PROYECTO FIN DE GRADO

TÍTULO: Actualización y mejora de la herramienta virtual “Virtual Mixer” de simulación de un mezclador de vídeo.

TITLE: Update and improvement of the virtual tool “Virtual Mixer” for simulation of a video switcher

AUTOR/A: David Sánchez Gómez

TITULACIÓN: Grado en Ingeniería de Sonido e Imagen

TUTOR/A: Martina Eckert

DEPARTAMENTO: Ingeniería Audiovisual y Comunicaciones

VºBº TUTOR/A

Miembros del Tribunal Calificador:

PRESIDENTE: Jerónimo López-Salazar Codes

TUTORA: Martina Eckert

SECRETARIO: José Luis Rodríguez Vázquez

Fecha de lectura: 11 de septiembre de 2025

Calificación:

El Secretario,

Resumen

El proyecto titulado "Actualización y mejora de la herramienta virtual 'Virtual Mixer' de simulación de un mezclador de vídeo" tiene como objetivo principal actualizar y mejorar la herramienta "Virtual Mixer" creada en App Designer de Matlab. Esta aplicación, utilizada en la asignatura de Ingeniería de Vídeo del Grado en Ingeniería de Sonido e Imagen, permite a los estudiantes simular el uso de un mezclador de vídeo profesional, facilitando la práctica y la preparación del laboratorio de la asignatura.

La aplicación, que fue creada inicialmente por el profesor Luis Ortiz en el entorno GUIDE y migrada posteriormente a App Designer por Jiajie Chen como parte de su Proyecto de Fin de Grado, es de gran utilidad, por lo que es muy interesante mejorar distintos aspectos, además de añadir funcionalidades nuevas.

En este proyecto se analiza tanto la situación actual de la aplicación como el marco tecnológico en el que se desarrolla. También se estudian las fortalezas y debilidades de la implementación actual con el fin de llevar a cabo las mejoras pertinentes. Además, se detallan los cambios realizados, tanto estéticos como técnicos, así como su funcionamiento en detalle.

Abstract

The project titled " Update and improvement of the virtual tool “Virtual Mixer” for simulation of a video switcher" aims to update and enhance the "Virtual Mixer" tool created in Matlab's App Designer. This application, used in the Video Engineering course of the Bachelor's Degree in Sound and Image Engineering, allows students to simulate the use of a professional video mixer, facilitating practice and preparation for the course's laboratory sessions.

The application, initially created by Professor Luis Ortiz in the GUIDE environment and later migrated to App Designer by Jiajie Chen as part of his Final Degree Project, is very useful, making it highly desirable to improve various aspects and add new functionalities.

This project analyzes both the current state of the application and the technological framework in which it is developed. It also examines the strengths and weaknesses of the current implementation to carry out the necessary improvements. Additionally, it details the changes made, both aesthetic and technical, as well as their detailed functionality.

Índice de figuras

Índice de tablas

Lista de acrónimos

AVI: *Audio Video Interleave*

B: *Blue*

BMP: *Bitmap*

CCU: *Camera Control Unit*

DSK: *Downstream Keyer*

DVE: *Digital Video Effects*

G: *Green*

GRSA: Guía para trabajar la Responsabilidad Social y Ambiental

GUI: *Graphical User Interface*

HD: *High Definition*

HSV: *Hue Saturation Value*

IDE: *Integrated Development Environment*

ITU: *Internacional Telecommunication Union*

JPG: *Joint Photographic Expert Group*

M/E: *Mix/Effects*

MOV: *Movie*

MP4: *Moving Picture Experts Group 4*

ODS: Objetivos para el Desarrollo Sostenible

ONU: Organización de las Naciones Unidas

PGM: Programa

PNG: *Portable Network Graphics*

PVW: *Preview*

R: *Red*

WFM: Waveform Monitor

YUV: Brillo (Y), diferencias de color (U, V)

Índice

1. Introducción

1.1 Marco y motivación del proyecto

Una de las partes más importantes de la asignatura de Ingeniería de Vídeo, que forma parte del plan de estudios del Grado en Ingeniería de Sonido e Imagen de la Escuela Técnica Superior de Ingeniería y Sistemas de Telecomunicación, es la parte práctica, la cual se desarrolla en el laboratorio de televisión. Dicho laboratorio cuenta con un plató, equipado con cuatro cámaras e iluminación, y una sala de control con dos puestos de trabajo completos.

Cada uno de estos controles disponen de diferentes equipos, siendo el equipo principal el mezclador de vídeo de la marca Blackmagic ATEM. También cuenta con monitores conectados a los mezcladores y configurados como multi-vista, un panel de control de cámaras Blackmagic ATEM Camera Control Panel, dos ordenadores equipados con tarjetas capturadoras de vídeo Ultrascope, etc.

El objetivo de dicho laboratorio es el de permitir a los alumnos el aprendizaje, no solo en el uso de estos equipos, sino también su funcionamiento interno, así como la posibilidad de analizar distintas características de la señal de vídeo. Debido a esto, conviene dedicar bastante tiempo a la práctica con los equipos y a la realización de diferentes pruebas, lo cual puede suponer un problema debido a la alta demanda del laboratorio para que todos los alumnos puedan dedicar el tiempo suficiente.

Por estas razones, en 2014 el profesor Luis Ortiz desarrolló dos herramientas virtuales en Matlab GUIDE que simulan una unidad CCU y un mezclador de vídeo, llamadas VisionCCU y VisionMixer, con el fin de facilitar a los alumnos la preparación de las prácticas con estos equipos antes de asistir presencialmente al laboratorio, así como la posibilidad de practicar más tiempo en casa posteriormente. Sin embargo, estas herramientas quedaron obsoletas por dos motivos: la sustitución de Matlab GUIDE por App Designer y la limitación a funcionalidades básicas.

Por ello, en 2021 la alumna Jiajie Chen, como parte de su proyecto de fin de grado [1] migró ambas herramientas a App Designer, llamadas VirtualCCU y VirtualMixer, y añadió algunas funcionalidades nuevas. No obstante, debido a la sustitución del antiguo mezclador de vídeo Sony por los nuevos ATEM, y a la falta de funcionalidades básicas de los mezcladores de vídeo es necesaria otra actualización para añadir nuevas funcionalidades y mejorar el rendimiento de la aplicación, optimizando el código y haciendo un uso más eficiente de los recursos.

1.2 Objetivos técnicos y académicos

El objetivo de este proyecto de fin de grado es realizar una actualización y mejora de la aplicación “VirtualMixer”, así como mejorar su capacidad didáctica de cara a la mejor comprensión por parte de los alumnos de la asignatura de Ingeniería de Vídeo del uso y funcionamiento de un mezclador de vídeo. Los objetivos concretos son los siguientes:

- Depuración y corrección de errores en el código
- Optimización:
    - Mejorar la eficiencia de los cálculos
    - Utilizar funciones y propiedades de Matlab
- Mejorar la estructura del código para hacerlo más comprensible:
    - Separación de funciones extensas en varias funciones específicas
    - Separación de funciones en ficheros independientes
- Añadir nuevas funcionalidades:
    - Generación de bordes (segundo doble multiplicador)
    - Más tipos de cortinillas
    - Carga y reproducción de vídeo
    - Mejora del monitor de forma de onda (WFM), añadir *Parade RGB*.
- Mejorar el diseño de la interfaz gráfica de usuario

Además, el proyecto también tiene como objetivo mejorar las competencias del proyectista en las siguientes materias:

- Programación en entorno de Matlab y App Designer
- Procesado de imágenes
- Optimización de código y ejecución de aplicaciones
- Funcionamiento de un mezclador de vídeo profesional

1.3 Estructura del resto de la memoria

La memoria se organiza en los siguientes apartados: El marco tecnológico, donde se detalla toda la información necesaria para comprender el resto del proyecto, especialmente relativa al mezclador de vídeo y sus funciones, los monitores para el análisis de la señal, la herramienta App Designer de Matlab y un resumen de proyectos anteriores. También se exponen las especificaciones y restricciones de diseño, donde se enumeran los detalles técnicos que debe cumplir la aplicación y qué funcionalidades debe incluir. A continuación, se expone la solución propuesta detalladamente, incluyendo las fases de depuración y corrección de errores y creación de nuevas funcionalidades. Posteriormente se muestran los resultados obtenidos, el impacto del proyecto y una serie de conclusiones obtenidas durante el desarrollo del proyecto. También se detalla el posible trabajo futuro a desarrollar en otros proyectos. Por último, se muestran las referencias utilizadas, así como los anexos de presupuesto y manual de usuario.

2. Marco tecnológico

Para entender el trabajo realizado en la aplicación "VirtualMixer" es necesario estudiar los aspectos básicos de un mezclador de vídeo profesional y de su uso en un entorno de producción audiovisual.

2.1 Mezclador de vídeo

El mezclador de vídeo es un equipo presente en la producción audiovisual que combina múltiples fuentes de video para crear una única salida. A continuación, se hace un análisis de sus principales características y funcionamiento.

2.1.1 Funcionamiento básico

El mezclador de vídeo, también llamado mesa de mezclas de vídeo o *video switcher* [2] es un equipo encargado de gestionar diferentes señales de vídeo que recibe a través de sus entradas o que genera internamente, de forma que, mediante el uso de transiciones o composiciones, crea una señal de salida lista para emisión o grabación llamada programa.

El mezclador de vídeo dispone de una interfaz de usuario física que consiste en una serie de botones, *faders* (deslizadores) y *knobs* (botones giratorios) que sirven al operador para elegir las señales de entrada, el tipo de transición o composición a realizar y la cantidad de cada una de las señales mostradas en la salida. Además, dispone de una pantalla externa donde se muestran los diferentes *displays* (pantallas) tanto con las señales de entrada como dos pantallas especialmente importantes, como son la pantalla de previo (*preview*) y la de programa (*program*). En la Figura 1 se muestra un diagrama básico de un mezclador de vídeo.

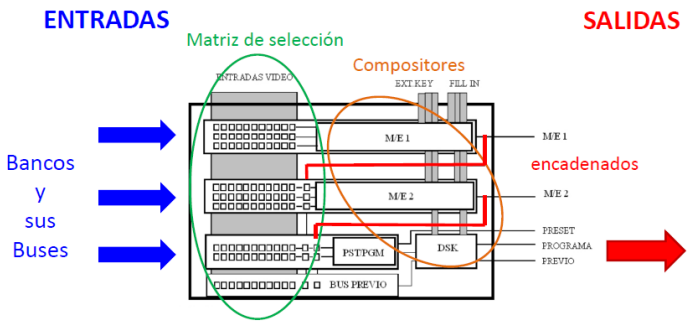

Figura 1. Diagrama de un mezclador de vídeo básico [2].

2.1.2 Procesador M/E

El procesador de Mezcla/Efecto (M/E) es el sistema principal del mezclador de vídeo, pudiendo disponer tan solo de un procesador M/E o de varios, en cuyo caso la salida de uno se puede encadenar a la entrada de otro para componer imágenes con mayor complejidad. Consiste en dos buses de señales, desde donde se seleccionan las fuentes de entrada, y un

bus auxiliar, a partir del cual se selecciona la señal de entrada para general la señal de control, si es necesario:

- Buses de entrada A y B: Son los buses donde se eligen las entradas al procesador M/E y por lo tanto al doble multiplicador.
- Bus Auxiliar: Es el bus donde se selecciona la señal a partir de la cual se calcula la máscara C en caso de ser necesario.

Además, como mínimo cuenta con dos salidas:

- Salida de previo: Es la salida utilizada para previsualizar cualquier señal antes de su envío a la salida de programa.
- Salida de programa: Es la salida del mezclador, es la señal que se envía a emisión o grabación.

También dispone de un *fader*, que gobierna la señal de control, y de un dispositivo llamado doble multiplicador.

En la Figura 2 se muestra un diagrama más detallado de un mezclador de vídeo con dos dobles multiplicadores, que se explican en el siguiente apartado, donde se pueden ver los diferentes buses de entrada a la izquierda de la imagen, los procesadores para obtener las máscaras para las transiciones *Luma* y *Chroma Key*, un generador de transiciones, un generador de bordes y, por último, la salida de programa a la derecha [3].

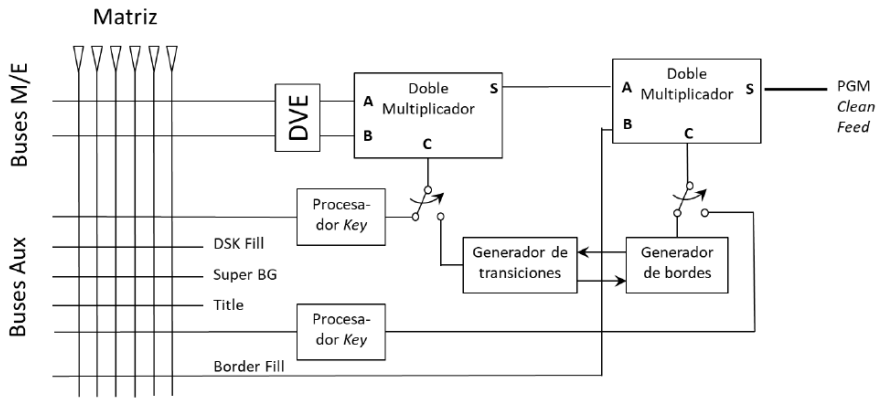

Figura 2. Diagrama de un mezclador de vídeo con dos dobles multiplicadores [3].

Doble Multiplicador

Es el encargado de procesar las imágenes de entrada en función de una señal de control, también llamada máscara. Su funcionamiento se basa en la siguiente operación (1)

$$S = A \cdot C + B \cdot (1 - C) \tag{1}$$

Donde: S: señal de salida

A y B: señales de entrada

C: señal de control

La señal de control C es una matriz de píxeles del mismo tamaño que las imágenes cuyo valor está comprendido entre 0 y 1. Cada píxel de la imagen de salida tendrá, por lo tanto, una cantidad del mismo píxel de A y de B en función del valor del píxel correspondiente en la máscara C. Todas las operaciones del doble multiplicador se realizan píxel a píxel y en tiempo real. En la Figura 3 se puede ver un esquema de su funcionamiento [2], donde se muestran las señales de entrada A y B a la izquierda, la señal de control C abajo, y la señal de salida S a la derecha.

$$S = A \cdot C + B \cdot (1 - C)$$

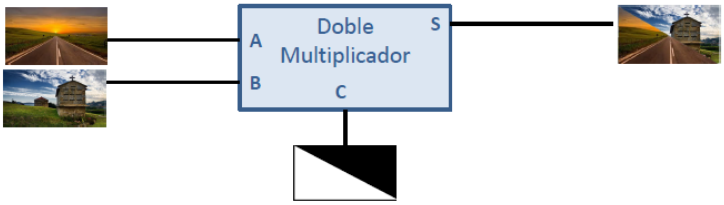

Figura 3. Diagrama del doble multiplicador [2].

Transiciones

Las transiciones son las diferentes formas de pasar de una señal A a una señal B. A continuación, se detallan las principales transiciones presentes en cualquier mezclador de vídeo.

Corte (*Cut*)

Consiste en el paso de una señal a otra de forma abrupta, sin ningún proceso intermedio. Es la transición más común en el mundo audiovisual.

Mezcla (*Mix* o *fade*)

En esta transición la imagen de salida va pasando de la señal A a la señal B de forma gradual y homogénea, de tal modo que durante un tiempo determinado se muestran ambas señales a la vez en diferente porcentaje. La cantidad de cada señal presente en la imagen de salida se puede controlar mediante el *fader*, o también se puede hacer de forma automática definiendo una velocidad de transición.

Cortinilla (*Wipe*)

Esta transición consiste en realizar el cambio de una señal a otra en base a una figura geométrica, por lo tanto, existen múltiples tipos de cortinilla. Por ejemplo, cortinilla circular, cuadrada, rombo, barrido vertical u horizontal, etc., en función de la figura geométrica utilizada en la máscara. Dependiendo de la figura, la transición se lleva a cabo de diferente manera, por ejemplo, en los barridos la transición se realiza desplazando la línea que separa ambas regiones a lo ancho o alto de la imagen. Sin embargo, en el caso de una cortinilla cuadrada o circular, la transición se realiza cambiando el tamaño de la figura.

En la Figura 4 se pueden ver unos ejemplos de cada una de estas transiciones, además de una representación de la señal de salida en función del tiempo [4].

Figura 4. Transición al corte (arriba izquierda), transición mezcla (arriba derecha) y transición cortinilla (abajo) [4].

Keying

Consiste en componer una imagen de salida a partir de dos señales de entrada en función de una señal *key* (clave), que puede ser obtenida a partir de diferentes procesos, típicamente a partir de la información de luminancia (*luma key*) o de crominancia (C*hroma Key*) de una imagen. Esta señal *key* se utiliza para calcular la máscara C, que después se envía al doble multiplicador.

- **Luma Key:**
  Consiste en utilizar como señal *key* el brillo de una imagen para crear la máscara. Se obtiene convirtiendo la imagen al espacio de color Y Cb Cr, del cual se extrae la componente de luminancia (Y). A continuación, se comparan los valores de luminancia de los píxeles con un umbral predefinido. Los píxeles que superan dicho umbral se establecen con el valor 1 en la máscara, y el resto con el valor 0.
- **Chroma Key:**
  Para crear la máscara se utiliza como señal *key* la información de crominancia de una imagen, y se selecciona un color o rango de colores. En este caso, se convierten tanto la imagen como el color al espacio de color HSV (brillo, saturación, valor) y se comparan los valores de cada píxel de la imagen con los valores del color elegido. Los píxeles que tienen el mismo valor en las tres componentes (o cercanos, si se establecen unos niveles de tolerancia) se ponen a 1 en la máscara, el resto se ponen a 0.

En la Figura 5 se muestran dos ejemplos de *keying* a partir de la información de luminancia y de crominancia, utilizando en este último caso como *key* el color verde [2].

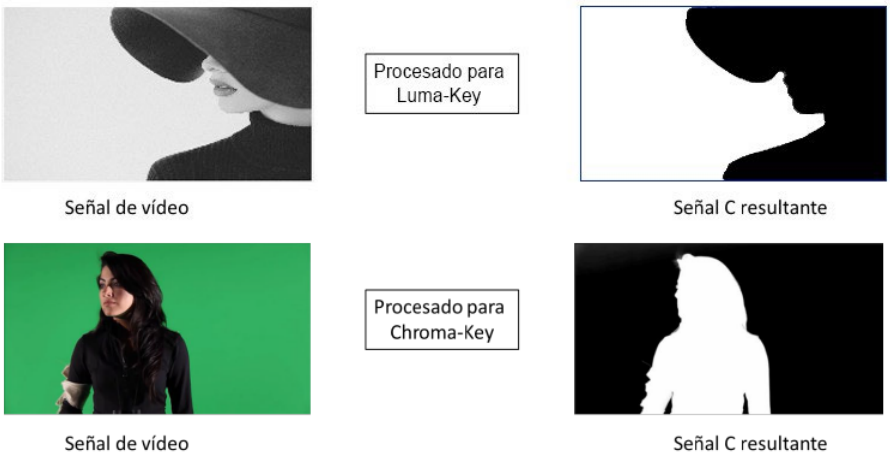

Figura 5. Ejemplo máscara a partir de *Luma Key* (arriba) y de máscara a partir de *Chroma Key* verde (abajo) [2].

Segundo doble multiplicador

Para determinadas funcionalidades es necesario encadenar la salida del primer doble multiplicador a la entrada de un segundo doble multiplicador. Un ejemplo de esto es cuando se desea añadir bordes a la frontera entre una imagen y otra. En este caso es necesario calcular una segunda máscara, llamada C2.

El esquema resultante es el representado en la Figura 6 [2], donde se pueden ver las dos señales de entrada A y B combinadas mediante una cortinilla controlada por la máscara C y cuya salida está procesada por el primer doble multiplicador. Esta salida se utiliza como entrada en el segundo doble multiplicador, junto a otra señal de entrada generada internamente con el color del borde. La máscara C2, creada de forma sincronizada con la máscara C1, controla la composición de la señal de salida del segundo doble multiplicador.

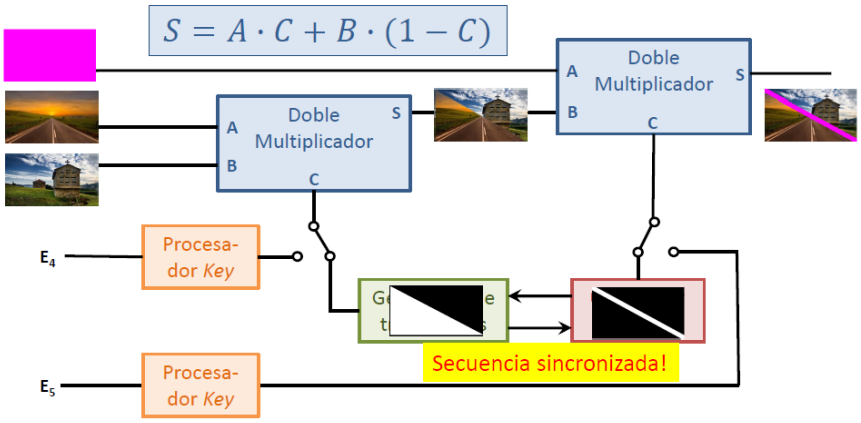

Figura 6. Diagrama de dos dobles multiplicadores [2].

2.2 Monitor para análisis de la señal

Con el fin de analizar distintas características de la señal de vídeo y, si es necesario, realizar ajustes, se hace uso de un *display* (pantalla) capaz de hacer varias representaciones que muestran distintos aspectos de la señal [5].

2.2.1 Monitor de forma de onda

El monitor de forma de onda (WFM) hace una representación del valor de cada píxel en función de su posición horizontal de cada línea de la imagen, superponiendo todas las líneas. Si la imagen a analizar es a color son necesarias tres representaciones, una por cada componente de color. A esta representación se la conoce con el nombre de *parade* (en fila), y se puede mostrar en diferentes espacios de color, como por ejemplo RGB e YCbCr [6].

Parade RGB

Consiste en una triple representación, una para la componente roja (R), otra para la componente verde (G) y otra para la azul (B).

Parade Y Cb Cr

En este caso las componentes representadas son una de luminancia (Y) y dos de diferencia de color (Cb y Cr).

En la Figura 7 se pueden ver dos modos de *parade*, RGB e YCbCr.

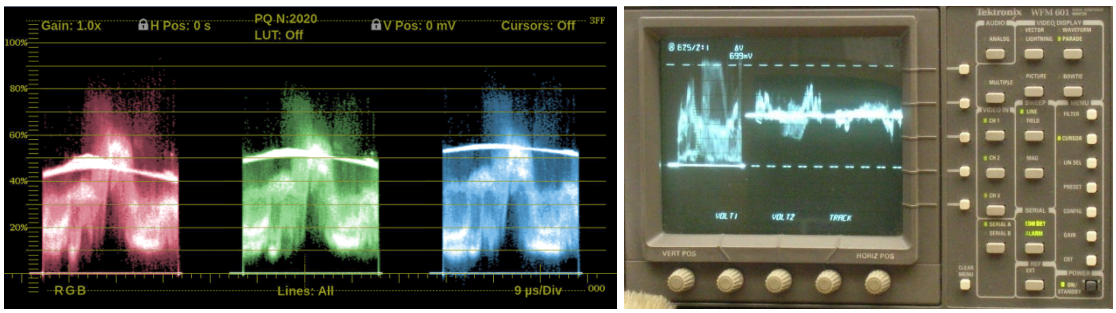

Figura 7. Parade RGB (izquierda) [7] y Parade Y Cb Cr (derecha) [8].

2.2.2 Vectorscopio

Otra forma de analizar la señal es usando un vectorscopio, que muestra la información de color de la señal mediante una representación X-Y, de forma que en el eje vertical se tiene la componente Cr y en el eje horizontal la componente Cb [5]. En la Figura 8 se muestra un diagrama de su construcción.

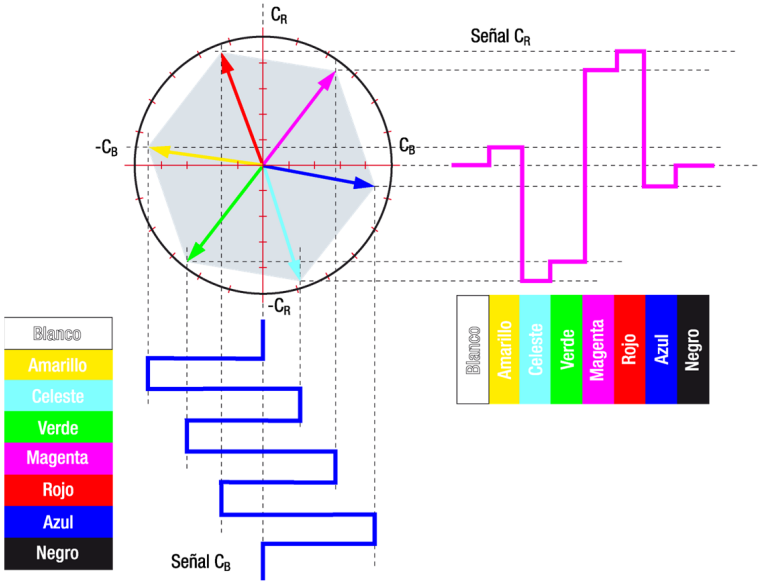

Figura 8. Propiedades del vectorscopio [9].

Además, la pantalla del vectorscopio cuenta normalmente con 6 casillas cuyas posiciones se corresponden con las amplitudes máximas de los colores primarios y secundarios (R, G, B, Yl, Cy y Mg) de una señal de carta de barras con una saturación del 75%. El vectorscopio no aporta información espacial ni de brillo.

2.2.3 Gamut

La última de las representaciones de la señal que se han tenido en cuenta para el análisis de la señal en la aplicación es el *Gamut*. El término *gamut* hace referencia a la gama de colores que puede reproducir un sistema de vídeo, por lo tanto, los colores cuyos valores exceden los límites del *gamut* se consideran ilegales. Dichos valores ilegales pueden aparecer al realizar la conversión entre distintos espacios de color, como, por ejemplo, de RGB a YCbCr durante la transmisión, al realizar corrección de color, o al generar gráficos digitales. Para controlar dichos niveles se hace uso de la visualización en forma de dos rombos que se denomina "Diamante", y que ayuda a mantener los límites [10].

Al igual que el vectorscopio, aporta información sobre el color de la imagen. Consiste en una representación formada por dos rombos, posicionados en vertical. En el eje horizontal del diamante superior se representa B – G, mientras que en el eje vertical se representa B + G. En el eje horizontal del diamante inferior se representa R – G, y en el eje vertical – R + G. Los diamantes están formados por las líneas que unen los valores máximos que pueden tomar dichas operaciones en una señal de vídeo analógica. En la Figura 9 se muestra un esquema de su construcción.

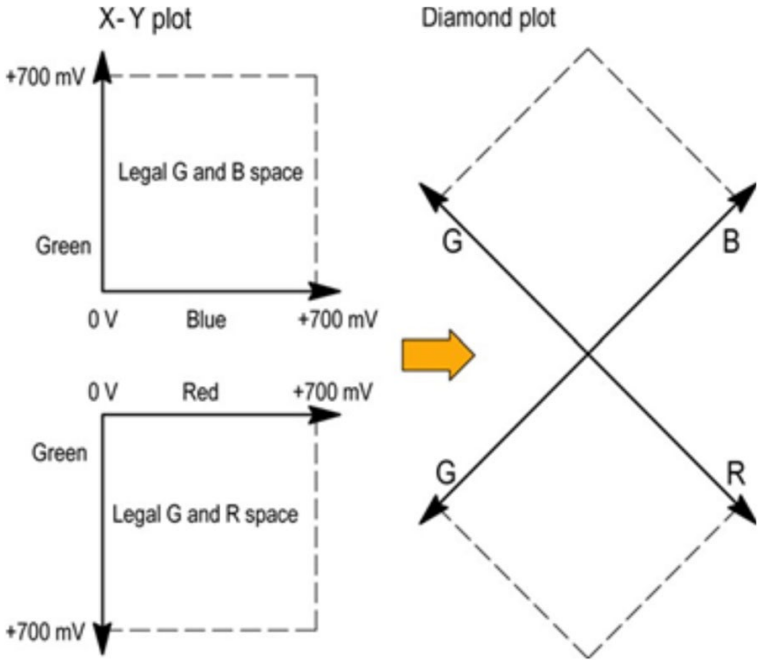

Figura 9. Construcción de la representación diamante [10].

2.3 Carta de ajuste

Una carta de ajuste consiste en una señal de barras de color del 75% o 100%, que se utiliza para comprobar la correcta recepción de la señal de vídeo y su posible ajuste en el caso en el que los niveles no sean correctos, como se establece en la norma ITU R BT.471 [11] Consiste en una señal formada por ocho barras, cada una con distintos niveles en cada componente RGB para conseguir una barra blanca, otra negra, los tres colores primarios (rojo, verde y azul) y los tres secundarios (amarillo, verde y magenta). En la Figura 10 se representa la señal de barras de color del 75% y en la Tabla 1 los niveles relativos a cada barra en cada componente de color. En este caso los niveles de la barra blanca serían del 100%, es decir, el máximo valor posible de la señal.

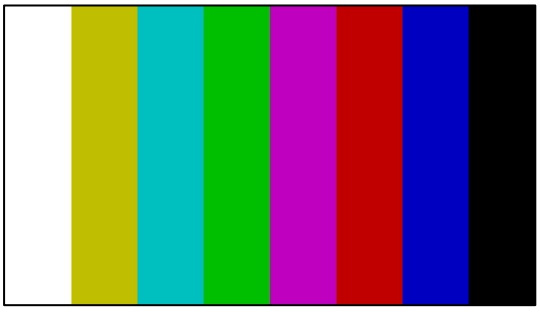

Figura 10. Barras de color del 75%.

Tabla 1. Niveles RGB de las barras de color del 75%.

	R (%)	G (%)	B (%)
Blanco	100	100	100
Amarillo	75	75	0
Cian	0	75	75
Verde	0	75	0
Magenta	75	0	75
Rojo	75	0	0
Azul	0	0	75
Negro	0	0	0

Esta señal se utiliza para ajustar los niveles mínimo (negro) y máximo (blanco) a los valores de 0 y 0.7 V con el WFM y el nivel de saturación en el vectorscopio, de forma que cada barra de color coincida con su casilla correspondiente. En la Figura 11 se muestra la representación de una señal de barras de color del 75% en un vectorscopio, donde se aprecia como cada color coincide con su casilla correspondiente.

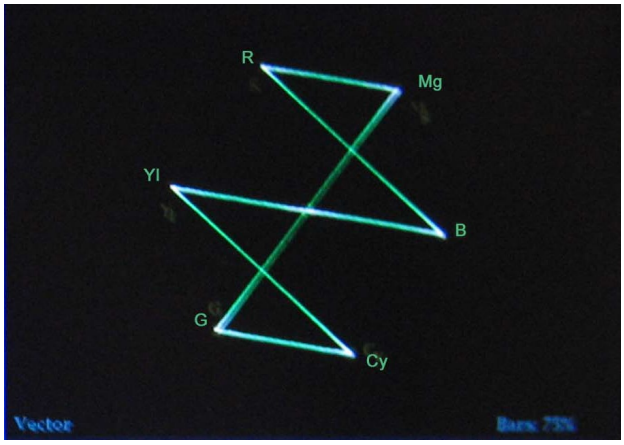

Figura 11. Barras de color del 75% en un vectorscopio [12].

2.4 App Designer

App Designer es una herramienta de Matlab que permite crear aplicaciones con interfaces gráficas de usuario (GUI) de forma sencilla y pudiendo utilizar el lenguaje de programación de Matlab utilizando sus cualidades de entorno de desarrollo integrado (IDE) y, por lo tanto, aprovechando todas sus capacidades para procesamiento de señales e imágenes [13]. También es compatible con funcionalidades de Toolbox de Matlab, por lo que sus aplicaciones son muy amplias.

Su funcionamiento se basa en una biblioteca de componentes para la interfaz de usuario, como botones, paneles, ejes, deslizadores, botones giratorios, cuadros de texto, menús contextuales, etc. Estos componentes se pueden personalizar visualmente, así como programar su funcionamiento mediante líneas de código. Además, se pueden vincular a funciones de llamada a la acción (*Callbacks*) para determinar la lógica que invocan cuando se

realiza una acción determinada, como por ejemplo mostrar una imagen cuando se pulsa un botón. En la Figura 12 se muestran algunos componentes incluidos en la biblioteca de App Designer.

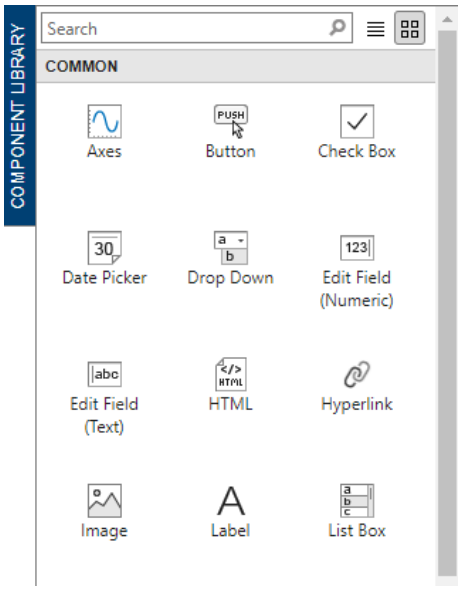

Figura 12. Ejemplo de la biblioteca de componentes de App Designer.

Principalmente, App Designer cuenta con dos ventanas, *Design View* (vista de diseño) y *Code View* (vista de código).

Design View es la ventana destinada al diseño de la interfaz gráfica de usuario, donde se puede visualizar tanto la figura principal de la aplicación en el centro, como los menús de biblioteca de componentes (*Component Library*) a la izquierda, como el buscador de componentes (*Component Browser*) a la derecha. Además, se pueden visualizar las propiedades del componente seleccionado y editar sus campos, también situado a la derecha. En la Figura 13 se muestra una ventana de App Designer en la vista de diseño.

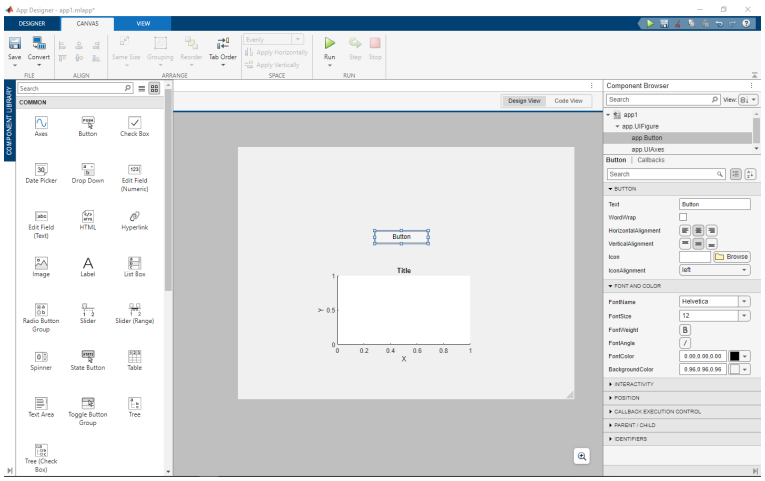

Figura 13. Design View en App Designer.

Code View es la ventana en la cual se implementa toda la lógica de la aplicación. En ella se encuentran el buscador de código (*Code Browser*) a la izquierda, que sirve para buscar

callbacks o funciones, además del *Component Browser* y las propiedades, al igual que en la vista de diseño situados a la derecha. En la Figura 14 se muestra la vista de código.

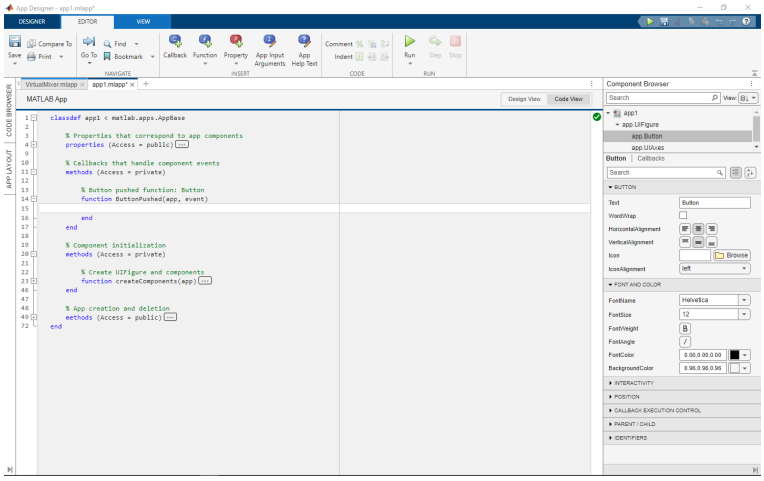

Figura 14. Code View en App Designer.

2.4.1 Funciones en App Designer

En cuanto a la programación de la aplicación, cuenta con un código fuente en el que se pueden implementar diferentes funciones, tanto funciones creadas por el desarrollador como funciones específicas creadas por *App Designer*.

Constructor

Esta función se llama automáticamente al lanzar la ejecución de la aplicación y tiene como nombre el mismo que la aplicación, en este caso "`VirtualMixer`". Es la encargada de construir y configurar los elementos necesarios, invocando otras funciones como `createComponents` y `registerApp`.

`createComponents`

Es la encargada de crear y configurar los componentes de la interfaz de usuario. Sus principales tareas son la creación de la figura principal de la interfaz (`UIFigure`), que es la ventana que contiene todos los elementos de la GUI. Además, se definen y configuran los componentes de la interfaz, como botones, cuadros de texto, menús, etc. estableciendo sus propiedades como posición, tamaño, color, etc. También realiza la asignación de las funciones *callback* a sus componentes correspondientes. La inicialización de variables y lógica inicial se puede implementar en esta función o crear otra función específicamente para ello.

`registerApp`

Se encarga de registrar la aplicación creada junto con su figura principal (`app.UIFigure`), es decir, establece la conexión entre la lógica de la aplicación y su interfaz de usuario.

`delete`

Es una función que se ejecuta justo antes de terminar la ejecución de la aplicación y sirve para eliminar la interfaz de usuario, así como la finalización de tareas pendientes o la liberación de recursos utilizados.

Callbacks

En App Designer se pueden asignar funciones especiales llamadas *callback* a componentes de la interfaz de usuario con el fin de establecer la lógica que se ejecutará cuando se produce un evento en dicha componente. En estas funciones se puede implementar una lógica con toda la complejidad que se requiera, por ejemplo, la llamada a funciones propias de Matlab o a funciones creadas en ficheros externos.

App Designer, al igual que Matlab, también cuenta con herramientas para depuración y corrección de errores. Además, cuenta con una herramienta para crear un instalador ejecutable en cualquier versión de Matlab igual o superior a R2016a, aunque es recomendable el uso de la última versión disponible.

2.5 Resumen de proyectos anteriores

Este proyecto se puede considerar una continuación del proyecto de fin de grado realizado por Jiajie Chen en 2021 titulado "Implementación de herramientas virtuales de aprendizaje para control de cámara y mezcla/efectos de vídeo" [1], concretamente de la parte relativa a la herramienta "Virtual Mixer", que consistía en la migración de dos aplicaciones creadas por el profesor Luis Ortiz Berenguer en 2014 en el entorno GUIDE de Matlab, a App Designer.

2.5.1 Proyecto original

En el proyecto de Luis Ortiz, una herramienta replica una Unidad de Control de Cámara (CCU) entonces usada en el laboratorio de la asignatura de Ingeniería de Vídeo. La otra recrea un mezclador inexistente, con una interfaz puramente didáctica, con la misión de facilitar a los alumnos la comprensión y práctica en estos equipos antes de ir presencialmente al laboratorio de la asignatura.

En estas aplicaciones, las imágenes de entrada están en formato YUV, que consiste en tres planos (Y, luminancia, U y V, crominancia) organizados de forma secuencial en un único *stream* (flujo) de píxeles, y con un tamaño de 1920x1080 píxeles. Ambas herramientas cuentan con un WFM para visualizar distintos niveles y características de la señal en modo *Parade*, *Vector* y *Gamut*.

Aplicación VisionCCU

Esta aplicación simula una unidad de control de cámara, concretamente el modelo RM-P210 de la marca JVC. Cuenta con algunos controles básicos presentes en el equipo original, como se puede ver en la Figura 15 donde se muestra la ventana de la aplicación.

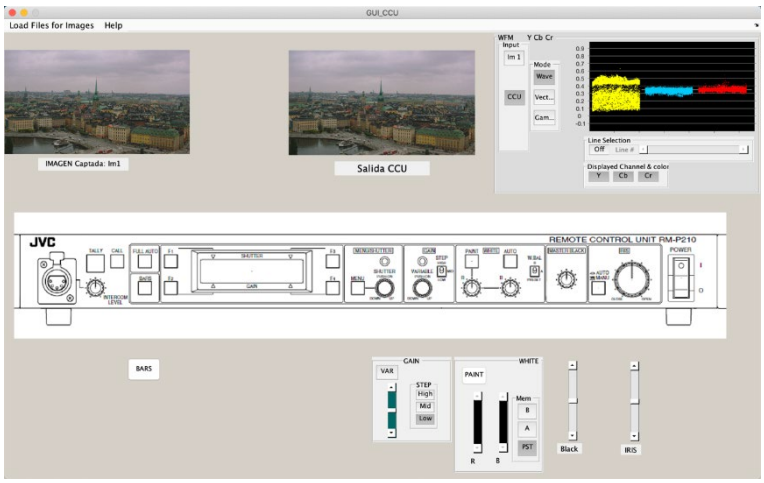

Figura 15. VisionCCU [1].

Aplicación VisionMixer

Esta aplicación simula un mezclador de vídeo básico, con un circuito de doble multiplicador y transiciones como mezcla, cortinilla y *Luma Key*. Dispone de una matriz de entrada y un Key Bus de dos señales. También cuenta con dos ventanas para visualizar la señal correspondiente a cada entrada, otra para visualizar la máscara C y una última ventana de programa, que muestra la imagen de salida. Se puede ver una captura de la aplicación en la Figura 16, en la cual se muestra un diagrama del doble multiplicador para mejorar su uso didáctico.

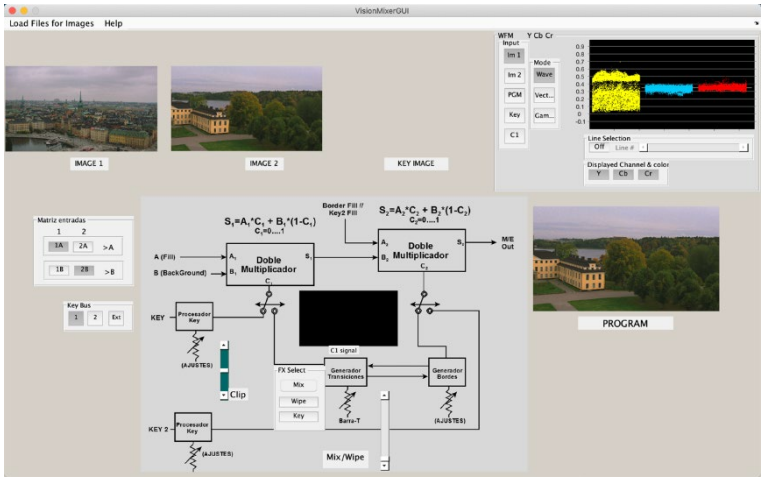

Figura 16. VisionMixer [1].

2.5.2 Proyecto de Jiajie Chen

Jiajie Chen, aparte de migrar ambas aplicaciones del entorno GUIDE a App Designer, adaptó y mejoró la interfaz de usuario y la usabilidad para facilitar el aprendizaje de los alumnos. Además, amplió las funcionalidades originales para hacer las herramientas más completas. Se exponen las mejoras realizadas en “VisionMixer”, ya que da lugar al “VirtualMixer”, que fue la base para el proyecto presentado aquí:

- Compatibilidad con más formatos de imagen: Además del formato YUV, se añade la compatibilidad con PNG, JPG y BMP.
- Transiciones: Se añaden dos nuevas cortinillas: vertical y diagonal.
- *Keying*: Se mejora el *Luma Key* (ya existente) y se añade *Chroma Key*.
- Monitor multivista (*Multiview*): Se organiza la visualización de las señales de entrada y salida de forma más parecida a como se hace en un monitor *multiview* profesional.
- Se añade señal de test a los tres buses de entrada.

2.5.3 Estudio y análisis de la herramienta existente y su código

Para abordar la actualización de la aplicación "VirtualMixer", en primer lugar, se hace un estudio de la lógica implementada con el fin de comprender su funcionamiento y tomar las decisiones adecuadas.

En primer lugar, se percibe que el procesamiento de la señal se hace simulando un flujo de vídeo YUV, y se hace una simulación de la digitalización de una señal analógica. Además, las imágenes se almacenan separadas por componentes Y, Cb y Cr, en lugar de mapas de bits, todas en variables globales para facilitar el acceso a ellas desde cualquier función durante su ejecución.

También se observa que el código del WFM es especialmente extenso y complejo y no hace uso de las capacidades de Matlab para analizar y representar cualidades de las imágenes, además de poco eficiente computacionalmente.

Además, se encuentra que en muchos cálculos no se hace uso de las capacidades de Matlab para trabajar con matrices, así como tampoco se hace uso de funciones ya existentes en Matlab. Tampoco se hace uso de la capacidad de Matlab de implementar funciones en ficheros independientes, lo cual es muy útil para organizar mejor el código.

Por último, como resultado de la migración, se detecta mucha redundancia en el código, lo cual dificulta la comprensión y actualización del código. Además, al realizar la migración de dos herramientas complejas de forma simultánea, no queda tiempo de realizar un proceso de optimización en la ejecución de la aplicación.

3. Especificaciones y restricciones de diseño

Las especificaciones que debe cumplir el proyecto se detallan a continuación:

- La aplicación debe simular un mezclador de vídeo mezcla/efecto (M/E) básico con sus principales funcionalidades:
    - Entradas: Cámara 1 y 2, que soportan tanto imágenes fijas (JPG, PNG, y BMP) como vídeos (mp4, mov y avi), además de entrada de captura de la *webcam.*
    - Buses A, B y auxiliar.
    - Dos dobles multiplicadores encadenados para añadir bordes.
    - Transiciones: Fundido (*Mix*) y cortinilla (*Wipe*)
    - *Chroma Key* y *Luma Key*.
    - Monitores: Programa (PGM), previsualización (PVW), entradas (Cam1, Cam2, *Test*, *Webcam*).
    - Procesador *Down Stream Keyer* (DSK) con segundo doble multiplicador: Mosca, subtítulos, fundido a negro (*Fade to Black*).
    - Carga, reproducción y composición de vídeo.
    - Generación de bordes.
- La herramienta debe disponer de funcionalidades extra para facilitar su comprensión:
    - Monitor de forma de onda (WFM):
        - Entradas (*Input*): Cam1, Cam2, PGM, Key, C1, C2.
        - Modo (*Mode*): Forma de onda de rojo, verde y azul (*RGB Parade*), forma de onda de luminancia y crominancias (*YCbCr Parade*), Vector y *Gamut*.
        - Selector de línea (*Line Selection*).
- La aplicación debe funcionar en cualquier ordenador con Matlab instalado.
- La aplicación se debe poder visualizar correctamente en cualquier tamaño y resolución de pantalla.
    - La aplicación debe funcionar correctamente tanto en pc con Windows como en Mac.

4. Descripción de la solución propuesta

A continuación, se detalla la solución llevada a cabo en la aplicación "VirtualMixer" en distintas fases, empezando por la depuración y corrección de errores, optimización, mejoras, y terminado por la implementación de nuevas funcionalidades.

4.1 Depuración y corrección de errores

4.1.1 Situación inicial

Antes de exponer los cambios realizados se realiza un análisis de algunas funciones existentes para poner en contexto los cambios realizados.

`startupFcn`

Es la primera función que se invoca al ejecutar la aplicación y se encarga de inicializar la ejecución de la aplicación. Las operaciones que realiza esta función son:

- Inicializar variable chroma (color para el C*hroma Key*), la variable C (máscara) y diferentes variables para la representación del WFM.
- Configuración de los ejes del WFM.
- Cargar imágenes yuv, separar sus componentes en matrices, reescalar e interpolar componentes de color.
- Calcular imagen de salida mediante operaciones de doble multiplicador.
- Construir una imagen de barras de color del 75%, convertirla a YCbCr y mostrarla en el monitor de test.
- Inicialización de visualización del WFM, creando fondo, retículas y ejes.
- Llamada a la función "`wfmshow`".

`wfmshow`

Es la función encargada de la gestión y el cálculo del monitor de forma de onda, es una función con una gran cantidad de líneas de código, mucho código redundante y poco eficiente. Se encarga de realizar una representación de un monitor de forma de onda, vectorscopio y visualización en diamante (Gamut) representando punto a punto el valor de cada píxel de la imagen, así como de las retículas.

`Mezcla`

Contiene todo el código necesario para gestionar la parte de la interfaz de usuario relativa a la transición de Mezcla, así como el código para calcular la imagen de salida, incluyendo los cálculos del doble multiplicador. Además, también contiene el código de la transición cortinilla.

`Threshold`

Esta función contiene el código de la transición *Luma Key*, incluyendo la gestión de la interfaz de usuario, el cálculo del umbral y la máscara, así como la obtención de la imagen de salida realizando, de nuevo, las operaciones del doble multiplicador.

`ChromaKey`

La función ChromaKey se encarga de la transición *Chroma Key*, para lo cual descompone el color de *key* guardado en la variable "`chroma`" en componentes RGB y, a partir de estas, realiza las operaciones para obtener las componentes YCbCr. A continuación, realiza las operaciones para obtener la máscara C a partir de las componentes de las imágenes de entrada y obtiene la imagen de salida realizando las operaciones del doble multiplicador. Por último, muestra la imagen de salida en el monitor de programa.

4.1.2 Eliminación de simulación de *stream* de vídeo YUV

Para eliminar la simulación de un *stream* de vídeo YUV, que se ha considerado innecesaria al trabajar directamente con imágenes en mapas de bits, además de modificar distintas partes del código que se explican a lo largo de todo el apartado se han eliminado las siguientes funciones.

Función `Comptes444`

Esta función realiza la conversión de YCbCr a RGB según la Recomendación UIT-R BT.2087 [14] que describe las fórmulas para la conversión de color de la Recomendación UIT-R BT.709 [15] a la Recomendación UIT-R BT.2020 [16], que utiliza la matriz de conversión mostrada en (2)

$$\begin{bmatrix} E'_R \\ E'_G \\ E'_B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1.5748 \\ 1 & -1.1873 & -0.4682 \\ 1 & 1.8556 & 0 \end{bmatrix} \begin{bmatrix} E'_Y \\ E'_{Cb} \\ E'_{Cr} \end{bmatrix} \quad (2)$$

Que, además, tiene que en cuenta la cuantificación recogida en la Recomendación 601 (ITU-R BT.601) [17], que especifica, según indica en el Cuadro 3, que la señal debe estar cuantificada entre los valores 16 y 235 cuando se codifica con 8 bits. Esta conversión la hace para que las imágenes puedan ser representadas por Matlab, sin embargo, al trabajar directamente con mapas de bits ya no es necesaria.

Función `RGB_to_YCbCr444`

Esta función convierte una imagen mapa de bits RGB a una imagen YCbCr 4:4:4 utilizando las fórmulas de la Recomendación UIT-R BT.709 [15], según (3), (4) y (4), una vez normalizadas las componentes RGB entre 0 y 1.

$$\mathrm{Y} = 0.2126 \cdot \mathrm{R} + 0.7152 \cdot \mathrm{G} + 0.0722 \cdot \mathrm{B} \quad (3)$$

$$\mathrm{Cb} = -0.1146 \cdot \mathrm{R} - 0.3854 \cdot \mathrm{G} + 0.5 \cdot \mathrm{B} \quad (4)$$

$$\mathrm{Cr} = 0.5 \cdot \mathrm{R} - 0.4542 \cdot \mathrm{G} - 0.0458 \cdot \mathrm{B} \quad (5)$$

Y, a continuación, los valores obtenidos se escalan y desplazan para ajustarlos a los rangos típicos de YCbCr: Y entre 16 y 235, y Cb y Cr entre 16 y 240 y desplazados para centrarlos en el valor 128.

No obstante, en Matlab ya existe una función llamada "`rgb2ycbcr`" que realiza esta conversión, por lo tanto, tampoco es necesaria.

4.1.3 Separación de funciones y código principal

Debido al funcionamiento de la herramienta *App Designer* y de la complejidad de la aplicación, el fichero principal cuenta con un elevado número de líneas de código y funciones muy extensas, por lo que se ha optado por crear funciones externas para procesos y cálculos que se repiten, así como extraer funciones ya existentes a ficheros distintos del principal, aprovechando las ventajas de Matlab de gestionar funciones en ficheros independientes.

Se ha intentado mantener en el fichero principal únicamente el código para la gestión de la interfaz de usuario, por ello las funciones ejecutadas por *callback* se han dejado en el fichero principal, ya que son funciones que responden a eventos y por lo tanto deben estar en dicho fichero, como por ejemplo las funciones que responden ante la pulsación de un botón, la modificación de un valor o la propia función de inicio de la aplicación, llamada "`startupFcn`".

4.1.4 Modificación de funciones existentes

Las funciones modificadas son las siguientes

Función `startupFcn`:

En primer lugar, se han eliminado todas las variables globales que ya no se usan, así como su inicialización. Estas variables correspondían al WFM original de Luis Ortiz, así como a la simulación del stream YUV. Las variables eliminadas son todas las referentes a las componentes de las imágenes de entrada y salida llamadas Y Cb Cr seguido de un identificador, las retículas y rejillas del WFM, así como las componentes de la imagen de las barras de test.

En su lugar, se han creado variables globales que almacenan las imágenes de entrada y de salida en mapas de bits (bitmap) con las tres componentes, las máscaras, el color para el *Chroma Key*, así como variables para el funcionamiento de las distintas funcionalidades añadidas como los anchos del borde y del suavizado de las máscaras, los valores de tolerancia para el *Chroma Key*, la selección de línea, etc., además de inicializar sus valores.

Se ha modificado la carga de imágenes, simplificándola, ya que ahora solo es necesario cargar la imagen inicial y adaptar su resolución mediante la función "`adapta_resolucion`", que se explica en el apartado 4.2.1.

Por último, mediante la función "`CreaImagenBarras`", que se explica en el apartado 4.1.5, se crea una imagen de barras de test del 75% y se muestran todas las imágenes en sus respectivos monitores. Además, se llama a la función "`wfm`", que se explica en el apartado 4.3.1.

Función Mezcla

Esta función se ha separado en varias funciones independientes. En primer lugar, se ha creado una nueva función llamada "`Wipe`", donde se ha implementado el código correspondiente a la funcionalidad de cortinilla, como se detalla en el siguiente apartado. Además, se ha creado otra función llamada "`DobleMultiplicador`" donde se ha codificado la lógica para el cálculo relativo al doble multiplicador, que se detallada en el apartado 4.2.3. También se ha eliminado la representación del WFM, cuya tarea ahora es realizada por la función "`actualizarSalida`", explicada más adelante en este mismo apartado. En este caso, el cálculo de la máscara necesaria para esta transición se ha dejado en esta función debido a su simplicidad, ya que es tan solo una línea de código.

Por lo tanto, la función "`Mezcla`" actualmente se ocupa de llamar a la función "`DobleMultiplicador`" para obtener la imagen de salida en función de las entradas elegidas en los buses A y B en la interfaz de usuario y de la máscara obtenida a partir del valor del *fader*, además de representar la imagen de salida en el panel de programa y la máscara en el panel C1.

Función `Threshold`

La función "`Threshold`" se ha modificado para eliminar las líneas de código que calculan la máscara para el *Luma Key*, dejado únicamente la gestión de las señales a partir de la cual se calcula la máscara y las imágenes que se mandan al doble multiplicador para obtener la imagen de programa. El código que calcula la máscara se ha movido a una nueva función llamada "`MascaraLuma`" que se detalla en el siguiente apartado.

Para ello, en primer lugar, calcula el umbral normalizando el valor del *fader* y multiplicándolo por 255 para obtener un valor entero de 8 bits. A continuación, en función de la señal seleccionada en el bus auxiliar, invoca la función "`MascaraLuma`" pasándole como variables de entrada la imagen seleccionada y el valor del umbral calculado. Posteriormente, dependiendo de las señales seleccionadas en los buses A y B llama a la función "`DobleMultiplicador`" para calcular la imagen de salida, pasándole la máscara y las dos imágenes seleccionadas. Por último, muestra la imagen de salida en el panel de programa y la máscara en la pantalla C1.

Función `actualizarSalida`

La función "`actualizarSalida`" se encarga de actualizar la imagen mostrada en programa dependiendo del tipo de transición seleccionada en la interfaz de usuario. Se ha actualizado para añadir la llamada a la nueva función "`Wipe`" cuando está seleccionada la opción "`Cortinilla`" en la interfaz, además de las funciones "`Threshold`", "`ChromaKey`" y

"`Mezcla`" para las opciones de *Luma Key*, *Chroma Key* y Mezcla respectivamente. Además, se ha añadido una llamada a la nueva función "`ActualizaWFM`", detallada en el siguiente apartado, que se encarga de actualizar el monitor de forma de onda y así eliminar ese código del resto de funciones.

4.1.5 Creación de nuevas funciones de funcionalidades existentes

Función `CreaImagenBarras`

La función "`startupFcn`", codificada por el profesor Luis Ortiz [1], contenía el código para la creación de una carta de barras que se ha extraído a una función en un fichero independiente. Además, se ha modificado para adaptarla al nuevo funcionamiento con imágenes basadas en mapas de bits.

La función, en primer lugar, inicializa los tres planos R, G y B como matrices de tamaño 270 por 480 con todos los valores a 0. Posteriormente calcula los valores de cada una de las componentes y los escribe en la posición correspondiente de cada matriz.

Por último, crea una variable de tres dimensiones que contiene cada una de las componentes R, G y B y convierte todos los valores a números enteros de 8 bits.

El resultado de ejecutar esta función puede verse en la Figura 17.

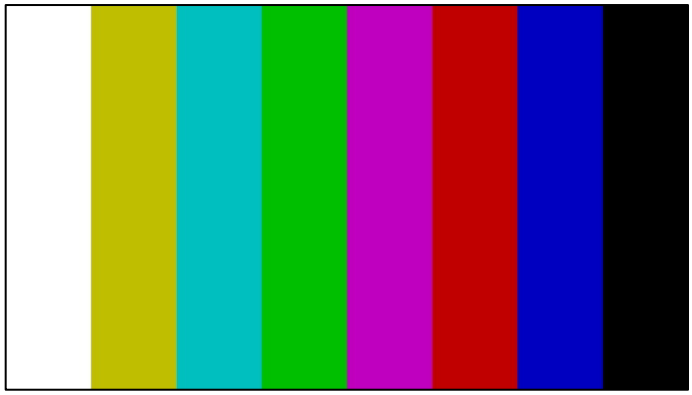

Figura 17. Imagen de barras de test del 75% generada por la función "CreaImagenBarras".

Función `Wipe`

En el código original, las líneas para gestionar las transiciones de cortinilla y calcular las máscaras se encuentran dentro de la función "`Mezcla`". Se ha optado por crear una nueva función llamada "`Wipe`" que realiza la gestión de los tipos de cortinillas e invoca a las funciones necesarias para realizar los cálculos. Esta función decide, en función del tipo de cortinilla seleccionado en la interfaz de usuario, que variables pasar al llamar a la nueva función "`crearMascaraCortinilla`", explicada más adelante.

Además, se ha añadido la opción de añadir borde a las cortinillas y de suavizar el borde, por lo que también se gestionan esas opciones, llamando a las funciones "`crearMascaraBorde`" para crear la máscara C2 y "`suavizarMascara`", que se detallan en el apartado 4.4.1.

Posteriormente, para calcular la imagen de salida se invoca la función "DobleMultiplicador" pasándole como variables de entrada la máscara C1 y las dos imágenes elegidas en los buses A y B en la interfaz de usuario. Si está elegida la opción de suavizar bordes, se vuelve a calcular la imagen de salida con la función "DobleMultiplicador" pero pasándole como entradas la imagen de salida anterior, la máscara C2 y la imagen con el color del borde generada en la función "startupFcn" o al elegir un color en la interfaz de usuario.

Por último, se representan la imagen de salida en el monitor de programa, la máscara C1 en su panel correspondiente y la máscara C2 también en su panel si la opción de borde está activa.

Función MascaraLuma

Se ha creado una función independiente a la cual se ha movido el código relativo al cálculo de la máscara para el *Luma Key*. La función tiene como variables de entrada la imagen a partir de la cual calcular la máscara, y el valor del umbral como número entero de 8 bits, es decir, comprendido entre 0 y 255. Como variable de salida tiene la máscara calculada, que es una imagen de 480 x 270 píxeles con valores 0 o 1.

Para calcular la máscara, en primer lugar, pasa la imagen de RGB a Y Cb Cr y extrae la componente de luminancia. Después, inicializa la máscara como una matriz de 270 filas y 480 columnas, con todos los valores a cero. A continuación, busca los píxeles que superan el umbral mediante la función "find", y, por último, mediante la función "ind2sub" y haciendo uso de la capacidad de operar con matrices de Matlab, da el valor 1 a los píxeles de la máscara que superan el umbral en la imagen original.

Función ActualizaWFM

Como se ha comentado anteriormente, se ha creado una función llamada "ActualizaWFM" para actualizar la representación mostrada en el monitor de forma de onda cuando se producen cambios, y así eliminar el código correspondiente a esta tarea del resto de funciones.

En primer lugar, comprueba si en la entrada del WFM está seleccionado programa, máscara 1 o máscara 2.

Si la selección está en programa, comprueba si está en modo *Parade* YCbCr o RGB, en modo Vector o en modo Gamut, y, en función de eso, llama a las funciones correspondientes ("wfm", "vector" o "gamut") pasándole como variable de entrada la imagen de salida del primer doble multiplicador o del segundo dependiendo de si la opción "bordes" está activada.

En el caso en el que en la entrada esté seleccionada una de las máscaras, llama a la función "wfm" con la máscara correspondiente como variable de entrada.

Función `crearMascaraCortinilla`

Los cálculos para obtener la máscara para las cortinillas se han movido a una función en un fichero independiente, además se ha modificado el código de las cortinillas horizontal y vertical para adaptarlos a la nueva resolución de trabajo, y también se ha modificado el código relativo a la cortinilla diagonal para corregir un fallo que impedía que la cortinilla recorriera toda la imagen.

Además, se han añadido nuevas cortinillas, que se detallan en el apartado 4.2.2.

4.1.6 Modificación Chroma Key

La funcionalidad de *Chroma Key* se ha modificado por completo, tanto a nivel de código como la interfaz de usuario, con la finalidad de hacerla más completa, así como más didáctica.

Interfaz de usuario

La interfaz de usuario de la sección relativa al *Chroma Key* se ha modificado, eliminando la selección de color mediante un menú emergente, colocando en su lugar tres ruedas para seleccionar el matiz (*hue*), la saturación (*saturation*) y el brillo (*value*). También se han añadido campos numéricos para estos mismos ajustes, además de tres deslizadores para modificar la tolerancia de cada valor. Por último, se ha añadido una visualización del color seleccionado como *key*.

Función "`ChromaKey`"

Esta función se ha modificado, dejando únicamente el código que se encarga de la gestión de la interfaz de usuario y de las imágenes de entrada y salida, así como de su representación. Para ello hace uso de las funciones "`DobleMultiplicador`" y "`mascaraChoma`".

El código que calcular la máscara se ha eliminado, y en su lugar se ha creado una nueva función llamada "`mascaraChroma`", que se detalla a continuación.

Función "`mascaraChroma`"

Se ha creado una función para calcular la máscara para el *Chroma Key* a partir de una imagen, el color del *key* y unos valores de tolerancia para el tinte (*Hue*), la saturación (*Saturation*) y el brillo (*Value*).

La función tiene como variables de entrada la imagen a partir de la cual calcular la máscara, "`keyColor`" que es el color en un vector con las componentes R, G y B, y tres variables ("`toleranciaHue`", "`toleranciaSat`" y "`toleranciaVal`") con cada uno de los valores de tolerancia en grados para el valor *Hue* y en porcentaje para los valores de saturación y brillo, y como variable de salida una matriz con la máscara calculada.

En primer lugar, se normalizan los valores de tolerancia mediante (6), (7) y (8) y se convierte la imagen a formato `double`. Además, se extrae el alto y el ancho de la imagen en otras dos variables y se inicializa la máscara con una matriz del mismo alto y ancho que la imagen y con

todos los valores iguales a 1. A continuación, se convierten tanto la imagen como la variable del color *key* a HSV (*Hue, Saturation, Value*).

$$toleranciaHue_{norm} = toleranciaHue/360 \quad (6)$$

$$toleranciaSat_{norm} = toleranciaSat/100 \quad (7)$$

$$toleranciaVal_{norm} = toleranciaVal/100 \cdot 255 \quad (8)$$

Posteriormente, se calcula el límite inferior del *Hue* restando el primer valor del vector del color en formato HSV, que corresponde con el valor *Hue* del color *key*, con la variable de la tolerancia del *Hue*. Si el límite inferior es menor que 0 se resta 1 menos el valor del límite inferior, ya que como el *Hue* se mide en grados, se considera que el valor ha dado una vuelta entera. También se pone como verdadera una variable booleana para indicar que se ha superado el límite inferior.

Con el límite superior se realiza el mismo procedimiento, se suma el primer valor del vector del color en formato HSV y si el resultado es mayor de 1 se suma 0 más la variable del límite superior. En este caso se pone como verdadera una variable booleana para indicar que se ha superado el límite superior. En la Figura 18 se hace una representación gráfica de un caso en el que, considerando un determinado ángulo *hue*, y una tolerancia, el rango es superado, por lo que es necesario considerar el límite al otro lado del 0. En la imagen se muestra el ángulo *hue* en verde, los valores dentro de la tolerancia en azul y la parte de esos valores que superan el rango en rojo.

Para los límites inferior y superior de la saturación y el brillo se resta y se suma las componentes de saturación y brillo de la variable del color *key* con los valores de tolerancia de saturación y brillo respectivamente, y, mediante las funciones de Matlab “min” y “max”, se truncan los resultados para que queden comprendidos entre 0 y 1.

Por último, se comparan los píxeles de cada componente de la imagen con los límites superior e inferior correspondientes y, si están dentro de dichos límites, se pone el píxel de la misma posición de la máscara a cero.

Figura 18. Fuera de rango del ángulo hue [18].

4.2 Creación de nuevas funcionalidades en la aplicación

4.2.1 Admisión de imágenes en cualquier resolución y formato

Con el fin de hacer más versátil la herramienta se ha modificado para admitir imágenes de cualquier resolución y formato, para ello se ha creado la función "`adapta_resolucion`", que analiza la relación de aspecto de la imagen de entrada y la adapta a la relación de aspecto 16:9, que es a la que trabaja internamente la aplicación y a la que se muestran las imágenes.

En primer lugar, obtiene el alto y el ancho de la imagen de entrada, a continuación, calcula la relación de aspecto dividiendo el alto entre el ancho.

A continuación, compara la relación de aspecto de la imagen original con la relación de aspecto establecida y, si la relación de aspecto de la imagen es mayor que la establecida calcula un nuevo ancho, que es el alto establecido dividido entre la relación de aspecto de la imagen, y si no, calcula un nuevo alto que es el ancho establecido dividido entre la relación de aspecto de la imagen.

Posteriormente redimensiona la imagen original teniendo en cuenta el nuevo ancho y alto.

Por último, si la relación de aspecto es diferente de la establecida añade fondo negro para rellenar los huecos.

4.2.2 Nuevos tipos de cortinillas

Para poder estudiar más detalladamente las funcionalidades de un mezclador de vídeo se han añadido varios tipos de cortinillas que comúnmente están presentes en estos equipos.

Cortinilla cuadrada

Consiste en una transición basada en la figura de un cuadrado, como puede verse en la Figura 19. Para implementarla, dentro de la función "`crearMascaraCortinilla`", en primer lugar, se inicializa la máscara con valores igual a cero y a continuación se da el valor 1 a todos los puntos desde el centro hasta la variable "p" que determina la posición de la transición en ambos ejes. Los rangos horizontales y verticales se detallan en (9) y (10)

$$rango\ vertical\colon \left[\frac{alto}{2} - p, \frac{alto}{2} + p\right] \tag{9}$$

$$rango\ horizontal\colon \left[\frac{ancho}{2} - p, \frac{ancho}{2} + p\right] \tag{10}$$

Cortinilla en cruz

Esta transición se basa en una cruz que parte del centro de la imagen y recorre ambos ejes hasta el borde, y que a medida que aumenta la posición determinada por el *fader*, va aumentando de grosor.

Para implementarla, una vez inicializada la máscara con valores iguales a cero, en primer lugar, se modifican todos los píxeles en el eje horizontal desde el centro hasta la posición determinada por el *fader* estableciendo su valor a 1, y, a continuación, se hace la misma operación con los píxeles del eje vertical. Los rangos horizontales y verticales se detallan en (11) y (12)

$$rango\ vertical\colon \left[\frac{alto}{2} - \frac{p}{2}, \frac{alto}{2} + \frac{p}{2}\right] \tag{11}$$

$$rango\ horizontal\colon \left[\frac{ancho}{2} - \frac{p}{2}, \frac{ancho}{2} + \frac{p}{2}\right] \tag{12}$$

Cortinilla rombo

La cortinilla rombo se puede considerar una variante de la cortinilla cuadrada, pero rotada 45°. Para implementarla, en primer lugar, se definen dos vectores que contienen las coordenadas de los vértices del rombo calculadas en función del ancho y el alto de la imagen y de la posición de la transición, mediante (13)

$$\begin{aligned} vertices\ x &= \left[\frac{ancho}{2}, \frac{ancho}{2} + \frac{p}{2}, \frac{ancho}{2}, \frac{ancho}{2} + \frac{p}{2}\right] \\ vertices\ y &= \left[\frac{alto}{2}, \frac{alto}{2} - \frac{p}{2}, \frac{alto}{2}, \frac{alto}{2} + \frac{p}{2}\right] \end{aligned} \tag{13}$$

A continuación, se utiliza la función de Matlab "`poly2mask`" para generar la máscara, que convierte un polígono definido por sus vértices en una máscara binaria, pasándole como argumentos de entrada los vectores con las coordenadas de los vértices y el ancho y alto de la máscara.

Cortinilla circular

Esta transición consiste en una circunferencia concéntrica con la imagen, que a medida que aumenta el valor del *fader* aumenta su radio. Para codificarla, en primer lugar, se calculan dos variables con las coordenadas x e y mediante la función "`meshgrid`", que genera matrices de coordenadas en 2 o 3 dimensiones. A continuación, se calcula una matriz con la distancia al centro desde cada píxel mediante (14)

$$d = \sqrt{\left(x - \frac{ancho}{2}\right)^2 + \left(y - \frac{alto}{2}\right)^2} \tag{14}$$

Donde d: distancia al centro

Por último, se crea la máscara comparando los píxeles de la imagen y viendo cuales cumplen la condición d ≤ p (posición de la cortinilla).

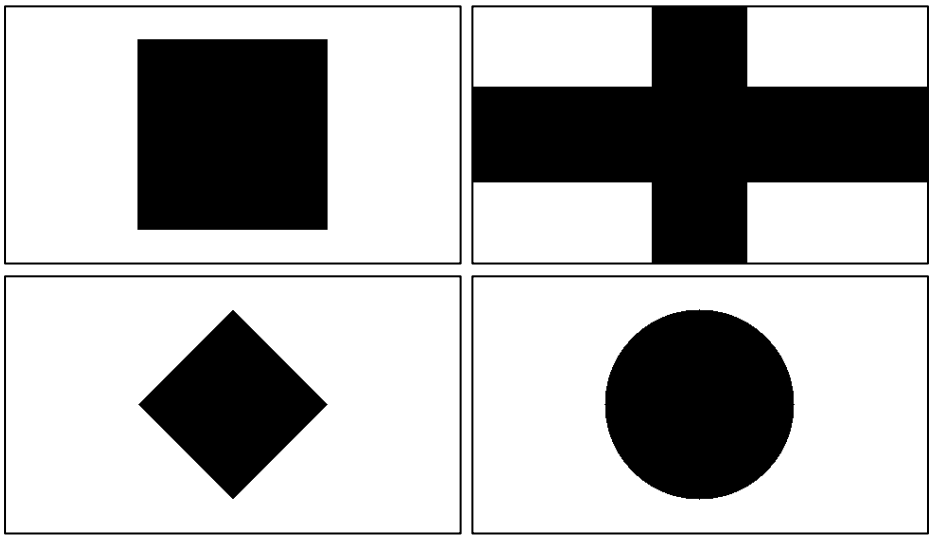

Figura 19. Forma de las máscaras para las nuevas cortinillas. Cortinilla cuadrada (arriba a la izquierda), cortinilla en cruz (arriba a la derecha), cortinilla en rombo (abajo a la izquierda) y cortinilla circular (abajo a la derecha).

4.2.3 Función `DobleMultiplicador`

Se ha creado una función en un fichero independiente para realizar los cálculos del doble multiplicador. La función se ha codificado totalmente nueva para realizar las operaciones con imágenes bitmap.

En primer lugar, la función convierte las tres variables de entrada, A, B y C, a formato `double`. Posteriormente calcula la imagen de salida S mediante (15)

$$S = A \cdot C + B \cdot (1 - C) \tag{15}$$

Por último, convierte la imagen de salida a formato uint8. En la Figura 20 se muestra un diagrama básico del doble multiplicador, donde A y B son las imágenes de entrada, C la máscara y S la imagen de salida.

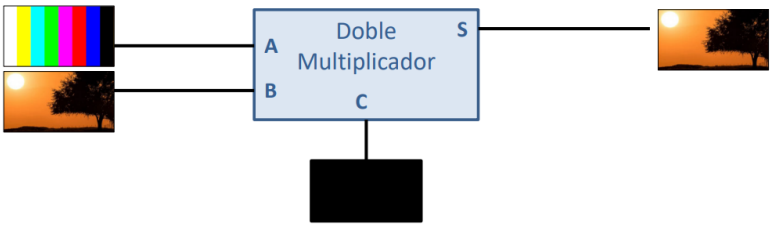

Figura 20. Diagrama del doble multiplicador, donde A y B son las imágenes de entrada, C la máscara y S la imagen de salida [2].

4.2.4 Carga de vídeos

Con el fin de estudiar la viabilidad de incluir la carga y reproducción de vídeos en la aplicación se ha creado una aplicación de prueba que utiliza las mismas funciones implementadas para la aplicación principal, pero con una funcionalidad reducida para facilitar las pruebas.

Interfaz gráfica de usuario

La interfaz de usuario de esta aplicación consiste en una versión simplificada del mezclador de vídeo. Cuenta con dos monitores que visualizan cada una de las cámaras y un monitor de programa, donde se visualiza la salida del mezclador. También dispone de un interruptor de *Play/Stop*, un *fader Mix* para seleccionar la cantidad de mezcla de cada cámara y un WFM con dos interruptores para visualizar *Parade* YCbCr o RGB. En la Figura 21 se puede ver una captura de esta aplicación.

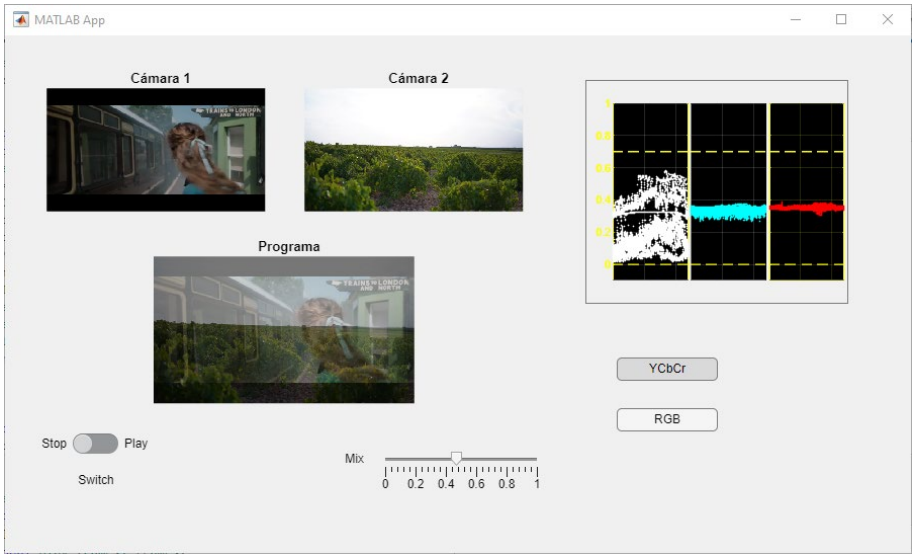

Figura 21. Aplicación de prueba de vídeo.

Funcionalidad

La lógica de esta aplicación es mucho más sencilla para facilitar el estudio y mantener un nivel de eficiencia alto. A continuación, se detalla parte de su programación, obviando la gestión de la interfaz de usuario y analizando únicamente la parte relativa al funcionamiento de la carga, reproducción y procesado de vídeo.

Función `startupFcn`

Esta función se encarga de inicializar la ejecución de la aplicación. Para ello realiza los siguientes pasos:

- Se inicializa una variable booleana llamada "`play`" con valor "`true`".
- Se inicializa otra variable llamada C para almacenar la máscara, que consiste en una matriz con todos los valores a cero de tamaño 270 x 480.
- Se carga dos vídeos de prueba en las variables "`video1`" y "`video2`" mediante la función de Matlab "`VideoReader`".
- Se inicializa una variable llamada "`numFrames`" con valor 12 para determinar el número de fotogramas que se van a utilizar.
- Se inicializan dos variables `cell` llamadas "`frames1`" y "`frames2`".
- Se leen los 12 primeros fotogramas de cada vídeo, se adapta su resolución a la resolución de trabajo mediante "`adapta_resolucion`", detallada en el apartado 4.2.1, y se almacenan en las variables "`frame1`" y "`frame2`".
- Por último, para reproducir los vídeos se llama a la función "`reproductorPGM`", que se detalla a continuación, pasándole como argumentos "`frames1`" y "`frames2`".

Función `reproductorPGM`

Esta función se utiliza tanto para reproducir cada uno de los vídeos en los monitores de cada cámara como para componer el vídeo de salida y reproducirlo en el monitor de programa, además de llamar a la función "`wfm`" para mostrar la forma de onda de cada fotograma en el monitor de forma de onda.

Para ello, se ha codificado un bucle `while`, que se ejecutará mientras la variable "`play`" tenga valor `true`. Dentro de este bucle hay otro bucle `for` que se ejecuta desde 1 hasta el número de fotogramas de los vídeos y que en cada iteración realiza las siguientes operaciones:

- Muestra el fotograma del vídeo 1 correspondiente al número de iteración en el monitor de la cámara 1.
- Muestra el fotograma del vídeo 2 correspondiente al número de iteración en el monitor de la cámara 2.
- Calcula el fotograma de salida de programa mediante la función "`DobleMultiplicador`", ya explicada en el apartado 4.2.3, a partir de los fotogramas correspondientes de ambos vídeos y la máscara C.
- Muestra el fotograma de salida calculado en el monitor de programa.

- Llama a la función “wfm”, explicada en el apartado 4.3.1, para mostrar la forma de onda del fotograma de salida.
- Mediante la función de Matlab “pause” realiza una pausa de 0.25 segundos para evitar la sobrecarga del procesador.

Callback `SwitchValueChanged`

Es una función para gestionar los cambios en el interruptor de *Play/Stop*. Si el interruptor está en la posición *play*, da el valor `true` a la variable “`play`” y ejecuta la función “`reproductorPGM`”, si no, da el valor `false` a la variable “`play`”.

Callback `MixSliderValueChanging`

Es una función para cambiar el valor de la máscara C en función del deslizador *Mix*. Si el valor del deslizador cambia, da ese valor a todos los píxeles de la máscara.

Resultado

Mediante esta versión simplificada del mezclador, con una lógica reducida, se ha conseguido un funcionamiento razonable de la reproducción de dos vídeos, la composición de ambos y la visualización de la forma de onda en tiempo real. Aun así, debido a la falta de tiempo por los problemas surgidos, la complejidad de la aplicación y la dificultad de implementar esta funcionalidad en el código principal no ha sido posible añadir la reproducción de vídeo a la aplicación definitiva. Por lo tanto, queda como trabajo futuro.

4.2.5 Actualizar interfaz y diseño

Se ha actualizado la interfaz de usuario para adaptarla a las nuevas funcionalidades y, además, intentar hacerla más compatible con cualquier resolución de pantalla. Sin embargo, no se ha conseguido que mantenga la coherencia al cambiar el tamaño de la pantalla principal. Cuando se cambia el tamaño de la pantalla principal algunos elementos se escalan y otros no, por lo que cambian su posición en el espacio, separándose o juntándose unos de otros pudiendo llegar a superponerse y hacer inviable su manejo. Por lo tanto, se ha decidido desactivar la propiedad de “Resize” del elemento `UIFigure`, que es la figura principal de la aplicación, para deshabilitar el cambio de tamaño de la ventana y así evitar estos problemas.

También se ha modificado el diseño para darle una apariencia más parecida a un mezclador de vídeo profesional, eliminando el fondo con el esquema de funcionamiento y eligiendo colores más oscuros.

Como se puede ver en la Figura 22, se han movido los buses de entrada y auxiliar y se han colocado juntos arriba a la izquierda. Debajo de los buses se han puesto las visualizaciones de las máscaras C1 y C2. Las pestañas que contienen las transiciones se han mantenido con la misma organización y posición, pero se ha ampliado su tamaño. Además, el deslizador para ajustar el nivel del *Luma Key* se ha cambiado por una rueda y se ha colocado en la pestaña de *Luma Key*. En la pestaña del *Chroma Key* se han colocado los controles nuevos. También se ha

añadido una pestaña nueva llamada “Bordes” junto a la de “Mezcla” y “Cortinilla”, donde se han implementado dos deslizadores, “Ancho” para elegir el ancho del borde y “Suavizado” para elegir el tamaño del suavizado de los bordes. El WFM se ha dejado en la misma posición, tan solo ajustando los tamaños y añadiendo la pestaña “*Parade RGB*”. Por último, el *fader* se ha colocado entre las pestañas de las transiciones y el WFM, haciéndolo más grande debido a su importancia.

Para mantener el carácter didáctico de la herramienta, en la pestaña “Ayuda” situada en el menú superior se han incluido tanto las instrucciones del mezclador como una breve explicación de funcionamiento. Además, se ha añadido el diagrama de bloques para poder consultarlo en cualquier momento.

Por último, se ha añadido el logo de la asignatura de Ingeniería de Vídeo. En la Figura 22 se muestra el resultado obtenido.

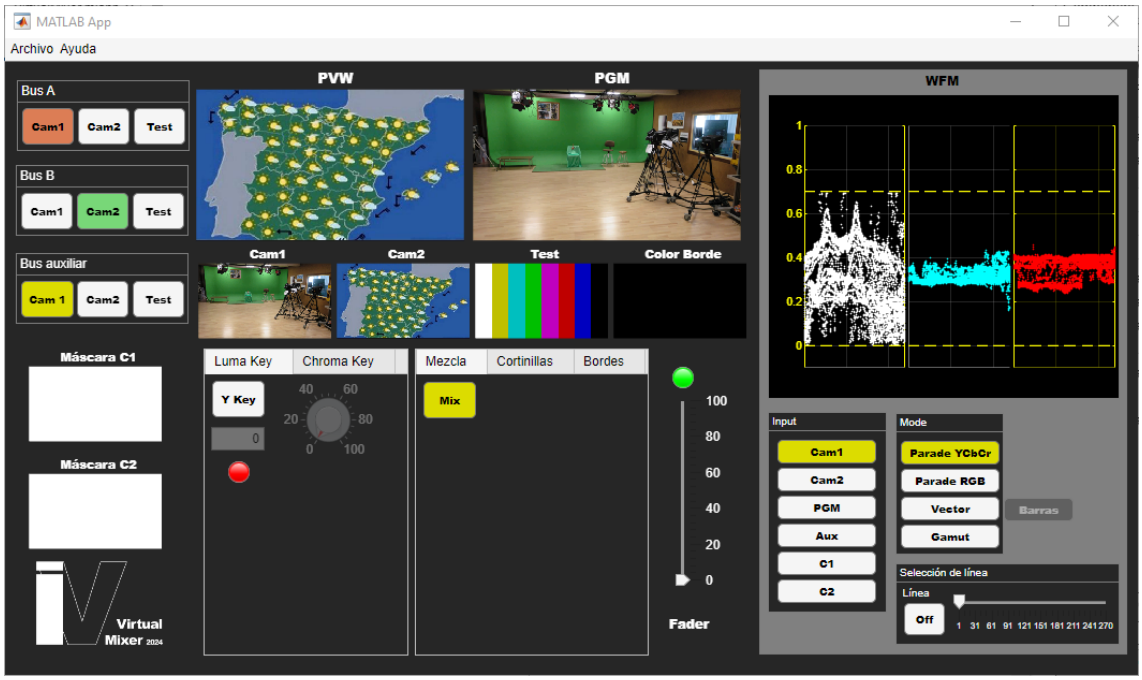

Figura 22. Interfaz gráfica de usuario de la aplicación.

4.3 Monitor de forma de onda nuevo

Con el fin de optimizar tanto el funcionamiento de la herramienta como el código, se ha implementado el monitor de forma de onda totalmente desde cero.

El código anterior, además de realizar la conversión de la simulación analógico-digital comentada anteriormente, analiza el valor de cada uno de los píxeles línea por línea y dibuja punto a punto formando la imagen del WFM, para lo cual hace uso de varios bucles, dando como resultado una función con muchas líneas de código y poco eficiente computacionalmente. Además, tanto el modo “*Parade*” como los modos “Vector” y “Gamut” están implementados en la misma función dentro del mismo fichero.

Para solucionar esto se han codificado tres funciones totalmente nuevas en ficheros diferentes y haciendo uso de funciones y operaciones matriciales de Matlab.

4.3.1 Función `WFM`

Por un lado, se ha implementado una función llamada "`wfm`" que se encarga de calcular y mostrar en el panel correspondiente los modos "*Parade YCbCr*" y "*Parade RGB*", que no existía en la versión anterior.

Para ello, en primer lugar, se borra la visualización anterior presente en el panel. A continuación, se comprueba si se ha recibido una imagen nueva, en cuyo caso se reemplaza la imagen guardada en caché, o no, en cuyo caso no se hace nada. Posteriormente, se comprueba si la imagen recibida tiene tres componentes, como es el caso de una imagen a color, o sólo una, como es el caso de las máscaras. Posteriormente se comprueba si el modo elegido es "*Parade* YCbCr" o "*Parade* RGB".

En el caso de "*Parade* YCbCr", en primer lugar, se crean tres variables para especificar el color que debe tener la gráfica correspondiente a cada componente. Estás variables son:

- color1 (Y): blanco
- color2 (Cb): *cyan*
- color3 (Cr): rojo

Después, se convierte la imagen de RGB a Y Cb Cr y se crean otras tres variables, donde se extraen las tres componentes:

- comp1: componente Y
- comp2: componente Cb
- comp3: componente Cr

Por último, se normalizan los valores de las componentes entre 0 y 0,7.

En el caso de "*Parade* RGB" se especifican los colores que tendrá la gráfica en las mismas variables, en este caso utilizando su color para cada componente:

- color1 (R): rojo
- color2 (G): verde
- color3 (B): azul

Se extrae cada una de las componentes:

- comp1: componente R
- comp2: componente G
- comp3: componente B

Se normalizan los valores de las componentes entre 0 y 0,7.

Para representar los valores, en primer lugar, se crean tres ejes donde se mostrarán cada una de las componentes. A continuación, mediante la función "`plot`" de Matlab se dibujan los valores de las tres componentes (comp1, comp2 y comp3) con los colores elegidos anteriormente (color1, color2 y color3), utilizando únicamente la primera línea de cada bloque de 12 líneas para disminuir la carga computacional y acelerar el proceso. Por último, se muestran los ejes verticales, la rejilla, las etiquetas de los valores en el primer eje y una línea discontinua representando los valores 0 y 0,7. En el caso de que la imagen tenga una sola componente, como es el caso de las máscaras, el proceso es el mismo, pero con una sola componente y utilizando únicamente un eje que ocupa todo el panel.

Además, en la interfaz de usuario se puede activar el selector de línea, únicamente para los dos modos "*Parade*", para lo cual se ha añadido una condición, que si se cumple sólo se representa los valores de la línea seleccionada.

El resultado se puede observar en la Figura 23.

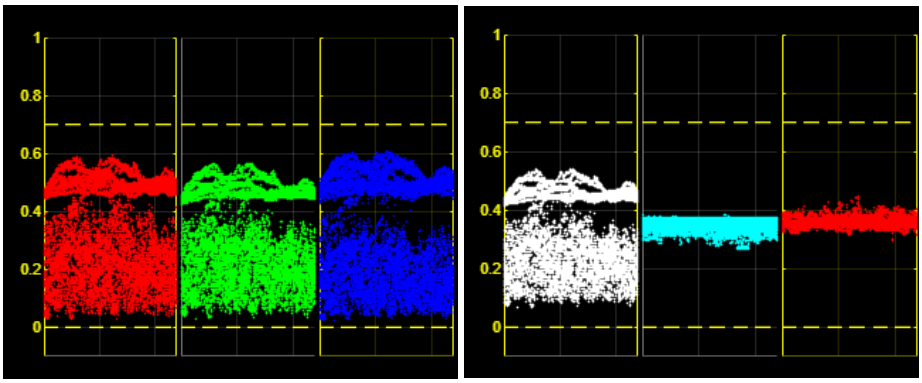

Figura 23. Visualización del display WFM, Parade RGB (izquierda) y Parade YCbCr (derecha).

4.3.2 Función `vector`

La función vector se encarga de hacer una representación de un vectorscopio a partir de la información de color de la imagen.

En primer lugar, se limpia el panel borrando la imagen anterior. A continuación, se convierte la imagen a YCbCr y extrae las componentes de diferencia de color Cb y Cr. Posteriormente, se crea un eje para representar la gráfica y se muestra un diagrama de dispersión de las componentes Cb y Cr mediante la función de Matlab "`scatter`". A continuación, se crea una rejilla con líneas discontinuas mostrando los valores 0 de ambos ejes, y por último se crea una circunferencia con líneas discontinuas para mostrar el valor máximo de la diferencia de color de ambos ejes.

Por último, se calculan los valores de los colores de una barra de test del 75%, se convierten a Y Cb Cr y se dibuja en su posición un cuadrado para mostrar la posición que tendría en la gráfica una señal de test del 75%, y se le añade la etiqueta de cada color.

Además, en la interfaz de usuario existe la posibilidad de elegir mostrar el recorrido que haría una señal analógica de barras de test del 75% en el vectorscopio, para lo que se ha añadido

una condición que, cuando se cumple, dibuja una serie de líneas que unen las posiciones de los colores de las barras de test. El resultado se puede ver en la Figura 24.

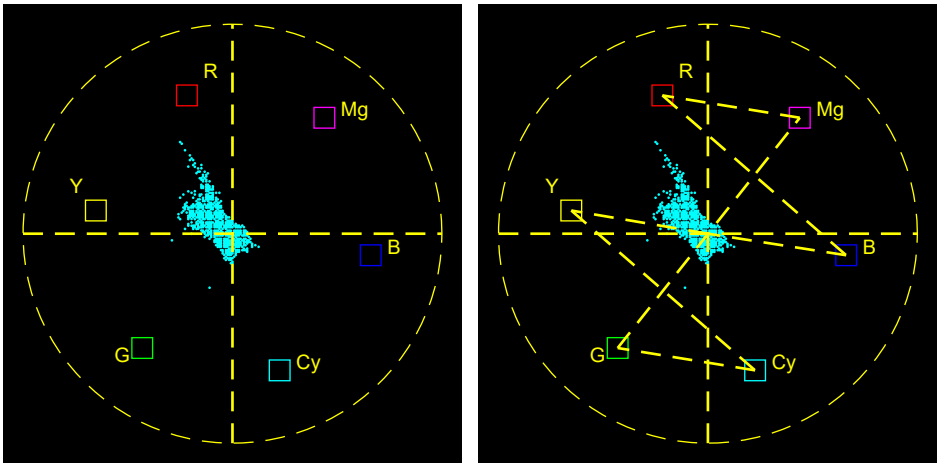

Figura 24. Visualización del display "Vector", sin barras (izquierda) y con barras (derecha).

4.3.3 Función `Gamut`

La función "`gamut`" se encarga de hacer una representación en dos diamantes de las señales G-B (arriba) y G-R (abajo) utilizada para medir el Gamut de las imágenes.

Para ello, una vez borrada la representación anterior, se extraen las componentes R, G y B de la imagen, se crea un nuevo eje y, mediante la función "`plot`", se representa G-B, G+B, G-R y G+R en dicho eje, haciendo uso de la capacidad de Matlab para realizar operaciones con matrices.

Por último, se dibuja la rejilla añadiendo líneas entre los valores máximos y mínimos y el 0, situado en el centro. El resultado se puede ver en la Figura 25.

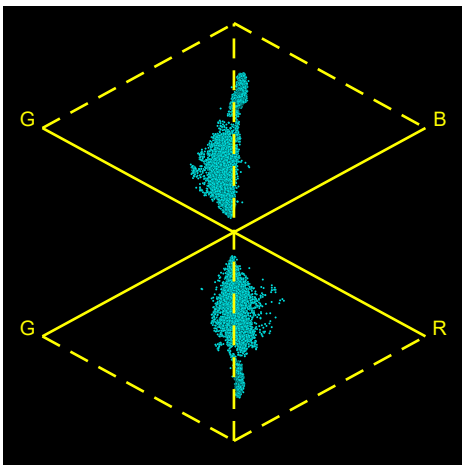

Figura 25. Visualización del display Gamut.

4.4 Segundo doble multiplicador

Para calcular la imagen resultante del segundo doble multiplicador se hace uso de la función "`DobleMultiplicador`", comentada anteriormente, ya que la única diferencia son las variables de entrada y de salida.

En este caso, las entradas del segundo doble multiplicador son, la salida del primer doble multiplicador, por un lado, la imagen nueva con la que se desea componer la imagen de salida, por otro, y una nueva máscara C2, por último, como se puede ver en la Figura 26. La imagen de salida es la que se muestra en el monitor de programa y la que muestra el WFM en caso de estar seleccionada la entrada PGM.

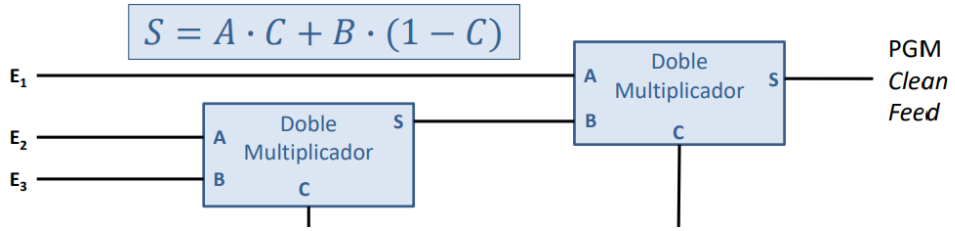

Figura 26. Diagrama de los dos dobles multiplicadores [2].

4.4.1 Generador de bordes

Para la generación de bordes se hace uso del segundo doble multiplicador, utilizando como entrada A una imagen con el color seleccionado en la interfaz de usuario y generada en la función "`ColorBordeButtonPushed`", como entrada B la salida del primer doble multiplicador, y como entrada C una máscara generada por la función "`crearMascaraBorde`", cuyo funcionamiento se detalla a continuación.

Función `CrearMascaraBorde`

La función "`CrearMascaraBorde`" tiene como variables de entrada "p", que es la posición del borde utilizada para crear la máscara C1 ya que es necesario que estén sincronizadas, "`anchoBorde`", que es el ancho del borde y una variable llamada "`opción`" utilizada para definir el tipo de cortinilla a la que se le quiere aplicar el borde. Las cortinillas y los bordes evolucionan en función del valor que el usuario le dé tanto al *fader*, que modifica la posición, como al ancho del borde.

Para crear cada tipo de mascara se implementa un switch a partir de la variable "`opción`", siendo las distintas opciones las siguientes.

Cortinilla horizontal:

A partir de la posición del borde, se crea un rectángulo blanco de alto igual que la imagen y de ancho igual a la variable "`anchoBorde`", que define el ancho del borde, teniendo en cuenta que la mitad del ancho está situado a la izquierda de la posición, y la otra mitad a la derecha,

tal como se indica en (16). Además, se tiene en cuenta si el ancho supera el tamaño de la imagen para truncar el rectángulo.

$$\left[p - \left\lfloor \frac{anchoBorde}{2} \right\rfloor, ancho - p - \left\lceil \frac{anchoBorde}{2} \right\rceil \right] \quad (16)$$

Donde: p: posición del borde

anchoBorde: píxeles que tiene de ancho el borde

ancho: Ancho de la máscara

Como se ha dicho anteriormente, ambas máscaras deben estar sincronizadas, por lo que al cambiar el valor del *fader*, y por lo tanto el valor de la variable "p", ambas máscaras se modifican. En la Figura 27 se puede ver una secuencia de ambas máscaras, siendo la máscara C1 la de arriba y C2 la de abajo.

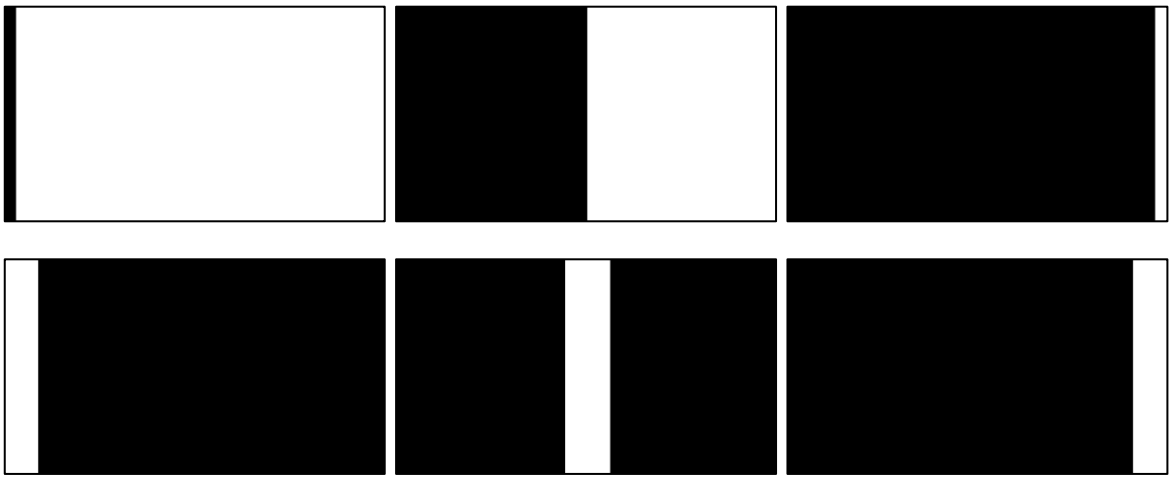

Figura 27. Máscara C1 en diferentes momentos de la transición cortinilla horizontal (arriba) y máscara C2 en los mismos instantes (abajo) en función del valor del fader.

Cortinilla vertical:

Igual que el caso anterior, pero con el ancho igual a la imagen y el alto igual a la variable que define el ancho del borde, según (17). El resultado se puede ver en la Figura 28.

$$\left[p - \left\lfloor \frac{anchoBorde}{2} \right\rfloor, alto - p - \left\lceil \frac{anchoBorde}{2} \right\rceil \right] \quad (17)$$

Donde: p: posición del borde

anchoBorde: píxeles que tiene de ancho el borde

alto: Alto de la máscara

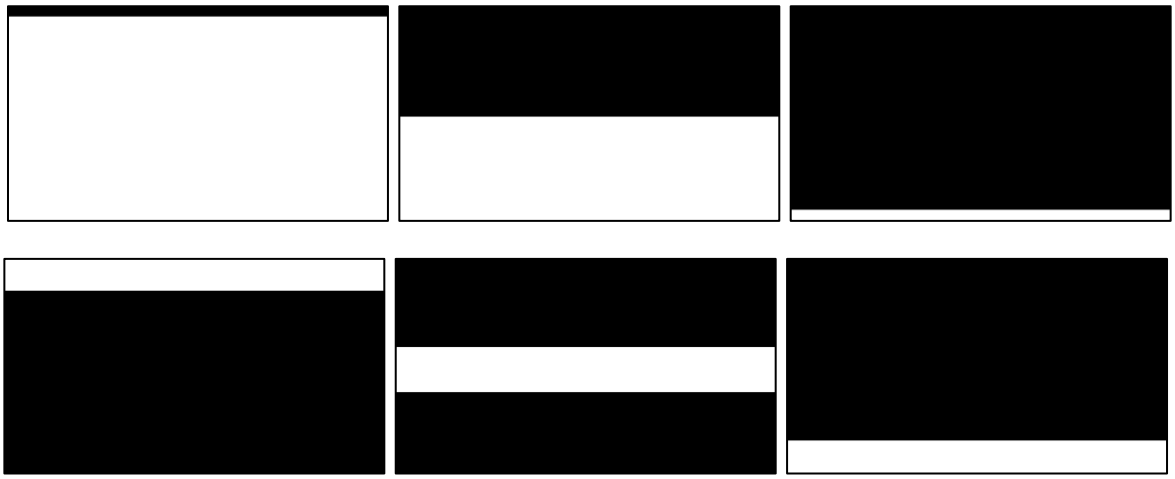

Figura 28. Máscara C1 en diferentes momentos de la transición cortinilla vertical (arriba) y máscara C2 en los mismos instantes (abajo) en función del valor del fader.

Cortinilla diagonal:

La creación de esta máscara se basa en el funcionamiento de la función "`tril`" de Matlab, que, dada una matriz y la posición de la diagonal formada por la hipotenusa, crea un triángulo rectángulo en la parte inferior de la matriz.

La máscara C1 para la cortinilla diagonal se crea realizando la operación `C1 = 1 – trill(ones(alto, ancho), p)`, con lo que se obtiene una máscara como la mostrada en la Figura 29. Como se puede observar, la máscara se compone a partir de un triángulo cuya hipotenusa coincide con la posición del borde diagonal, que forma un ángulo de 45° con la imagen.

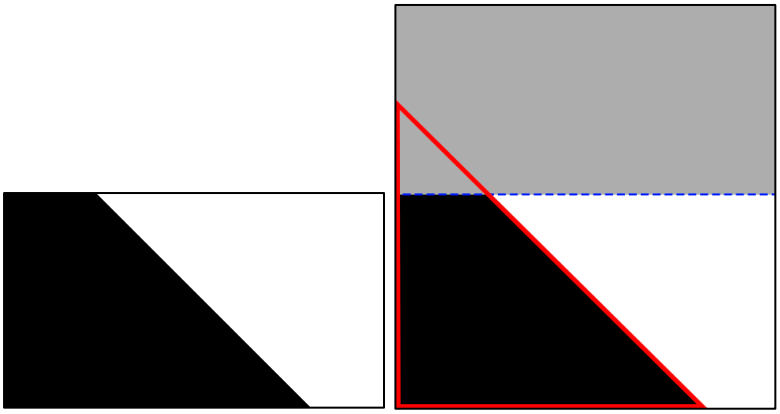

Figura 29. Composición de la máscara mediante la función `trill`.

En base a esto, la máscara C2 que se desea generar se puede descomponer en dos triángulos rectángulos, ya que la posición de las hipotenusas de los triángulos coincide con el borde de la máscara C1 con un desplazamiento, como se muestra en la Figura 30, donde se muestran ambos triángulos y la posición del borde de la máscara C1 en línea discontinua. Por lo tanto, se hace uso de la función "`tril`" para formar dos triángulos sumando y restando este desplazamiento que se calcula como se explica a continuación.

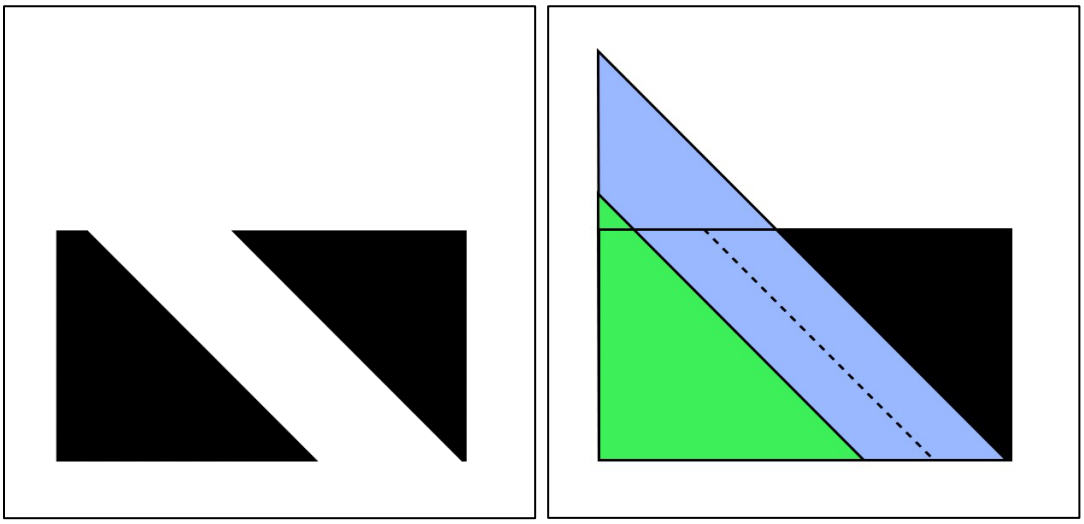

Figura 30. Máscara C2 (izquierda) y composición de la máscara a partir de dos triángulos (verde y azul) y la posición del borde (línea discontinua).

Como los bordes forman un ángulo de 45° con los márgenes de la imagen, se puede trazar un triángulo rectángulo contenido en el borde con el cateto opuesto alineado con uno de los bordes de la máscara C2, cuyo cateto contiguo mide lo mismo que el ancho del borde, como se muestra en la Figura 31.

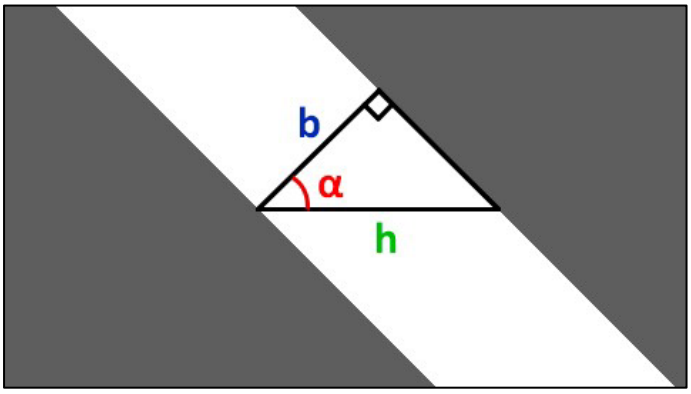

Figura 31. Desplazamiento de los dos triángulos que forman la máscara.

Entonces, a partir de la razón trigonométrica (18)

$$\cos(\alpha) = \frac{b}{h} \tag{18}$$

Siendo b el cateto contiguo, por lo tanto, b = ancho del borde, y h el desplazamiento total se tiene (19)

$$h = \frac{2 \cdot b}{\sqrt{2}} \tag{19}$$

Como el desplazamiento se divide entre los dos triángulos, la cantidad a desplazar cada triangulo viene dada por (3)

$$h' = \frac{h}{2} = \frac{b}{\sqrt{2}} \tag{20}$$

Teniendo todo esto en cuenta, la máscara se compone a partir de un triángulo con la diagonal en la posición p – h' y otro triángulo con la diagonal en la posición p + h', realizando la operación expresada en (21), siendo "p" la variable que define la posición del borde de la máscara C1 como se ha comentado anteriormente.

$$C2 = triangulo_1 + (1 - triangulo_2) \tag{21}$$

Expresado en código, quedaría de la siguiente forma:

```
C2 = tril(ones(alto, ancho), p - fix(anchoBorde/sqrt(2))) + (1 - tril(ones(alto, ancho), p + fix(anchoBorde/sqrt(2))));
```

Se muestra el resultado de ambas máscaras en distintos momentos de la transición en función de la posición del *fader* en la Figura 32.

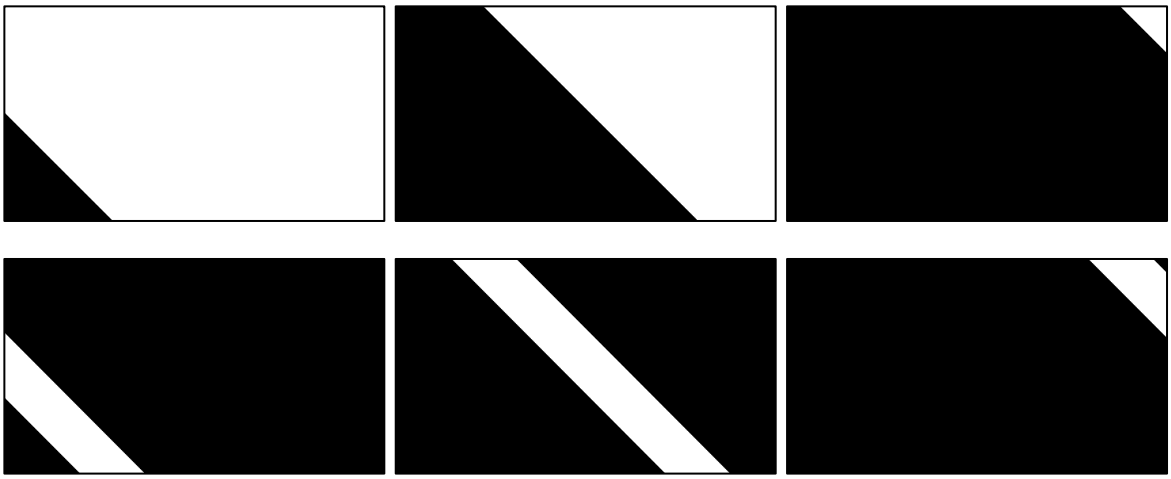

Figura 32. Máscara C1 en diferentes momentos de la transición cortinilla diagonal (arriba) y máscara C2 en los mismos instantes (abajo) en función del valor del fader.

Cortinilla cuadrada:

Se crea una máscara igual a la máscara C1, detallada en el apartado 4.2.2, y se detectan los bordes con la función "`edge`" de Matlab, que es una función que permite detectar bordes en imágenes (cambios abruptos) y devuelve una imagen binaria que contiene el valor 1 en los píxeles donde detecta un borde y 0 en el resto. Después se crea un elemento estructural cuadrado de tamaño igual al ancho del borde necesario para crear el ancho del borde mediante la función "`strel`", que crea un elemento que define el patrón que se emplea en operaciones morfológicas, en este caso cuadrado, especificando `'square'` y "`anchoBorde`" como argumentos de entrada. Por último, se realiza la dilatación del borde utilizando la función "`imdilate`" de Matlab, que a partir de la imagen binaria procedente de la detección de bordes y el elemento estructural cuadrado realiza la dilatación. La máscara resultante se

muestra en la Figura 33, como se puede ver, tanto el cuadrado formado en la máscara C1 como el borde formado en la máscara C2 van cambiando de tamaño en función del valor del *fader*.

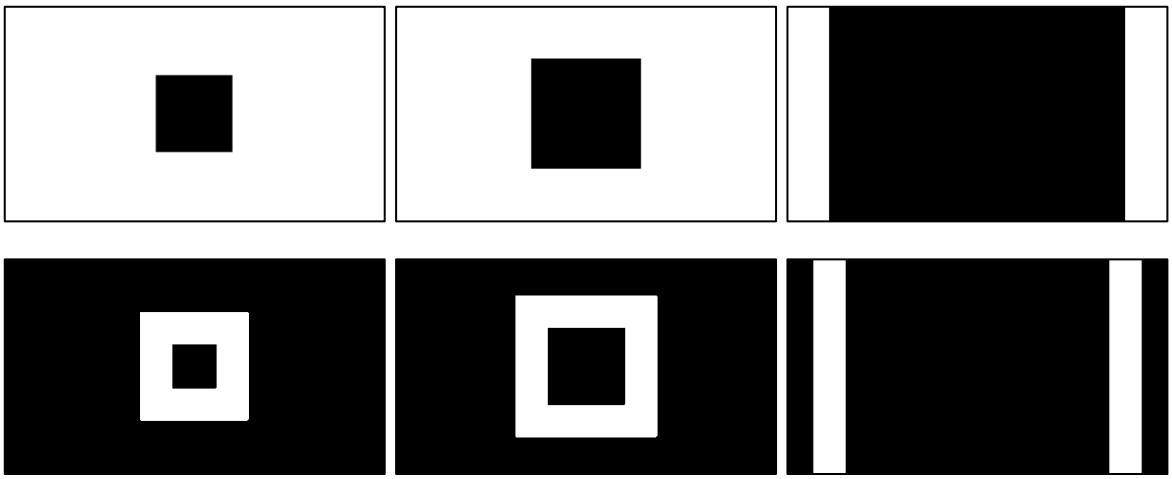

Figura 33. Máscara C1 en diferentes momentos de la transición cortinilla cuadrada (arriba) y máscara C2 en los mismos instantes (abajo) en función del valor del fader.

Cortinilla cruz:

Se utiliza exactamente el mismo procedimiento que en la cortinilla cuadrada, pero a partir de la cortinilla en cruz, explicada en el apartado 4.2.2, el resultado se puede observar en la Figura 34 donde se ve la evolución de ambas máscaras en función del valor del *fader*.

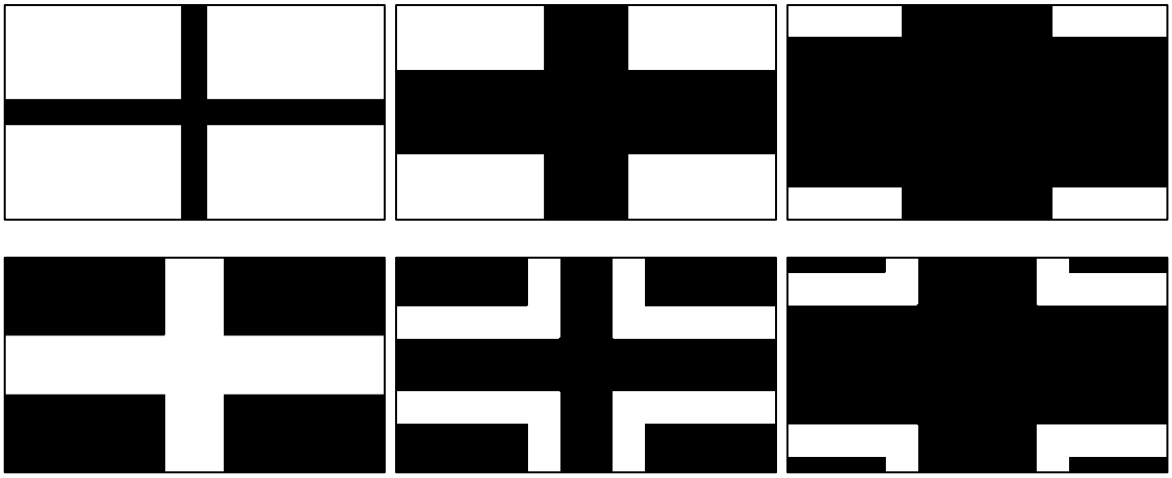

Figura 34. Máscara C1 en diferentes momentos de la transición cortinilla en cruz (arriba) y máscara C2 en los mismos instantes (abajo) en función del valor del fader.

Cortinilla rombo:

A partir de la cortinilla en rombo detallada en el apartado 4.2.2, se utiliza el mismo procedimiento anterior, pero con un elemento estructural con forma de diamante, cuya distancia "r" entre el origen del elemento hasta los puntos del diamante es igual al ancho del borde dividido entre raíz de 2, debido al mismo razonamiento que en la cortinilla diagonal, como se puede ver en la Figura 35.

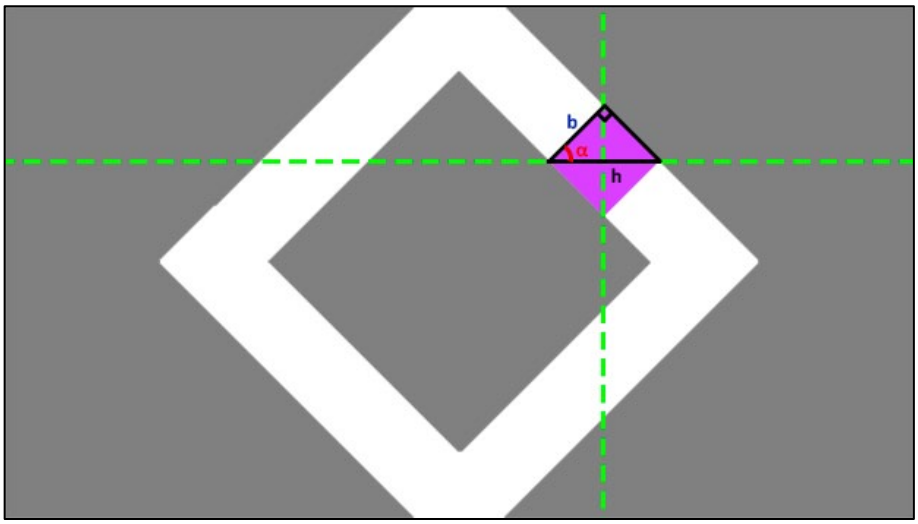

Figura 35. Distancia "r" del elemento estructural en forma de rombo para dilatar el borde y formar la máscara C2.

Por lo tanto, "r" se define mediante (22):

$$r = \frac{h}{2} = \left\lfloor \frac{anchoBorde}{\sqrt{2}} \right\rfloor \tag{22}$$

En la Figura 36 se muestra la máscara C2 obtenida a partir de la máscara C1 y su variación en función del *fader*.

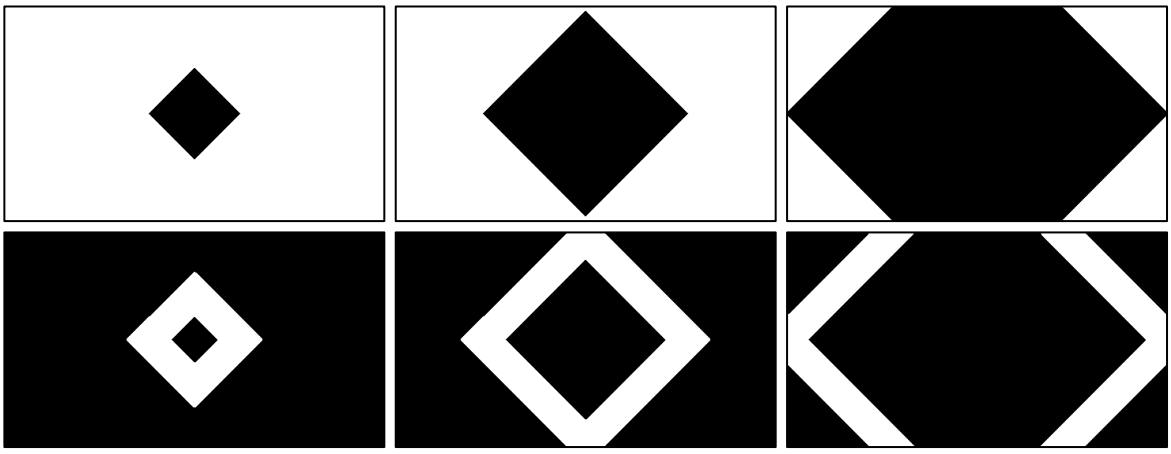

Figura 36. Máscara C1 en diferentes momentos de la transición cortinilla en rombo (arriba) y máscara C2 en los mismos instantes (abajo) en función del valor del fader

Cortinilla circular:

Se crean dos círculos concéntricos con el mismo procedimiento que la máscara C1, explicada en el apartado 4.2.2, con una diferencia de radio igual al ancho del borde, siendo el más pequeño de radio igual a la posición menos la mitad del ancho del borde, y el más grande de radio igual a la posición más la mitad del ancho del borde. En (23) se muestra el cálculo del radio del círculo pequeño y en (24) el cálculo del círculo grande.

$$r_1 = p - \frac{anchoBorde}{2} \tag{23}$$

$$r_2 = p + \frac{anchoBorde}{2} \tag{24}$$

Donde: r1: Radio del círculo pequeño

r2: Radio del círculo grande

Una vez creados los dos círculos, se realiza la operación `C2 = 1 - (C2_2 - C2_1)`, siendo `C2_2` el circulo grande y `C2_1` el circulo pequeño. En la Figura 37 se muestra el resultado.

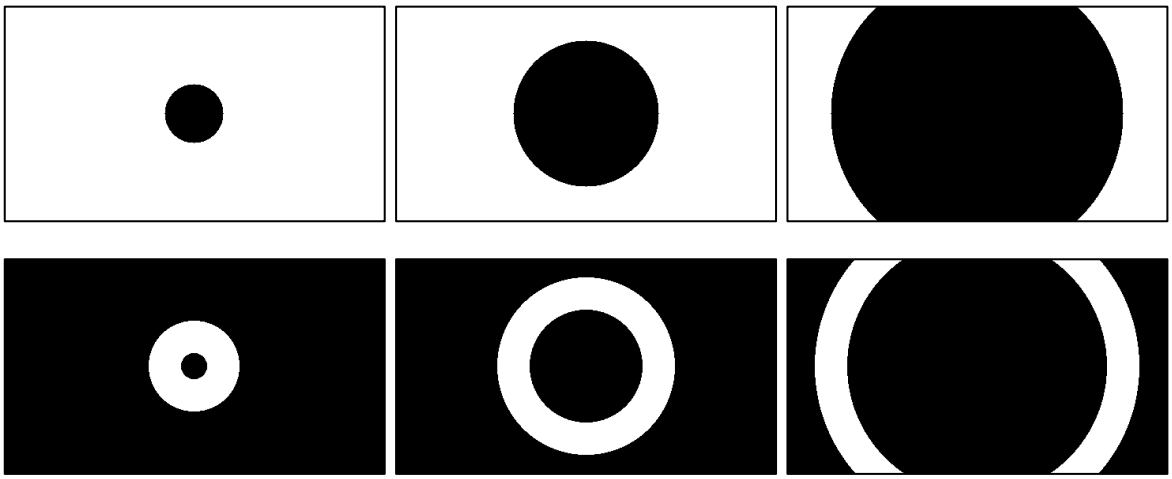

Figura 37. Máscara C1 en diferentes momentos de la transición cortinilla circular (arriba) y máscara C2 en los mismos instantes (abajo) en función del valor del fader

Además, se crea otra función para suavizar el borde, llamada "`SuavizarMascara`"

Suavizar Bordes

Para suavizar los bordes se hace uso de la función "`suavizarMascara`", que consiste en un filtro de media de tamaño igual a la variable de entrada "`ancho`", seleccionado mediante el deslizador "Ancho", que se aplica a la máscara que se recibe como variable de entrada.

En primer lugar, se calcula el filtro a aplicar mediante la función "`fspecial`" de Matlab, que es una función para obtener distintos tipos de filtros de forma sencilla de un tamaño determinado. El filtro obtenido se guarda en la variable "`filtroSuavizado`", especificando como argumentos de entrada `'average'`, para indicar que es un filtro de media, y "`ancho`", que indica el tamaño del filtro. Este filtro se aplica a la máscara mediante la función "`imfilter`" de Matlab, que realiza una convolución de una imagen con un filtro anteriormente calculado de forma eficiente y que permite replicar los bordes de la imagen. Para ello se establecen como argumentos de entrada la variable "`mascara`", el filtro calculado "`filtroSuavizado`" y `'replicate'`, para que se repliquen los píxeles de los bordes de la

máscara. De esta forma se consigue una máscara con un suavizado del tamaño elegido por el usuario.

Para que el resultado sea lo más parecido posible al que se obtiene en el equipo Blackmagic ATEM esta función se aplica a las dos máscaras, la máscara C1, utilizada para la transición de cortinilla, y la máscara C2, utilizada para los bordes. En el caso de la máscara C1 el ancho es el elegido mediante el deslizador "Suavizado" en la interfaz de usuario. Para la máscara C2 el ancho es un porcentaje del ancho del borde, calculado a partir del valor del deslizador "Suavizado" según (25):

$$\% = anchoSuavizado/100 \tag{25}$$

Por lo tanto (26):

$$ancho = \lfloor anchoBorde \cdot anchoSuavizado/100 \rfloor \tag{26}$$

Donde `ancho`: Tamaño del filtro a aplicar

`anchoBorde`: Ancho del borde elegido en el deslizador "Ancho"

`anchoSuavizado`: Ancho del suavizado elegido en el deslizador "Suavizado"

El resultado de suavizar el borde de una cortinilla circular con ancho de borde de 26 píxeles y ancho de suavizado de 55 píxeles, y, por lo tanto, un suavizado del 55% del tamaño del borde para la máscara C2, se muestra en la Figura 38.

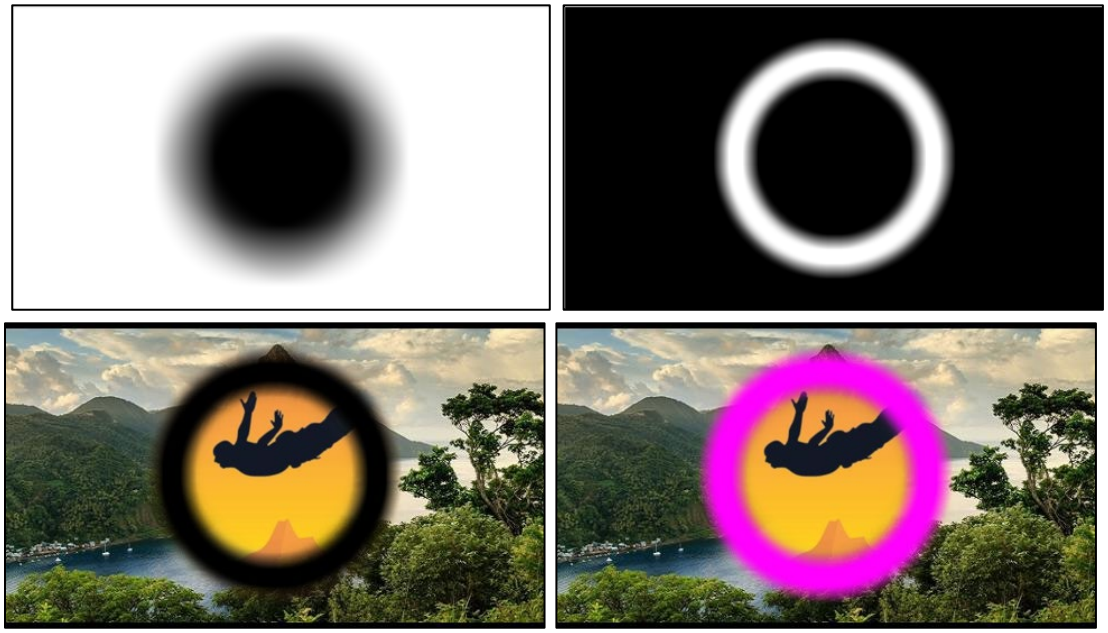

Figura 38. Máscara C1 suavizada con un tamaño del filtro de 55 píxeles (arriba izquierda), Máscara C2 suavizada con un tamaño del filtro de 14 píxeles (arriba derecha), resultado de aplicar ambas máscaras con un color de borde negro (abajo izquierda) y resultado

Con el fin de comprobar el correcto funcionamiento se hace una comparación entre el equipo real y la aplicación, estableciendo los mismos parámetros. En primer lugar, se hacen dos pruebas, en una se elige una cortinilla horizontal con ancho de borde 0 y suavizado 100, y en otra se cambia el suavizado a 50, el resultado se puede ver en la Figura 39.

la Figura 40, la Figura 41 y la Figura 42.

Figura 39. Cortinilla horizontal con ancho de borde 0 y suavizado 100 en equipo real (arriba izquierda) y aplicación (arriba derecha), y ancho de borde 0 y suavizado 50 en equipo real (abajo izquierda) y aplicación (abajo derecha).

A continuación, se repite la misma prueba, pero con el ancho de borde y el suavizado a 100 primero, y a 50 después, en la Figura 40 se muestran las imágenes obtenidas.

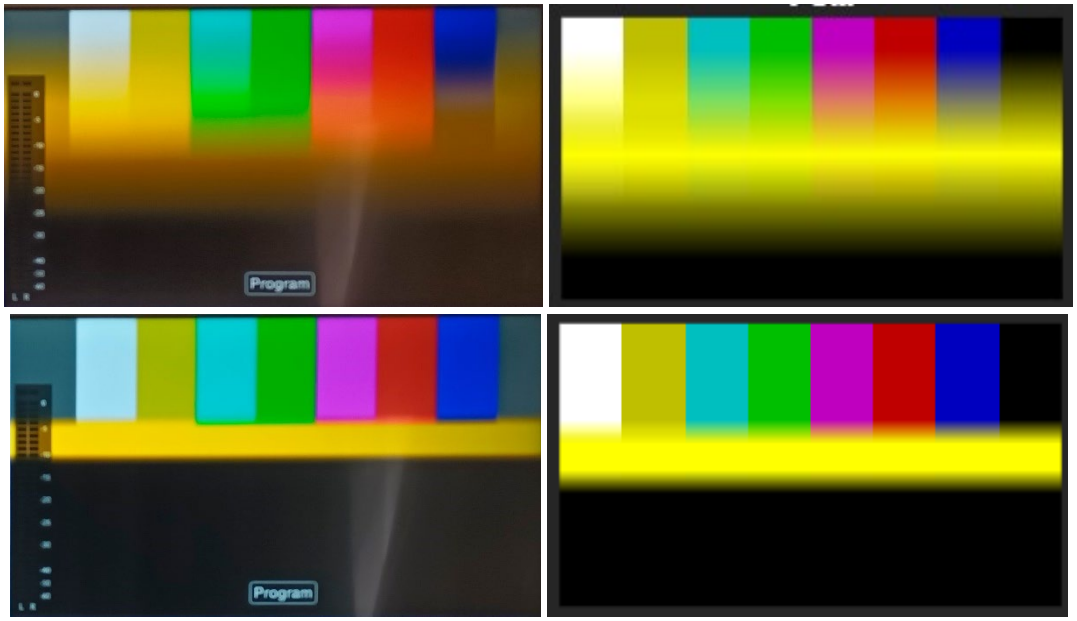

Figura 40. Cortinilla horizontal con ancho de borde 100 y suavizado 100 en equipo real (arriba izquierda) y aplicación (arriba derecha), y ancho 50 y suavizado 50 en equipo real (abajo izquierda) y aplicación (abajo derecha).

Posteriormente, se configura el ancho de borde a 100 y el suavizado a 50, en la Figura 41 se representa el resultado.

Figura 41. Cortinilla horizontal con ancho de borde 100 y suavizado 50 en equipo real (izquierda) y aplicación (derecha).

Por último, se elije una cortinilla circular, en primer lugar, con ancho de borde y suavizado a 0, y después con el suavizado a 50. En la Figura 42 se puede observar el resultado.

Figura 42. Cortinilla circular con ancho de borde 0 y suavizado 0 en equipo real (arriba izquierda) y aplicación (arriba derecha), y ancho de borde 0 y suavizado 50 en equipo real (abajo izquierda) y aplicación (abajo derecha).

Analizando las imágenes obtenidas, se puede concluir que los resultados, sin ser idénticos, son muy parecidos y, sobre todo, se mantiene el mismo concepto.

5. Resultados

A continuación, se muestran algunas pruebas realizadas y sus resultados, con el fin de comprobar el correcto funcionamiento de la aplicación. En primer lugar, se lanza la ejecución de la aplicación y se comprueba que se inicia correctamente, por defecto en modo mezcla como se puede observar en la Figura 43.

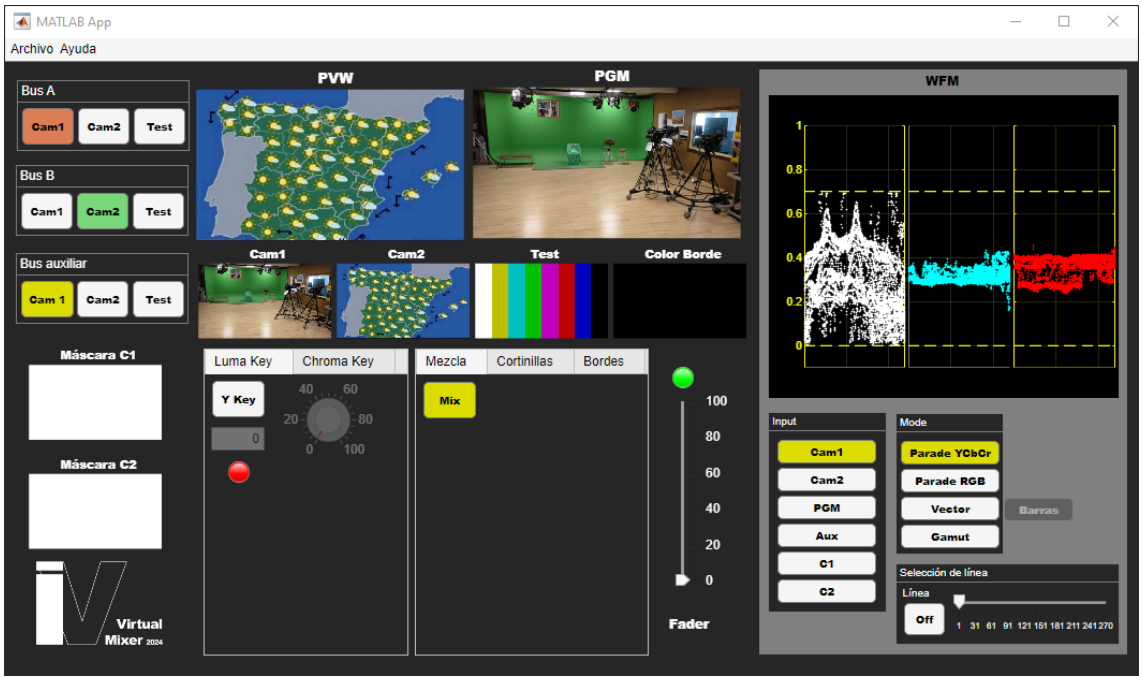

Figura 43. Inicio de la aplicación

Posteriormente, se ajusta el nivel de mezcla al 50% mediante el *Fader*, el resultado se puede ver en la Figura 44, siendo el resultado el esperado.

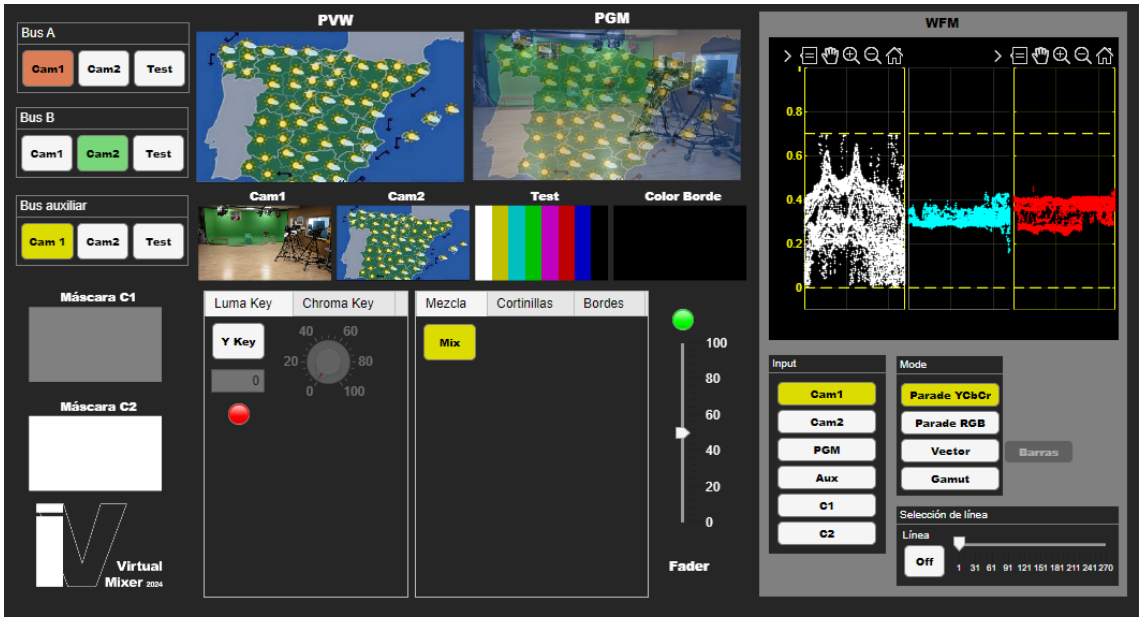

Figura 44. Mezcla al 50%.

A continuación, se ha procedido a cargar la imagen “presentador.jpg” en la cámara 1 y aplicar un *Chroma Key* de color verde, con un valor de matiz de 120°, saturación y brillo del 100%, además, se han ajustado las tolerancias con valor de 10° para el matiz, y 50% para la saturación y el brillo. En el bus A se ha mantenido la cámara 1, en el bus B la cámara 2 y en el bus auxiliar la cámara 1.

También se ha realizado otra prueba cargando la imagen “montañas.png” en la cámara 1, cargando la imagen “mujer.bmp” en la cámara 2, eligiendo la señal de la cámara 1 en el bus auxiliar y seleccionando *Luma Key* con un valor del 80%.

El resultado de ambas pruebas ha sido satisfactorio, como se puede comprobar en la Figura 45.

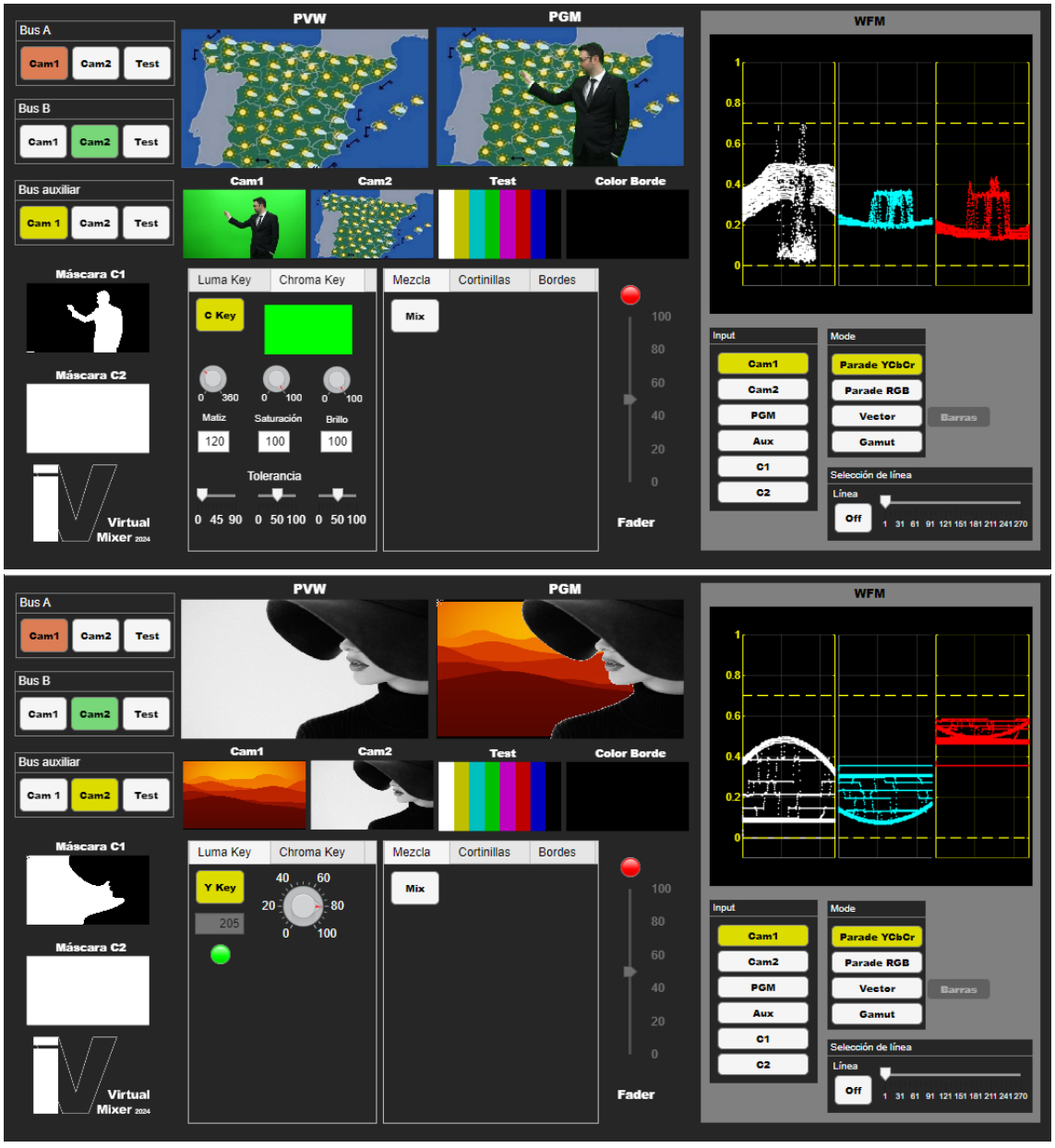

Figura 45. Carga de imagen en cámara 1 y *Chroma Key* verde (arriba) y carga imagen en cámaras 1 y 2 y *Luma Key* al 80% (abajo).

Posteriormente, se ha seleccionado la imagen de test en el Bus A, se ha elegido la transición de cortinilla en modo diagonal con el fader al 50% y se ha seleccionado la entrada de programa en el WFM en modo vector. El resultado es satisfactorio, ya que la transición es la esperada y la visualización en el vectorscopio también es correcta, ya que aparecen puntos en la zona entre el amarillo y el rojo con distinto nivel de saturación, correspondientes a la imagen de la cámara 1 y también aparecen puntos en cada casilla correspondiente a las barras de color de la imagen de test.

Después, se carga la imagen "paisaje.jpg" en la cámara dos, se elige la cortinilla cuadrada con el *fader* al 20% y se selecciona la cámara 2 en el Bus A. El WFM se pone en modo Gamut con la entrada en programa. De nuevo el resultado es el esperado, ya que la transición es correcta y en la visualización Gamut se puede observar puntos correspondientes a la imagen de fondo, que tiene un color equilibrado con cierta predominancia de verde sobre el azul, y puntos correspondientes a la imagen de la cámara 1, que tienen mayor predominancia del rojo sobre el verde. Ambos resultados se muestran en la Figura 46.

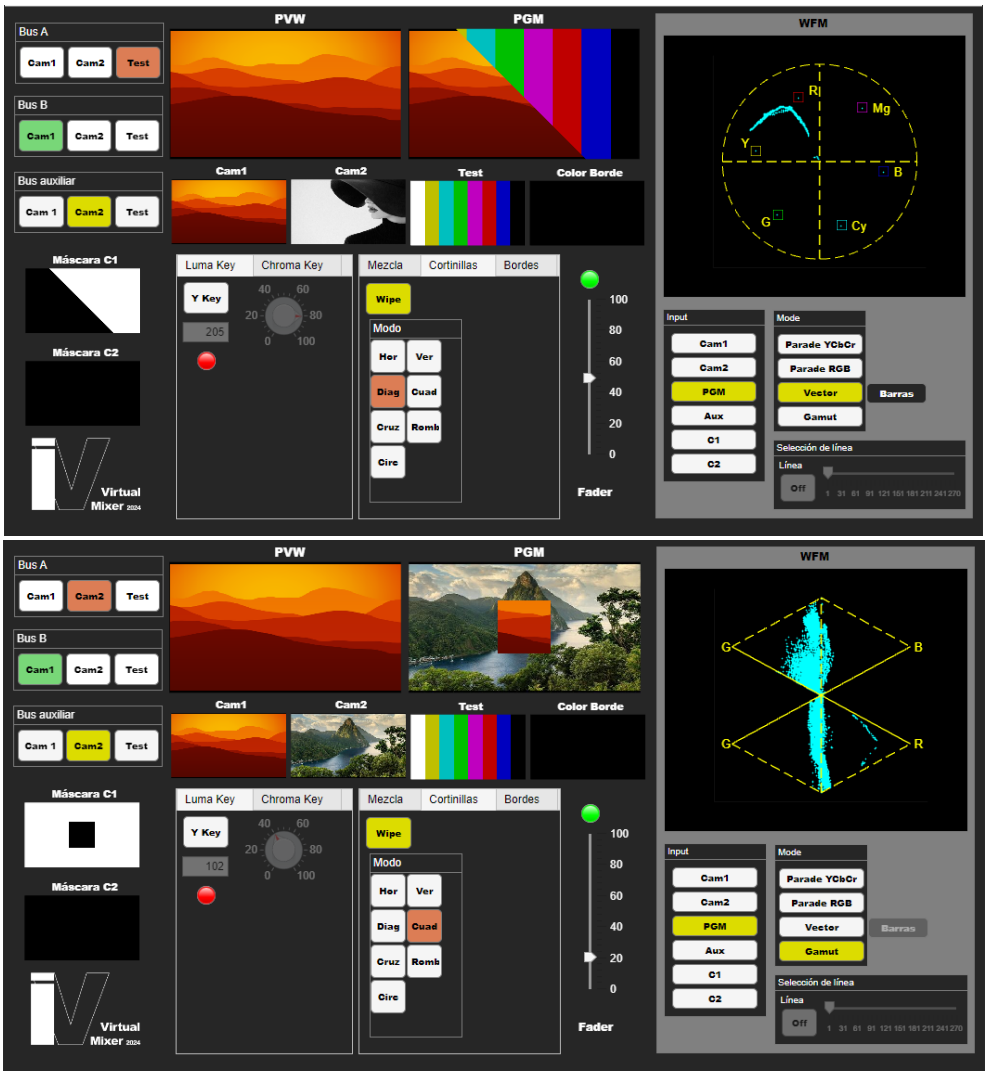

Figura 46. Imagen de test en el bus A, cortinilla diagonal y WFM en modo Vector con entrada PGM (arriba), cortinilla cuadrada al 20 % y WFM en modo Gamut con entrada PGM.

Después, se elige la cortinilla horizontal con el nivel del fader al 50% y se selecciona el modo *Parade YCbCr* en el WFM con la entrada PGM. En la Figura 47 se comprueba que el resultado, tanto de la transición como de la visualización en el WFM, es correcto, ya que en las gráficas correspondientes a las diferencias de color Cb y Cr se aprecia como media imagen no tiene información de color. También se selecciona la cortinilla vertical al 60% y el modo *Parade RGB* en el WFM. De nuevo, en la Figura 47 se observa que el resultado es satisfactorio, ya que los tres colores tienen la misma información en la mayoría de la imagen, con excepción del rojo, que tiene, además, información en los niveles medios-bajos correspondiente a la parte inferior de la imagen de PGM.

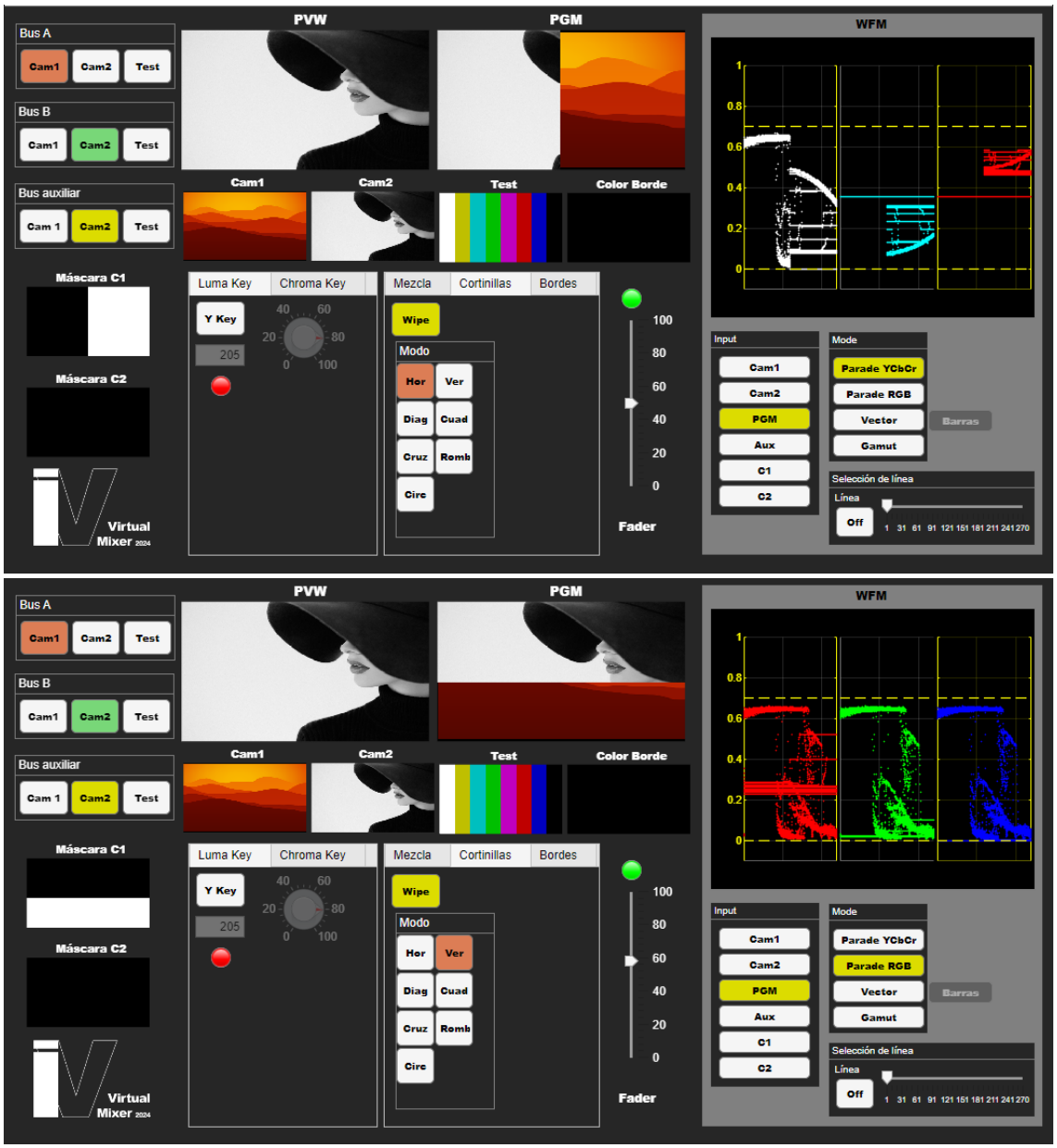

Figura 47. Cortinilla horizontal al 50% y WFM con entrada de programa (arriba), cortinilla vertical al 60% y WFM en modo Parade RGB (abajo)

También se seleccionan las cortinillas cruz, rombo y circular, cuyo resultado se puede ver en la Figura 48.

Figura 48. Cortinilla en cruz al 20% (arriba izquierda), cortinilla en rombo al 20% (arriba derecha), cortinilla circular al 40% (abajo).

Para comprobar el correcto funcionamiento de la generación de bordes se realizan cuatro pruebas con cortinilla horizontal, una con borde negro de 40 píxeles y sin suavizado, con borde negro con suavizado del 50%, con borde negro y suavizado del 100% y sin borde y suavizado del 100% (abajo derecha). El resultado conseguido es correcto, como se observa en la Figura 49.

Figura 49. Cortinilla horizontal con borde negro de 40 píxeles y sin suavizado (arriba izquierda), con borde negro con suavizado del 50% (arriba derecha), con borde negro y suavizado del 100% (abajo izquierda) y sin borde y suavizado del 100% (abajo derecha).

Para comprobar la correcta visualización en el WFM de la máscara C1 se selecciona la transición *Mix*, con el *fader* al 50% aproximadamente. En la Figura 50 se observa que en el WFM en modo *Parade RGB* y con entrada C1 aparece una única gráfica (señal monocromática) con un nivel aproximado de 0.35 en todos los píxeles, por lo tanto, es correcto, ya que es el 50% del nivel máximo de 0.7.

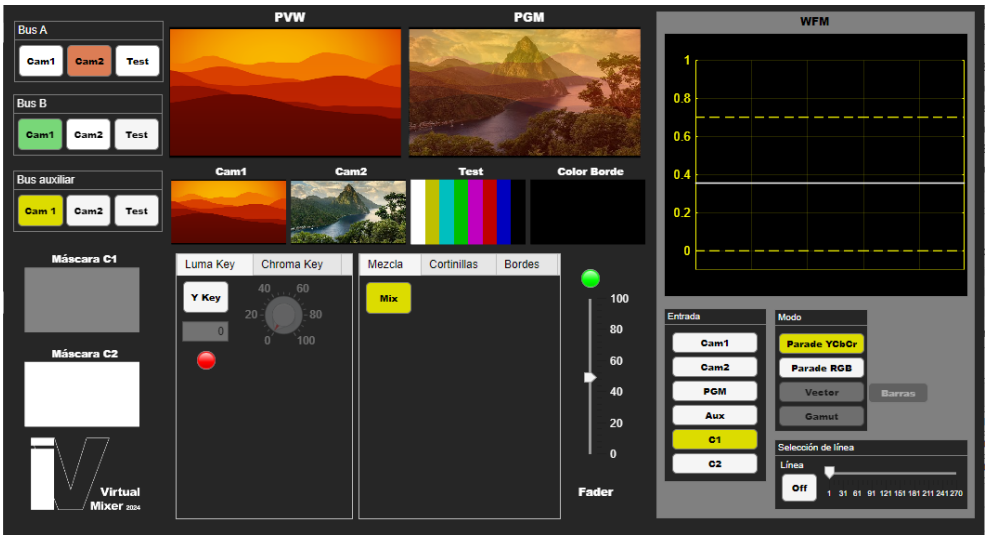

Figura 50. WFM con entrada C1 utilizando la transición Mezcla al 50%

Se realiza otra prueba, seleccionando la entrada C2 del WFM en modo *Parade YCbCr* y se elige una cortinilla horizontal con el ancho del borde de 70 píxeles y suavizado al 100%. En la Figura 51 se observa que el resultado es correcto, ya que los niveles de la máscara en los píxeles correspondientes al borde van de 0 al nivel máximo y vuelven a bajar de forma progresiva.

Figura 51. WFM con máscara C2 a la entrada en modo Parade, con transición cortinilla, borde de 70 píxeles y suavizado al 100%.

Para probar el funcionamiento del selector de línea del WFM, se selecciona la transición *Mix* al 0%, se carga la imagen “Mujer-verde.png” y se elige el modo *Parade RGB* con la entrada de PGM en el WFM, además de activar el selector de línea. En la Figura 52 se pueden observar los resultados de seleccionar las líneas 20 y 150 aproximadamente. Mientras que, en el primer caso, correspondiente a una línea de la parte superior de la imagen, el nivel de los tres colores es uniforme siendo mucho mas alto el de verde, en el segundo caso se observan píxeles con distintos niveles en las tres componentes, ya que se corresponde con una línea de la mitad de la imagen, donde está la mujer, por lo tanto, el resultado es el esperado.

Figura 52. Selección de línea activado, en línea 20 aproximadamente (izquierda) y línea 150 aproximadamente (derecha)

6. Impacto del proyecto

El desarrollo de este proyecto presenta múltiples implicaciones y beneficios en diversas áreas [19]:

Implicaciones sociales y medioambientales

La herramienta Virtual Mixer aportar a los estudiantes de la asignatura de Ingeniería de Vídeo la posibilidad de practicar y familiarizarse con el funcionamiento de un mezclador de vídeo desde cualquier lugar. Por ello, se reduce la necesidad de asistir presencialmente al laboratorio de la asignatura, lo cual contribuye con varios de los Objetivos de Desarrollo Sostenible (ODS) de la Organización de las Naciones Unidas (ONU) [20], como, por ejemplo:

- ODS 4: Educación de calidad. Democratiza el acceso a la educación, permitiendo que estudiantes con limitaciones geográficas o económicas tengan las mismas oportunidades de aprendizaje.
- ODS 11: Ciudades y comunidades sostenibles. Reduce la necesidad de desplazamientos, contribuyendo a la sostenibilidad urbana.
- ODS 12: Producción y consumo responsable. Fomenta un uso más eficiente de los recursos.
- ODS 13: Acción por el clima. Al minimizar los desplazamientos de estudiantes al campus, se reduce la huella de carbono asociada al transporte.

Impacto económico

Además, al ser una herramienta virtual de fácil acceso para todos los alumnos, implica un impacto económico positivo, ya que no tiene gastos de mantenimiento y, sin embargo, si consigue una facilidad en el aprendizaje sin invertir recursos económicos.

Impacto tecnológico

La mejora de la herramienta "Virtual Mixer" incluye la actualización de su capacidad de procesamiento y la incorporación de nuevas funcionalidades, lo que convierte a esta aplicación como una herramienta avanzada para la educación en tecnologías audiovisuales. La exposición de los estudiantes a estas tecnologías fomenta su preparación para el mercado laboral, dotándolos de habilidades prácticas y conocimiento avanzado.

7. Conclusiones

Una vez finalizado el proyecto, se obtienen las siguientes conclusiones

7.1 Conclusiones

Tras la realización de este proyecto se ha obtenido un amplio conocimiento sobre el desarrollo de aplicaciones en App Designer de Matlab, así como de la gestión y procesado de imágenes y vídeos en Matlab.

Por otro lado, se puede concluir que la actualización de la herramienta ha sido satisfactoria, incluyendo la mayoría de los objetivos propuestos y, además, realizando algunas mejoras extra. También se ha facilitado la realización de trabajos futuros al llevar a cabo una reorganización del código y las funciones de la aplicación.

Adicionalmente, han aparecido nuevas ideas para mejorar la aplicación, que se detallan en el apartado 7.1.

7.2 Trabajos futuros

Aunque la actualización de la herramienta ha sido exitosa, algunos de los objetivos detallados en el anteproyecto no se han cumplido. Además, durante el desarrollo de la actualización han surgido ideas nuevas para mejorar la aplicación, habiéndose desarrollado algunas de ellas. Por ello, se propone una serie de tareas para trabajo futuro a desarrollar en otro proyecto de fin de grado, que se detallan a continuación:

7.2.1 Añadir captura de vídeo de webcam

Aunque se ha investigado la posibilidad de añadir captura de vídeo a través de la webcam del equipo en el que se está ejecutando la aplicación, finalmente no se ha implementado debido a su complejidad alta y la necesidad de hacer cambios sustanciales en gran parte del código existente.

7.2.2 Añadir DSK

Esta funcionalidad también era un objetivo inicial del proyecto, no obstante, debido al tiempo extra dedicado a la optimización y mejora de la eficiencia de la aplicación no se ha podido llevar a cabo.

7.2.3 Eliminar variables globales

Se ha intentado la eliminación de variables globales, ya que su uso en Matlab es ineficiente, sin embargo, no se ha conseguido llegar a una solución óptima sin plantear todo el funcionamiento de la aplicación desde cero. Consiste en la integración de otra etapa con otros dos dobles multiplicadores que simulan el efecto DSK (Down Stream Key) para poder poner subtítulos o “moscas” a la salida de video.

7.2.4 Añadir display modo rayo

Este monitor es ampliamente utilizado en ingeniería de vídeo, por lo que ha surgido la idea de implementarlo en el futuro.

7.2.5 Añadir generador de test

Disponer de un generador de señales de test diferentes es una ventaja didáctica muy interesante. Por ejemplo, añadiendo la posibilidad de crear cartas de ajuste con barras de color del 75%, ya implementada, pero también barras de color 100% utilizadas en el entorno de alta definición (HD).

7.2.6 Transición automática

La transición automática es una función de la que disponen todos los mezcladores de vídeo, por lo que es una buena mejora.

7.2.7 Generador señal matte

Desarrollar un generador de señal matte con la posibilidad de incluirla en los buses de entrada. De esta forma se puede utilizar como señal en las mezclas y se puede visualizar en el WFM. Este generador debe tener un menú dedicado especialmente para su configuración.

7.2.8 Añadir bordes a cualquier máscara

Implementar la posibilidad de añadir bordes a cualquier máscara, no solamente a las cortinillas, mediante un detector de bordes genérico.

A.1.1. Añadir posibilidad de cargar vídeos

Terminar de implementar la carga, reproducción y procesado de vídeos, ya que como se ha explicado en el apartado 4.2.4, aunque se ha avanzado mucho, no ha sido posible finalizar completamente esta funcionalidad.

8. Referencias

[1] J. Chen, “Implementación de herramientas virtuales de aprendizaje para control de cámara y mezcla/efectos de vídeo”, PFG, Dpto. ingeniería Audiovisual y Comunicaciones, Universidad Politécnica de Madrid, Madrid, España, 2021.

[2] M. Eckert, *Mezcladores Y Dve*, Dpto. ingeniería Audiovisual y Comunicaciones, Universidad Politécnica de Madrid, ETSIST, 2021.

[3] M. Eckert, *Mezcladores de Vídeo y DVE (Digital Video Effects)*, Dpto. ingeniería Audiovisual y Comunicaciones, Universidad Politécnica de Madrid, ETSIST, 2020.

[4] “ATEM Television Studio – Features | Blackmagic Design”, 2024, https://www.blackmagicdesign.com/products/atemtelevisionstudio/features, [Consultado: 2 de julio de 2024].

[5] M. Eckert, *Medidas de contenido visual – parte A*, Dpto. ingeniería Audiovisual y Comunicaciones, Universidad Politécnica de Madrid, ETSIST, enero 2020.

[6] M. Eckert, *Tema 2 – Medidas y Ajustes*, Dpto. ingeniería Audiovisual y Comunicaciones, Universidad Politécnica de Madrid, ETSIST, 2022.

[7] Telestream® “Telestream Announces Latest Release of PRISM Waveform Monitor Software”, julio 2021, https://www.telestream.net/company/press/2022-07-21-PRISM-release.htm, [Consultado: 2 de julio de 2024].

[8] “WFM601 – TekWiki”, 17 de octubre de 2022, https://w140.com/tekwiki/wiki/WFM601, [Consultado: 2 de julio de 2024].

[9] Tilano, “Tilano TV - ¿Necesitas saber cómo se mide con un vectorscopio?”, 26 de noviembre de 2014, https://tilanotv.es/necesitas-saber-como-se-mide-con-un-vectorscopio/, [Consultado: 2 de julio de 2024].

[10] M. Eckert, *Medidas de contenido visual – parte B*, Dpto. ingeniería Audiovisual y Comunicaciones, Universidad Politécnica de Madrid, ETSIST, enero 2020.

[11] UIT, “Rec. UIT-R BT.471-1. Nomenclatura y descripción de las señales de barra de color.” 1 de julio de 1986.

[12] “Wikipedia, la enciclopedia libre. Barras de color” 20 de noviembre de 2022, https://es.wikipedia.org/wiki/Barras_de_color, [Consultado: 2 de julio de 2024].

[13] “MATLAB App Designer”, https://es.mathworks.com/products/matlab/app-designer.html, [Consultado: 2 de julio de 2024].

[14] UIT, “Rec. UIT-R BT.2087. Conversión de color de la Recomendación UIT-R BT.709 a la Recomendación UIT-R BT.2020”, 14 de octubre de 2015.

[15] UIT, “Rec. UIT-R BT.709-6. Valores de los parámetros de la norma de TVAD para la producción y el intercambio internacional de programas”, 17 de junio de 2015.

[16] UIT, “Rec. UIT-R BT.2020-2. Valores de los parámetros de los sistemas de TVUAD para la producción y el intercambio internacional de programas”, 14 de octubre de 2015.

[17] UIT, “Recomendación UIT-R BT.601-7. Parámetros de codificación de televisión digital para estudios con formatos de imagen normal 4:3 y de pantalla ancha 16:9”, 8 de marzo de 2011.

[18] C. Almonacid Ramiro “UAM. Descripción del modelo de color HSL (Hue, Saturation, Ligthness)”, diciembre 2012, http://guiadigital.uam.es/SCUAM/documentacion/color.php, [Consultado: 2 de julio de 2024].

[19] Fernández Aller, Celia; Miñano, Rafael. "Guía para trabajar la Responsabilidad Social y Ambiental (GRSA)", ETSI Sistemas Informáticos, UPM, (Versión 2.0 mayo 2015), https://oa.upm.es/35542/1/Guia_Responsabilidad_Social_y_Ambiental-V2-1.pdf, [Consultado: 2 de julio de 2024].

[20] ONU, “Objetivos y metas de desarrollo sostenible - Desarrollo Sostenible”, https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/, [Consultado: 2 de julio de 2024].

[21] “Indeed. Sueldo de Ingeniero/a en telecomunicaciones en Comunidad de Madrid”, 16 de abril de 2024, https://es.indeed.com/career/ingeniero-en-telecomunicaciones/salaries/Comunidad-de-Madrid, [Consultado: 2 de julio de 2024].

Anexo I: Presupuesto

El presupuesto para la realización de esta aplicación se detalla a continuación.

Para su cálculo se han tenido en cuenta las siguientes cuestiones:

Herramientas utilizadas:

- Ordenador portátil personal
- Matlab 2023b
- Toolbox necesarios:
    - Image Processing Toolbox
    - Image Acquisition Toolbox
    - Signal Processing Toolbox
    - Instrument Control Toolbox

En la Tabla 1 se hace un desglose de los gastos materiales

Tabla 2. Gastos en recursos materiales.

Producto	Valor (€)	Vida útil (años)	Años de uso	Coste (€)
Ordenador portátil HP	649,00	5	1	129,80
MATLAB and Simulink Student Suite	69,00	1	1	69,00
Image Processing Toolbox	0,00	1	1	0,00
Signal Processing Toolbox	0,00	1	1	0,00
Instrument Control Toolbox	0,00	1	1	0,00
Image Acquisition Toolbox	7,00	1	1	7,00
TOTAL				205,80

A continuación, se hace el cálculo del tiempo de mano de obra, para ello se detallan las horas empleadas en la Tabla 3.

Tabla 3. Planificación de las tareas previstas.

Fase	ID	Nombre de la tarea	Inicio	Fin	Duración (h)
Fase 1	1	Probar la aplicación existente	11/09/2023	13/09/2023	6
	2	Definir las mejoras a realizar	14/09/2023	16/09/2023	6
	3	Análisis y estudio de Matlab AppDesigner	17/09/2023	20/09/2023	9
	4	Comprensión y depuración del código existente	21/09/2023	26/09/2023	15
Fase 2	5	Añadir cortinillas	27/09/2023	01/10/2023	12
	6	Añadir RGB al WFM	02/10/2023	12/10/2023	30
Fase 3	7	Añadir carga de vídeos	13/10/2023	22/10/2023	27
	8	Añadir captura de imagen de la webcam	23/10/2023	03/11/2023	33
	9	Añadir segundo doble multiplicador	04/11/2023	15/11/2023	33
	10	Añadir DSK	16/11/2023	28/11/2023	36
	11	Actualizar la interfaz y el diseño del mezclador	29/11/2023	03/12/2023	12
Fase 4	12	Pruebas, depuración y revisión	04/12/2023	12/12/2023	24
	13	Redacción de la memoria	13/12/2023	12/01/2024	90
	14	Preparación de la defensa	13/01/2024	22/01/2024	27
				Total horas	**360**

Para estimar el coste de la mano de obra, teniendo en cuenta que el salario medio de un ingeniero de telecomunicaciones en la Comunidad de Madrid es de 20,43 €/hora [21], se tiene (27)

$$360\ h \cdot 20{,}43 \frac{€}{h} = 7354{,}80\ € \tag{27}$$

Con estos datos, ya se puede calcular el presupuesto final, representado en la Tabla 4:

Tabla 4. Presupuesto final.

Concepto	Importe (€)
Recursos materiales	205,80
Mano de obra	7354,80
TOTAL	7560,6

Anexo II: Manual de usuario

A.2 Interfaz de usuario

La interfaz gráfica de usuario de la aplicación se divide en varios apartados.

En primer lugar, está el menú superior, donde se encuentra el menú Archivo y el menú Ayuda. A continuación, se encuentra la sección de buses. Debajo de los buses se encuentran dos monitores para mostrar las máscaras C1 y C2.

En la parte central, arriba, se encuentran los monitores de previo (PVW), programa (PGM), cámara 1 (Cam1), cámara 2 (Cam2), carta de ajuste (Test) y color de borde. En la parte inferior se encuentran las pestañas *Luma Key* y *Chroma Key* y Mezcla, Cortinillas y Bordes, que sirven para elegir el tipo de transición. A la derecha de estas pestañas está el *Fader*.

Por último, a la derecha de la interfaz está el monitor WFM.

A.3 Uso básico

A.3.1. Menú archivo

Permite elegir un archivo de imagen para cada una de las cámaras, en los formatos .png, .jpg o .bmp en cualquier resolución. La aplicación adaptará la resolución y la relación de aspecto automáticamente. Este menú se muestra en la Figura 53.

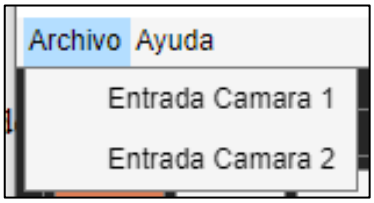

Figura 53. Menú archivo.

A.3.2. Buses

En los buses A y B se puede elegir qué señal se utiliza en las entradas A y B del doble multiplicador, pudiendo elegir entre cámara 1 (Cam1), cámara 2 (Cam2) o señal de carta de ajuste (Test). En el bus auxiliar se elige la señal que se utiliza como key para las opciones de *Luma Key* y *Chroma Key*. La máscara creada se muestra en el *display* Máscara C1. En la Figura 54 se muestra la sección de buses.

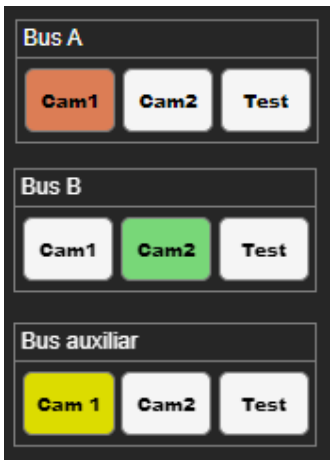

Figura 54. Sección de buses.

A.3.3. Pestaña *Luma Key* y *Chroma Key*

Luma Key

En la pestaña *Luma Key* se puede crear una máscara a partir del valor de luminancia de la señal elegida en el bus auxiliar. Mediante el *knob* se elige qué porcentaje de luminancia se desea incluir en la máscara. También se puede introducir el valor numérico en el cuadro de texto. La bombilla indica cuando está habilitado el *Luma key*.

Chroma Key

En la pestaña *Chroma Key* se puede crear una máscara a partir del color deseado. Para elegir el color se encuentran tres *knobs*, uno para el matiz (*hue*), otro para la saturación (*saturation*) y otro para el brillo (*value*). Debajo de cada *knob* también hay un cuadro de texto donde se puede introducir un valor numérico. El valor del matiz está establecido en grados, entre 0 y 360. Los valores de saturación y brillo están en porcentaje, entre 0 y 100. Por último, hay tres deslizadores de tolerancia, uno para cada valor: Matiz, saturación y brillo. Estas pestañas se pueden observar en la Figura 55.

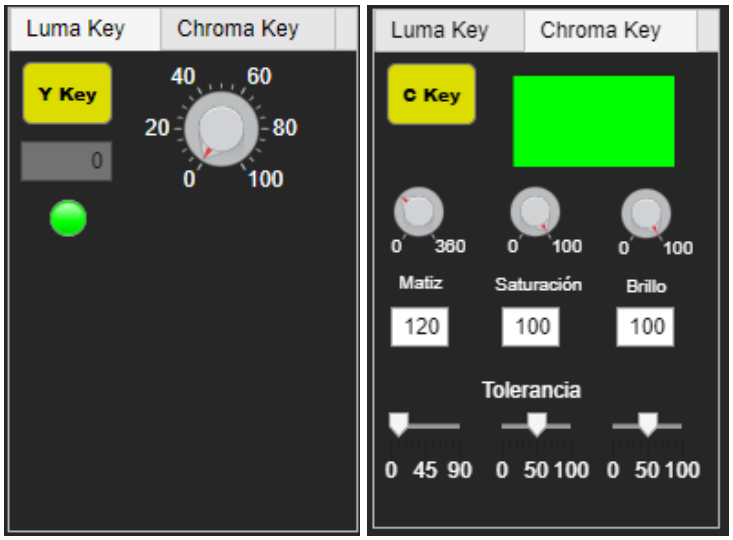

Figura 55. Pestañas *Luma* y *Chroma Key*.

Ambas máscaras se visualizan en el *display* Máscara C1.

A.3.4. Pestaña Mezcla, cortinillas y bordes

Mezcla

En esta pestaña se selecciona el modo mezcla, que consiste en una transición progresiva de fundido entre una imagen y otra. Dicha transición se controla mediante el *Fader*, que indica el porcentaje de cada imagen que se muestra en el monitor de programa (PGM).

Cortinillas

En esta pestaña se elige el modo cortinilla (*Wipe*), donde se pueden seleccionar los modos de cortinillas disponibles: Horizontal, vertical, diagonal, cuadrada, cruz, rombo y circular.

Bordes

En la pestaña borde se puede elegir el ancho del borde (sólo disponible para la transición de cortinilla), el suavizado del borde y el color del borde. La máscara necesaria para la creación de bordes se muestra en el monitor Máscara C2.

En la Figura 56 se representan estas pestañas.

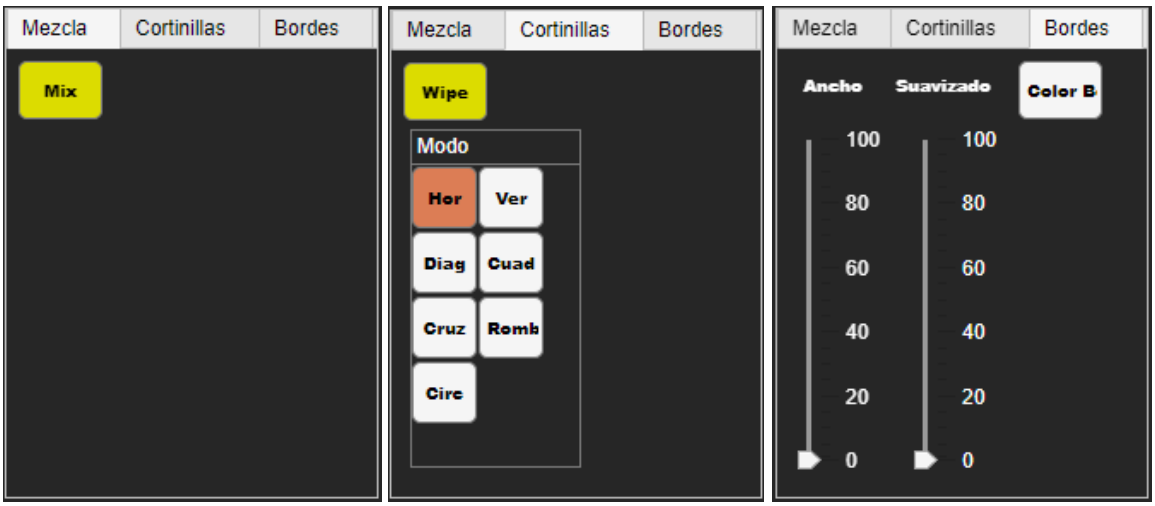

Figura 56. Pestañas mezcla, cortinillas y bordes.

A.4 WFM

A la derecha de la interfaz de usuario se encuentra el monitor de forma de onda (WFM), que se encarga de representar distintas propiedades de la señal elegida.

A.4.1. Entrada

En el menú Entrada se puede seleccionar la señal que se quiere analizar, pudiendo elegir entre cámara 1 (Cam1), cámara 2 (Cam2), programa (PGM), auxiliar (Aux), máscara 1 (C1) o máscara 2 (C2).

A.4.2. Modo

En este menú se elige qué modo se quiere visualizar en el monitor.

- **Parade YCbCr**: Consiste en tres formas de onda en fila, una para cada componente, luminancia (Y), diferencia de azul (Cb) y diferencia de rojo (Cr).
- **Parade RGB**: Igual que la anterior, pero mostrando las componentes primarias rojo (R), verde (G) y azul (B).
- **Vector**: Consiste en un vectorscopio, donde se puede visualizar la información de color de la señal, donde en el eje horizontal se tiene la componente Cb y en el eje vertical la componente Cr. Además, la distancia al centro es la saturación. El monitor dispone de seis casillas, que se corresponden con la posición en el vectorscopio de los colores presentes en las barras de la carta de ajuste del 75%. También se cuenta con una opción para mostrar el recorrido que realiza a través del vectorscopio la señal en una imagen de barras de color, activando el botón "Barras".
- **Gamut**: Es una representación en diamante de la información de color. En el diamante superior se muestra G – B, mientras que en el diamante inferior se muestra G – R.

A.4.3. Selección de línea

Para los modos *Parade* se puede activar la selección de línea, en cuyo caso sólo se representa la información de la línea seleccionada en el deslizador.

En la Figura 57 se representan las opciones del WFM.

Figura 57. Opciones del WFM.