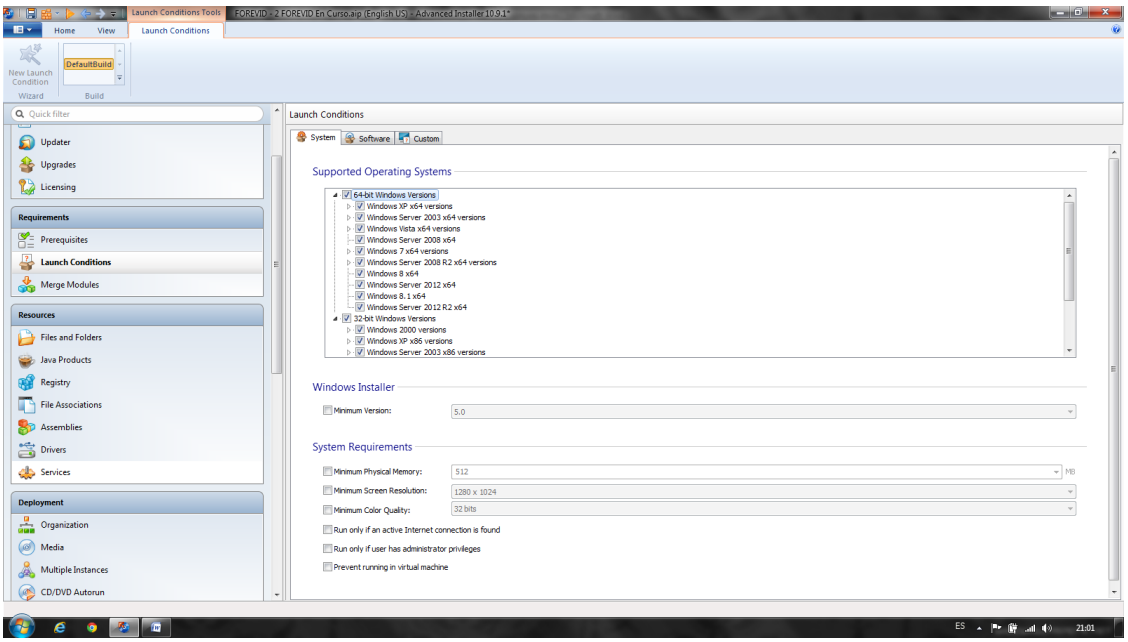


Figura 7.4: Advanced Installer - Launch Conditions

- **Files and Folders**: Ubicación y directorio de los archivos donde se va a proceder a la instalación.

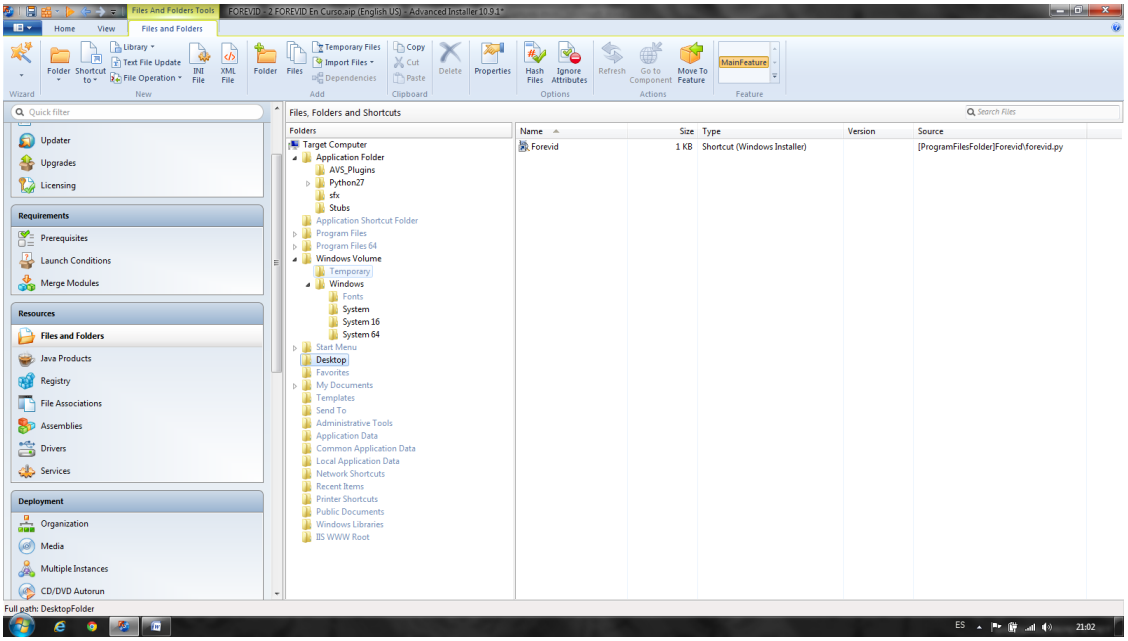


Figura 7.5: Advanced Installer - Files and Folders

- **Applicattion folder**: ruta raíz de la aplicación.
  - Python27: intérprete dentro de la aplicación.
  - Carpetas de forevid(sfx, Stubs, AVS_plugins).
- **Windows Volume**: ubicación del sistema Windows (dlls).
- **Desktop**: ubicación del acceso directo. Edición de etiqueta.
  - Es necesaria la edición del comportamiento del acceso directo, de forma que al abrirlo apunte a la carpeta de instalación del producto, ya

que venía deshabilitada. Campo “Working Directory: APPDIR, “desktop folder” por defecto”.

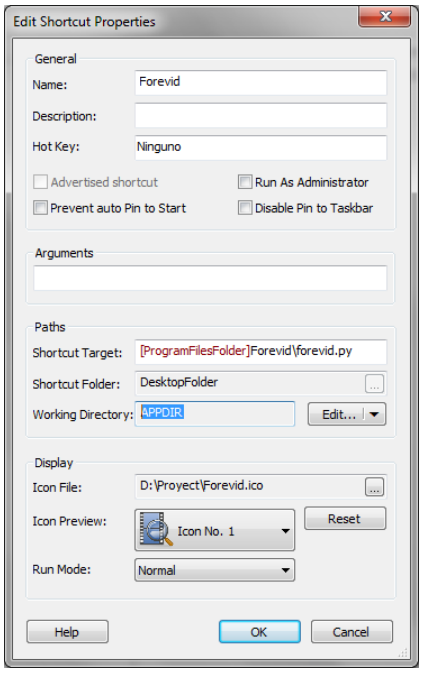


Figura 7.6: Advanced Installer - Acceso Directo: Propiedades

- **Registry**: importación del registro Windows (*.reg) creado previamente en el punto 7.1 del sistema operativo. Con la pestaña de la parte superior del programa “Impor REG” podemos habilitar esta opción para incluirla en el *.exe.

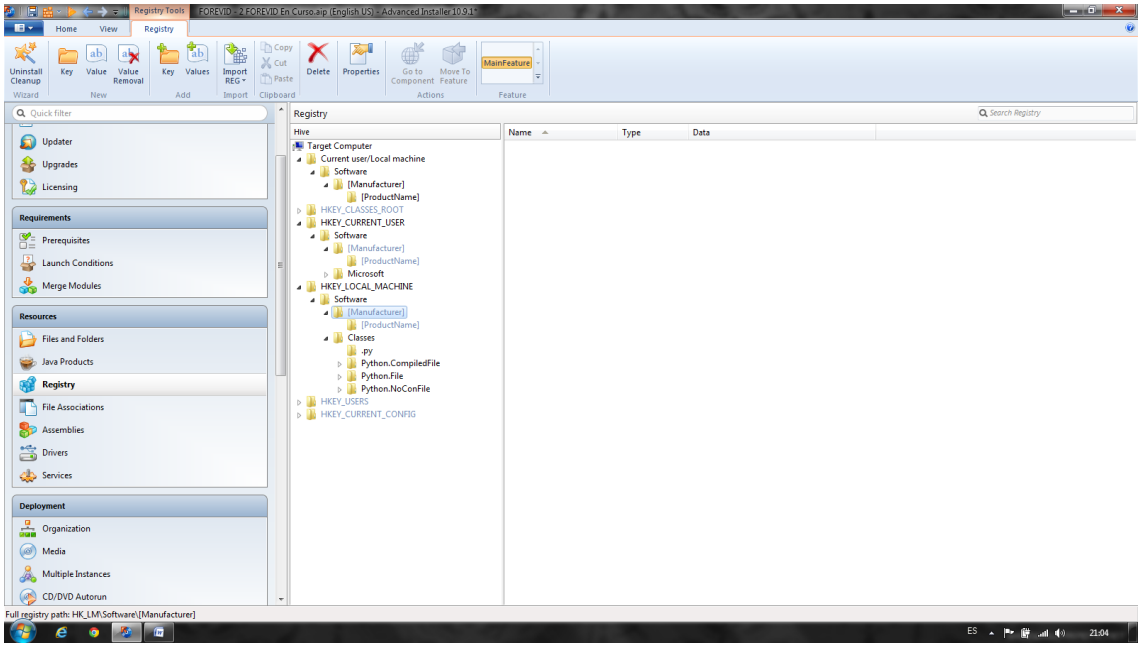


Figura 7.7: Advanced Installer - Registry

- **Media**: Indicamos que todos los archivos necesarios se incluirán dentro de un único *.exe.
    - Ruta de carpeta de los ficheros de la aplicación.

- Nombre del *.exe.
- Icono del *.exe.
- Tipo de compresión del archivo. Compresión del *.exe estándar. Se realizaron pruebas con compresión mayor en las que se obtuvieron errores durante la instalación, por lo que se mantuvo esta opción.

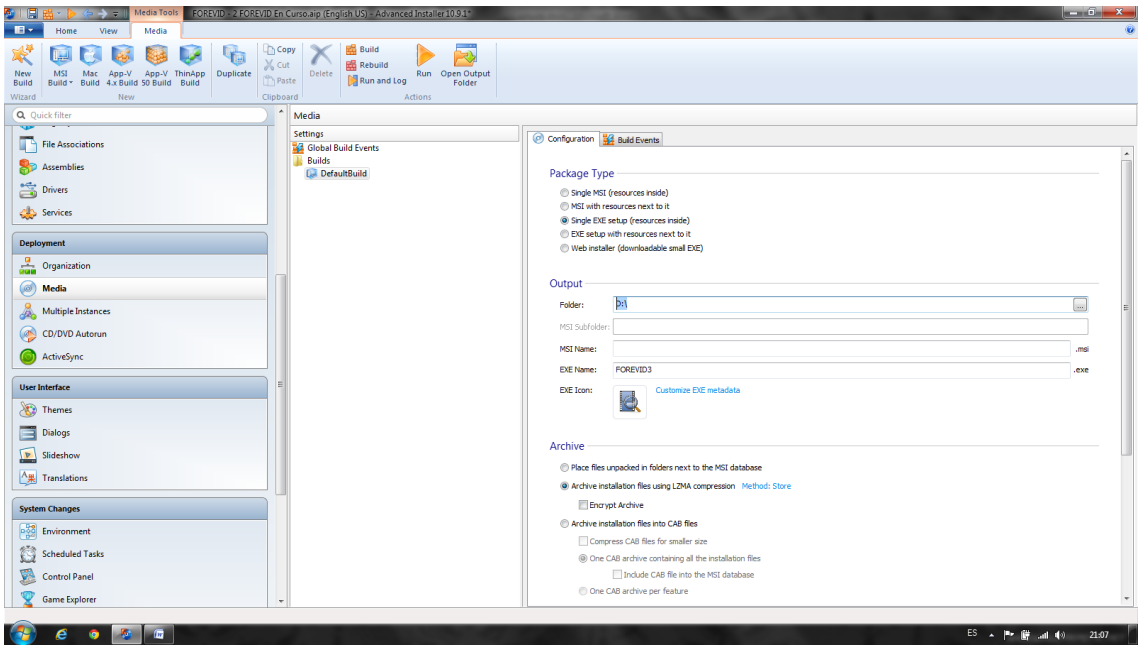


Figura 7.8: Advanced Installer - Media

- **Theme**: Tema visual de la interfaz durante la instalación.

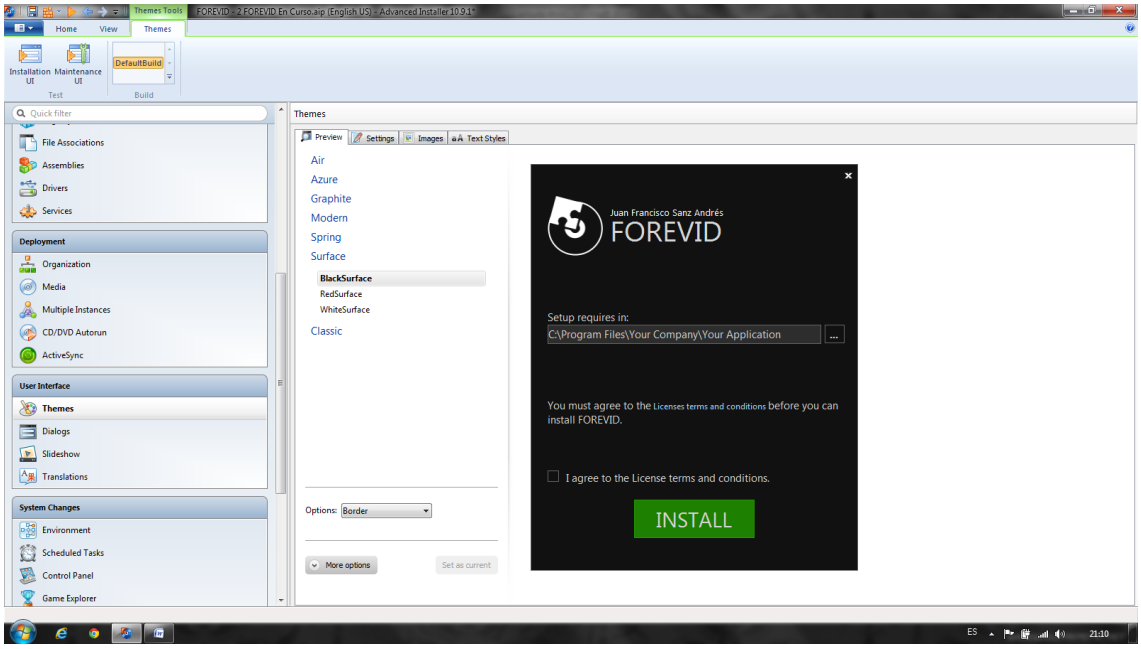


Figura 7.9: Advanced Installer - Theme

- **Dialogs**: Muestra los diferentes cuadros de diálogos personalizables durante el curso de la instalación.

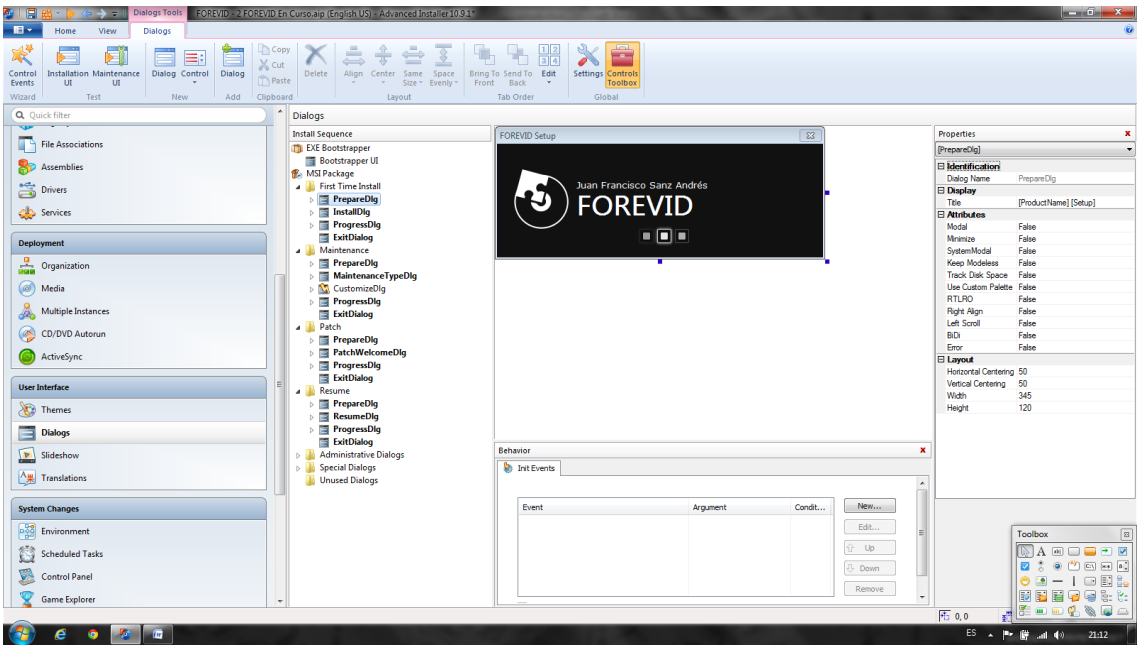


Figura 7.10: Advanced Installer - Dialogs

- **Translations**: Incluir todas las traducciones de los textos mostrados durante el proceso de instalación.

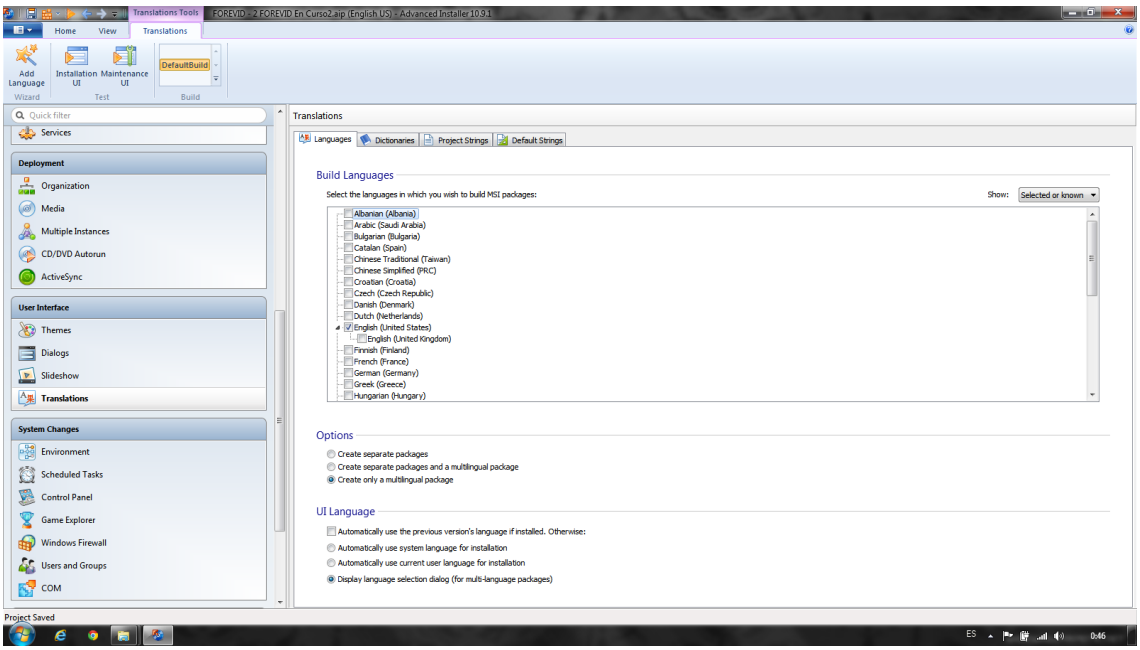


Figura 7.11: Advanced Installer - Translations

Con todos los ajustes realizados podemos efectuar la creación del archivo ejecutable, y ya que el proceso no ha sido igual que el que se realizó para el descargable desde la página de Forevid principalmente debido a que en éste se han utilizado librerías portables para todas y cada una de las acciones que requiere Forevid (desde código fuente hasta intérprete Python), el tamaño resultante es bastante mayor. También se debe a que en la versión modificada de este proyecto se han incluido nuevos scripts de Python y plugins de Avisynth.

8. Conclusiones

Ya que el objetivo principal de este proyecto era la implementación de mejoras sustanciales dentro de la aplicación Forevid, se puede decir que se ha completado en gran medida, aunque debido al desconocimiento inicial de funcionamiento e inexperiencia dentro de la programación avanzada, la gran parte del tiempo empleado en el proyecto se ha dedicado a la investigación y aprendizaje sobre el mismo y las herramientas que se han empleado para su desarrollo, por lo que no todos los objetivos propuestos inicialmente de mejoras se han logrado llevar a cabo. Añadido al motivo anterior (el aprendizaje del lenguaje de programación Python) con el cual se sabía que se había desarrollado la aplicación, hay que añadirle la inclusión del lenguaje C++ como código de más bajo nivel empleado para el procesado de videos a través de Avisynth, el cual no se ha logrado a dominar hasta el punto de poder implementar nuevos procesamientos de video e integrarlas en librerías útiles para nuestra aplicación, aunque si se han llegado a modificar partes de código creado. En general, el proceso de inmersión dentro de la aplicación con todo lo que conlleva ha supuesto aproximadamente tres de las cuatro partes del tiempo total empleado.

La inmersión en el código ya creado ha sido dificultosa en un inicio, debido a la cantidad de módulos existentes a las que se realizan llamadas desde diferentes partes del módulo principal, además de funciones aportadas por librerías ya sean internas o externas desconocidas en un principio, y que hubo que documentar para poder reconocer las acciones que realizaban. La unificación de toda la aplicación una vez reconocidas las utilidades de cada módulo fue básico a la hora de poder ampliar e introducir modificaciones en el código de a posteriori. Para conocer la estructura de objetos de la aplicación, inicialmente se implementó una reducida ejecución de código como prueba fuera de Forevid con Python, en la que se manejaba desde una ventana de comandos una base de datos de una tienda con diferentes características y especificaciones. Esto sirvió para posteriormente extrapolar los conocimientos y aplicarlos a Forevid.

Como siguiente parte importante el manejo de los softwares de PyQt, tanto para traducciones de textos como creación de UI, ha sido complicado pero imprescindible, sobre todo en el caso de Qt Linguist. El desarrollo de ventanas se ha realizado en base a las ya creadas por Forevid y aprendiendo de las mismas, pero también ha sido de gran apoyo la implementación de ventanas con Qt Designer, de las que posteriormente se utilizaban instancias u objetos que la herramienta nos traducía a código Python a partir de una ventana creada visualmente, además de llamadas a señales o interconexión entre widgets de la misma.

La creación e integración de los nuevos filtros en la aplicación se ha realizado mediante el aprovechamiento de clases ya creadas, sobre todo en los primeros introducidos, los cuales requerían de parámetros similares y sencillos a los ya existentes. A medida que se procuraron añadir otros más importantes y habituales, la ventana debía ser más intuitiva y reconocible para cualquier persona experimentada en aplicativos dedicados al

tratamiento de imagen, integrando gráficos o tablas de listado, además de matrices realizadas a partir de otros objetos o accesos directos a los mismos mediante una widget icono reconocible.

La integración de estos módulos para implementación ventanas para filtros más complejos se hacía de forma local, es decir, con pruebas ejecutadas al margen de Forevid, y una vez estaban listos se integraban en la misma con los pertinentes cambios necesarios. Esta depuración de código para comprobar el buen funcionamiento de la ampliación dentro de Forevid conllevaba la realización de multitud de pruebas en ejecución de la aplicación, observando a través de las líneas de ejecuciones o comentarios los posibles fallos o acciones que se estaban llevando a cabo.

8.1. Inconvenientes

Como suele suceder, en todas las aplicaciones existen bugs o defects, comportamientos no esperados o indeseados producidos durante la ejecución del mismo, y más cuando se introducen ciertas modificaciones sobre una base creada ampliando las funcionalidades. Estos son los bugs detectados que por falta de recursos, tiempo, o un parcial desconocimiento sobre el error producido o cómo solventarlo, se han producido durante las modificaciones pero no se han podido solucionar.

- Traducción de la aplicación: Después de crear el fichero de traducciones al idioma castellano y cargarlo, varios de los strings no se tradujeron por dos motivos: o bien dicho texto no estaba incluido dentro de la funionalidad "*self.tr()*" para poder localizarlas como texto traducible, o bien se trataba de botones o widget predefinidos por clases de objetos de la herramienta PyQt.

Las ventanas de los filtros levels y de tipo resize no mostraban sus botones traducidos debido a que eran clases de objetos heredados de ParameterWindow, por lo que hubo que volver a crear esos botones después de haber iniciado el objeto heredado dentro de cada clase que los heredaba.

Al cambiar al español los nombres de los filtros, los que ejecutaban la clase resize o levels para mostrarse, en lugar de ejecutarse su clase específica se ejecutaba la ventana general parameter. Esto ocurría porque a la hora de seleccionar que clase de ventana se ejecutaba se comparaba el nombre del filtro y se asociaba a la ventaba que debía mostrarse. Ya que su traducción al castellano no era igual al original, el string de texto no se equivalía en las condiciones del código para los filtros Levels y Resize y mostraba la condición *else* que hacía ejecutar la ventana parameters. Esto sucedía tanto en la instancia donde se añaden nuevos filtros como en la que permitía editar uno ya añadido con anterioridad. Para solucionarlo, se tradujeron también las cadenas de texto del nombre de los filtros que servían de comparativo.

- Implementación de filtros: Debido a la conversión entre código Python y C++ de los parámetros de los filtros se produce un error para aquellos de tipo booleanos que impide el uso de los mismos, devolviendo un error desde Avisynth que impide la ejecución del mismo. No se ha llegado a averiguar el motivo exacto de este problema

con el paso de parámetros booleanos a Avisynth, pero se recurre a la función "*eval*" para rodear el problema y ejecutar filtros con parámetros booleanos, algo muy necesario para la función F2Quiver que desempeña la transformación en frecuencia de los clips de video.

- Conversión de espacio de colores YUV para filtrado en frecuencia: Este error ha sido uno de los que se ha intentado solventar con mayor exigencia, considerando un defecto casi bloqueante durante el procesado de un video, pero que lamentablemente no se ha podido solventar presumiblemente por falta de conocimientos sobre Avisynth y el funcionamiento de conversión de espacio de colores esperado. Se conoce que para la ejecución del filtro F2Quiver es obligatorio establecer el vídeo en un espacio de colores YUV (YUY ó YV12), sin embargo, aún con este prerrequisito establecido, al aplicar el filtro da un error de conversión de colores. La solución a esto fue introducir de forma "manual" la conversión de colores previa a esta ejecución, por lo que en ocasiones dos filtros consecutivos de conversión de espacio de colores se sucedían. Esto solventaba el problema, pero produce una carga de cálculo extra e innecesaria, teniendo en cuenta también que si se desea ejecutar este filtro en varias ocasiones, los filtros de conversión se irán añadiendo en cada ejecución, creando una cadena extensa que realentiza el procesado de video.

Una solución para evitar este problema de forma sencilla, sin llegar a solucionarlo en caso de que fuese posible, podría ser evitar la modificación de un video con filtros posteriores a un video ya modificado por un filtrado FFT restringiendo las posibilidades de edición de video pero manteniendo un comportamiento más estable de la aplicación con menos errores.

8.2. Posibles mejoras futuras

Ciertos puntos que se acordaron al inicio del proyecto no se han podido llevar a cabo por la complejidad o tiempo que suponía, por lo que han quedado como extensiones interesantes a incluir en posteriores versiones que redondearían la finalidad de este proyecto de acercar al estudiante de forma didáctica el tratamiento digital de videos y los efectos que sobre ellos producen. Aquí se detallan estos puntos, y otros que han surgido a partir de las modificaciones añadidas:

- Si se introduce un filtro para un video en un determinado idioma y posteriormente se cambia de idioma, la restricción impuesta para evitar repeticiones de filtros para un mismo clip de video no funciona, debido a que se comparan los nombres con las traducciones correspondientes en el momento en que se han añadido evitando la similitud entre ambos nombres.
- Guardar los parámetros introducidos tanto para el filtrado en frecuencia como el cambio de perspectiva una vez que se hayan aplicado para posteriormente cargarlos. Esto imitaría el funcionamiento del resto de filtros de procesamiento de video donde los valores que se han introducido previamente en un filtro son cargados si se vuelve a abrir para editar el filtro
- Optimización de código: en ciertas ventanas de la aplicación se ha incurrido en llamadas a objetos de forma reiterada y repetitiva que se pueden solventar mediante la

creación de una clase a la cual recurrir las veces necesarias con ciertas características de personalización para cada una de las llamadas, reduciendo el volumen de código y facilitando la lectura posterior del mismo, además de un mejor rendimiento de la aplicación a la hora de mostrar únicamente lo deseado, en lugar de cargar la totalidad del objeto aún sin ser mostrado (pestañas de filtros FFT)

- Nuevos filtros dedicados al procesamiento de video: frame averaging, enfoque óptico, enfoque por movimiento.
- Análisis de formas para identificación y reconocimiento de formas o caracteres dentro de una imagen que permitan la identificación de los mismos.

Referencias

- Tratamiento digital de imágenes multiespectrales. Jorge Lira Chávez. Univ. Nacional Autónoma de México.
- Univ. Rey Juan Carlos. Visión artificial – Diapositivas Tema 2: Preproceso (realzado y filtrado) de imágenes digitales.
- Procesamiento de imágenes. Universidad de Salamanca. D. González Aguilera, Dept. de Ingeniería Cartográfica y del Terreno. Escuela Politécnica Superior de Ávila.
- Transformaciones: Perspectivas.
- Procesamiento audiovisual. Transformaciones geométricas. Dept. de informática y sistemas. Universidad de Murcia. Ginés García Mateos.
- Fundamentos del procesamiento digital de imágenes. H. Melh y O. Peinado.
- http://sabia.tic.udc.es/gc/Contenidos%20adicionales/trabajos/Imagenyvideo/Procesado%20de%20imagen/TrabajoGC.htm
- http://www.varpa.org/~mgpenedo/cursos/Ip/Tema4/nodo4_3.html
- http://en.wikipedia.org/wiki/YUV
- http://www.fourcc.org/yuv.php
- http://avisynth.nl/index.php/RGB32
- http://neuron2.net/LVG/frameserving.html
- http://avisynthnew.wikinet.org/wiki/Avisynth_Plugin_Development_in_C
- http://www.mundodivx.org/foro/index.php?action=printpage;topic=37925.0;images
- http://avisynth.nl/index.php/AviSource
- http://avisynth.org.ru/docs/english/corefilters/directshowsource.htm
- http://avisynth.nl/index.php/FFmpegSource
- http://ralgozino.wordpress.com/2011/06/26/como-traducir-aplicaciones-qt4/
- http://nullege.com/codes/search/PyQt4.QtGui.QDialog.tr