PROYECTO FIN DE GRADO

TÍTULO: Actualización y mejora de la herramienta virtual “VirtualCCU” de simulación de una unidad de control de cámara.

AUTOR/A: Juan Sempere Martínez-Conde

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/A: Oscar Ortiz Ortiz

TUTOR/A: Martina Eckert

SECRETARIO/A: Jose Luis Rodríguez Vázquez

Fecha de lectura: 18 Julio 2024

Calificación:

El Secretario/La Secretaria,

Resumen

Este proyecto de fin de grado se basa en la actualización de una aplicación desarrollada en la plataforma GUIDE y posteriormente migrada al entorno de interfaz de usuario App Designer de Matlab. La aplicación realiza una simulación de una unidad de control de una cámara JVC de TV estándar y tiene como fin resolver ejercicios preparativos de las prácticas de laboratorio de ingeniería de Video.

La unidad de control de cámara utilizada en dicho laboratorio fue renovada en el curso 2022-23 por la nueva unidad de control de cámara BlackMagic Design ATEM dejando dicha aplicación obsoleta. A lo largo de este trabajo se va a explicar tanto las actualizaciones realizadas en la aplicación como las nuevas funcionalidades que se han tenido que añadir para solventar la mencionada obsolescencia.

Abstract

This final year project is based on the update of an application developed on the GUIDE platform and later migrated to Matlab's App Designer user interface environment. The application simulates a control unit of a standard TV JVC camera and aims to prepare exercises for video engineering laboratory practices. The camera control unit used in this laboratory was replaced in the 2022-23 academic year by the new BlackMagic Design ATEM camera control unit, rendering the application obsolete. Throughout this work, both the updates made to the application and the new functionalities added to address the mentioned obsolescence will be explained.

Lista de acrónimos

WFM *Wave Form Monitor* – monitor forma de onda

RGB *Red Green Blue* – rojo verde azul

CCU *Camera Control Unit* – unidad de control de cámara

Índice de figuras

Contenido

1. Introducción

La importancia de la *Camera Control Unit* (CCU) en un plató es fundamental para garantizar la calidad y la coherencia visual de la producción audiovisual. Muestra de ello es que, por ejemplo, es el equipo usado para controlar y ajustar parámetros clave de la cámara desde la sala de control, como el balance de blancos, la exposición, la ganancia y la saturación de color.

El control de estos parámetros de calidad mencionados previamente se hace mediante la monitorización de las componentes RGB o YCbCr en un monitor de forma de onda (WFM – *Wave Form Monitor*). El WFM es un dispositivo de medición que ofrece distintos modos de visualización: modo Parade, donde se representa gráficamente la intensidad luminosa de las señales de video en función del tiempo; modo *Vector,* donde se analiza la colorimetría de la señal para poder detectar posibles fallos de tono y saturación*;* y el modo Gamut en forma de diamante, donde se puede comprobar rápidamente la legalidad de los valores de tensión de las señales*.* Todo esto lo convierte en una herramienta esencial para monitorear y ajustar la calidad de las imágenes en una producción.

Por lo tanto, viendo la importancia de la CCU, es imprescindible que cualquier alumno de la especialidad de Imagen y Sonido practique con una de ellas al cursar el laboratorio de la asignatura Ingeniería de Video, del sexto semestre. Esta asignatura dispone de un estudio de televisión como laboratorio, equipado con lo que sería necesario para hacer una pequeña producción (varias cámaras de video profesionales, un plató, una zona de control con dos puestos de trabajo equipados con CCU, mezclador de video, monitores de forma de onda, grabadoras, generador de señales etc.). Pero los dos puestos de práctica son escasos para el tiempo asignado a los ECTS de la asignatura por lo que una herramienta virtual permitiría un valor añadido al aprendizaje para los alumnos. Para aliviar esta situación se decidió utilizar la herramienta creada por el profesor Luis Ortiz en 2015, llamada VirtualCCU 1.0 que simulaba el entorno y las funcionalidades de una Camera Control Unit de una cámara JVC de TV estándar (programada en Matlab GUIDE) y migrarla a Matlab AppDesigner, con el fin de crear una aplicación de Matlab que pudiera ser usada por los alumnos desde casa y pudieran practicar con ella durante todo el curso sin necesidad de ir al laboratorio. Esta tarea fue llevada a cabo por la alumna Jiajie Chen en su proyecto de fin de grado en el curso 20-21 [1], la cual además añadió alguna funcionalidad más a la versión anterior (por ejemplo, la ampliación de formatos de imagen admitidos por el programa).

Una vez obtenida dicha aplicación de MatLab AppDesigner, los alumnos disponían de una herramienta virtual con todos los ajustes que encontrarían en la herramienta del laboratorio y que podría ser utilizada para probar todos los aspectos vistos en la teoría. En definitiva, tendrían una parte del equipamiento del laboratorio en su ordenador listo para ser utilizado en cualquier momento. Durante el curso 22-23 se compraron e instalaron las nuevas BlackMagic Design ATEM en el estudio. Este nuevo ATEM es particular ya que se trata de una herramienta que realiza la función de mezclador pero que contiene tambien la función de CCU. Con este cambio, la aplicación VirtualCCU 1.0 quedó obsoleta. El objetivo de este proyecto es, por tanto, simular el entorno que los alumnos tendrán con esta nueva herramienta del laboratorio y ampliar las funcionalidades que se tenía en la última herramienta virtual VirtualCCU 1.0.

2. Objetivos

Para cumplir el proyecto, se definen distintos objetivos que se enuncian a continuación:

- Optimización del código y resolución de posibles problemas.
- Actualización y adaptación de las funcionalidades existentes (balance de blancos, filtro de temperatura de color, ganancia etc.)
- Desarrollo de las nuevas funcionalidades que aporta la CCU de BlackMagic (balance de negro, botón giratorio de ganancia común para el balance de blancos, pedestal etc)
- Incorporación de nuevas funcionalidades a la aplicación ajenas a la nueva CCU (posibilidad de tratar videos además de imágenes, posibilidad de conectar la webcam)
- Actualización de la interfaz de la aplicación para que simule el nuevo entorno que se encontrarán los alumnos en el laboratorio.
- Ampliación las funcionalidades del WFM incorporado en la herramienta.
- Conseguir que el procesado sea fluido, aplicando una programación eficiente.

Para cumplir dichos objetivos, se usa el programa Matlab AppDesigner con el código creado por Luis Ortiz y Jiajie Chen. Al cumplir los objetivos, se obtiene una aplicación ejecutable en Matlab con una gran parte de las funcionalidades de la nueva CCU BlackMagic.

3. Marco tecnológico

Para el desarrollo de este proyecto se utiliza la versión ya existente de herramienta virtual VirtualCCU 1.0 basada en la antigua CCU del laboratorio y la herramienta de desarrollo de aplicaciones de Matlab AppDesigner.

3.1 VirtualCCU1.0

El nuevo mezclador incorpora funciones de control de cámara, por lo que ya no es necesario un equipo CCU. El nuevo mezclador incorpora funciones de control de cámara CCU [2], por lo que ya no es necesario un equipo CCU. La interfaz de dicha CCU se puede ver en la figura 1 y sus funcionalidades se explican en la siguiente enumeración:

Figura 1: Interfaz de la CCU BlackMagic

1. Botones de almacenamiento STORE: Al presionar el botón de STORE + botón de 1 o 2 de la parte superior izquierda se guardan los ajustes establecidos en la memoria 1 o 2 en el momento de presionar el botón. Presionando únicamente el botón 1 o 2 de memoria sin presionar previamente en botón de STORE, se recupera la información guardada en dicha memoria anteriormente.
2. Regulador de ganancia general. Esta ganancia varía de 0 a +18 dB y se regula en pasos de 6 dB.
3. Regulador del filtro de temperatura de color: en el indicador SHUTTER se puede ajustar la temperatura de color desde 1500K a 7000K en pasos de 50. Para ello se pulsan las flechas para aumentar o reducir dicha temperatura.

4. Botón barras de color: Al presionar el botón BARS, la cámara genera una carta de barras de color de 75% de saturación en la cámara. Si se vuelven a presionar, la cámara deja de generar dicha carta y muestra la imagen que obtiene por el objetivo.
5. Balance de blancos: Es posible ajustar el balance de blancos de cada una de las componentes con los tres botones giratorios de ganancia general de las tres componentes.
6. Controles para el balance de negros: La fila de mandos giratorios debajo de los mandos giratorios del balance de blancos se utilizan con la finalidad de ajustar el pedestal de los niveles de negro. Para realizar dichos cambios, basta con mover los controles de rojo, verde y azul hacia la izquierda o la derecha.
7. Control de apertura de diafragma y pedestal: La palanca de mando permite realizar ajustes precisos en los niveles de pedestal y apertura de diafragma. Al moverla hacia arriba o hacia abajo regula la apertura del diafragma. El indicador correspondiente se iluminará, proporcionando el valor de la exposición de la cámara. Por otro lado, la palanca cuenta con una rueda que permite controlar el nivel del pedestal. Al girar la rueda hacia la derecha o la izquierda, se aumenta o disminuye el nivel del pedestal respectivamente. Los valores de apertura de diafragma van desde f1 hasta f32 y los de pedestal desde -50mV hasta 400mV.

3.2 Matlab AppDesigner

El desarrollo de esta aplicación virtual se lleva a cabo utilizando la función AppDesigner del programa Matlab [3]. La interfaz de AppDesigner se divide en dos modos de vista: Design View y Code View, cada una de ellas con funciones específicas para facilitar el diseño y la programación de la aplicación.

Design View

En Design View, se modifica la interfaz de la aplicación. Esta vista se divide en tres partes principales:

1. **Interfaz Central:** En la parte central, representada por un recuadro negro en la figura 2, se observa la interfaz de la aplicación tal como la verá el usuario al ejecutarla. Aquí es donde se pueden ver y ajustar los elementos visuales de la aplicación.
2. **Lista de Componentes:** A la izquierda, en un recuadro rojo en la figura 2, se encuentra la lista de componentes que se pueden añadir a la interfaz. Estos componentes incluyen gráficos, imágenes, botones, entre otros. La interacción con estos componentes es simple: basta con arrastrar el componente deseado desde la lista hasta la interfaz central para integrarlo en el diseño.
3. **Componentes Creado:** A la derecha, en un recuadro azul en la figura 2, se muestra la lista de componentes que ya han sido creados, con sus respectivos nombres. Esta sección facilita la gestión y organización de los elementos añadidos a la interfaz, permitiendo acceder rápidamente a sus propiedades y ajustes.

Code View

La Code View es la parte de programación de la aplicación. Aquí es donde se crean y gestionan los códigos asociados a los componentes de la interfaz. Las principales características de esta vista son:

1. **Callbacks:** Cuando se añade un componente a la aplicación, se le puede dar funcionalidad mediante la creación de una callback asociada a dicho componente. Una callback es una función que se ejecuta cuando se modifica uno de los valores de los componentes de la aplicación. Por ejemplo, si se añade un botón, se puede programar una callback que defina qué sucede cuando dicho botón es pulsado por el usuario.
2. **Listado de Callbacks y Funciones:** En la parte izquierda de Code View, se encuentra un listado de las callbacks creadas, así como de las funciones externas utilizadas en el código. Esta lista facilita la navegación y organización del código, permitiendo acceder rápidamente a las diferentes partes de la programación de la aplicación.

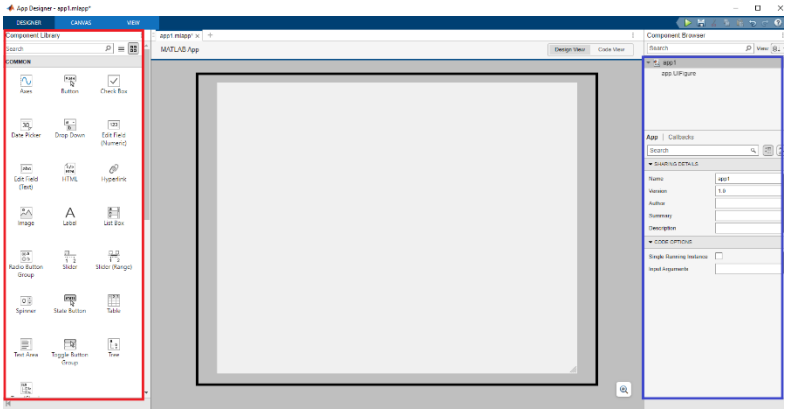

Figura 2: Design View de AppDesigner en Matlab

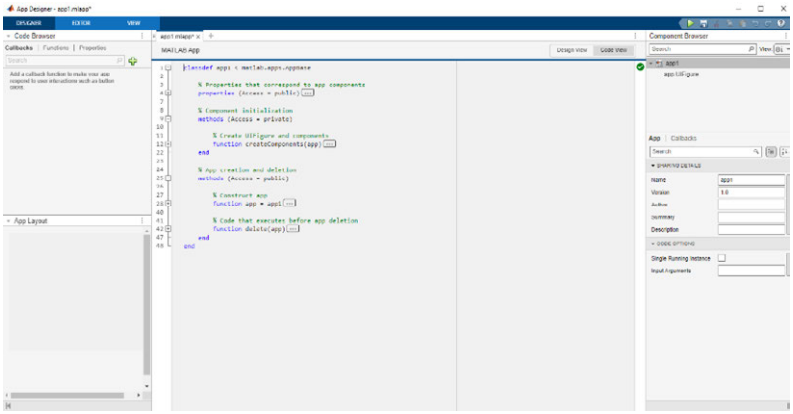

Figura 3: Code View de AppDesigner en Matlab

La aplicación creada con AppDesigner es compatible y ejecutable en cualquier versión de Matlab posterior a R2016a. Esto asegura que los usuarios con versiones recientes del software puedan utilizar la aplicación sin problemas de compatibilidad, garantizando una amplia accesibilidad y manejo.

En conclusión, AppDesigner de Matlab ofrece una plataforma robusta para el desarrollo de aplicaciones virtuales, combinando una interfaz de diseño intuitiva con potentes herramientas de programación. La división clara entre Design View y Code View permite a los desarrolladores concentrarse tanto en el aspecto visual como en la funcionalidad del software, asegurando que se puedan crear aplicaciones sofisticadas y fáciles de usar.

4. Antecedentes

Todo desarrollo se debe plantear como un problema de ingeniería a solventar. Por lo tanto, existen unas especificaciones iniciales que determinan la metodología y la solución. Las especificaciones están relacionadas con los objetivos del proyecto.

Se tiene como punto de partida de este proyecto la aplicación VirtualCCU 1.0. Esta aplicación tiene la interfaz de usuario de la figura 4 y se compone de distintas funciones que se resumen en la figura 5.

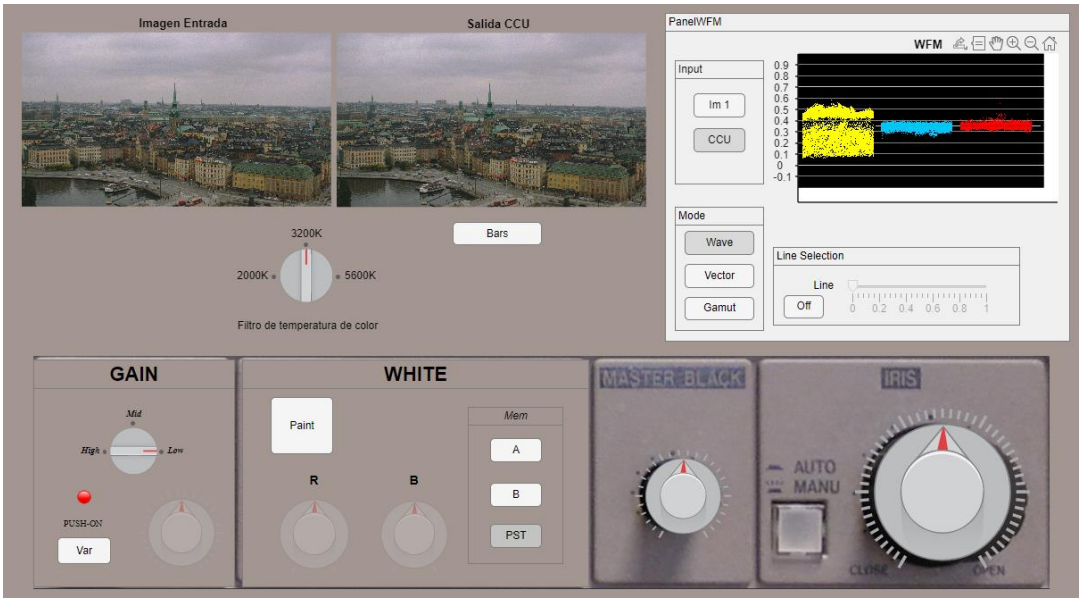

Figura 4: Interfaz de usuario de la aplicación VirtualCCU 1.0

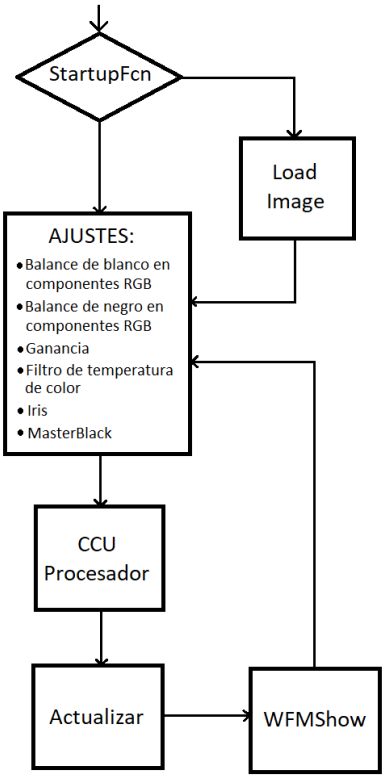

Figura 5: Diagrama del funcionamiento de la aplicación VirtualCCU 1.0

StartupFcn:

Es la función que se ejecuta al iniciar la aplicación. Primero crea la rejilla necesaria para mostrar los valores de R, G y B en el gráfico del WFM. Dicha rejilla se ejecutará como una matriz de datos donde el índice horizontal será el número de muestra y el índice vertical el valor de la intensidad lumínica de dicha muestra. Una vez hecho esto, se encarga de crear una estructura de datos llamada *Params*. Esta estructura es muy importante y la base de la aplicación ya que es donde se van a guardar los valores de los ajustes realizados por el usuario durante la utilización de la aplicación. Se compone de los siguientes datos:

- *GIris*: Esta variable corresponde a la apertura del diafragma. Este valor multiplica el valor de cada una de las muestras de todas las componentes cada vez que se haga un ajuste y será modificado cada vez que se haga un ajuste de Iris. Se inicializa con valor de 1, ya que al multiplicar por este valor no se obtendrá ningún cambio. Su valor oscila entre 0 y 2.
- *GBalR*: Esta variable corresponde al balance de blanco para la componente roja. Este valor multiplica el valor de cada una de las muestras de la componente roja cada vez que se haga un ajuste y será modificado cada vez que se haga un ajuste de balance de blanco para dicha componente. Al igual que la variable anterior, también se inicializa con un valor de 1. Su valor oscila entre 0,375 y 1,625.
- *GBalB*: Esta variable corresponde al balance de blanco para la componente azul. Este valor multiplica el valor de cada una de las muestras de la componente azul cada vez que se haga un ajuste y será modificado cada vez que se haga un ajuste de balance de

blanco para la componente azul. Al igual que la variable anterior, también se inicializa con un valor de 1. Su valor oscila entre 0,375 y 1,625.

- *GGlobal*: Esta variable corresponde a la ganancia general en decibelios de las componentes de la imagen. Este valor multiplica todos los valores de las componentes RGB de la imagen. Se inicializa con un valor de 1. Su valor oscila entre 1 y 4.
- *Black0*: Esta variable corresponde al pedestal. Este valor se suma al valor de cada una de las muestras de todas las componentes cada vez que se haga un ajuste y será modificado cada vez que se haga un ajuste de pedestal para dicha componente. A diferencia del resto de variables, se inicializa con un valor de 0, ya que al sumar este valor a las componentes no habrá ninguna modificación. Su valor oscila entre -0,3 y 0,3.
- *MemA*: Esta variable corresponde a los valores RGB que se guardan en esta memoria para recuperarse en un futuro. Para ello se utiliza una matriz de una fila por dos columnas donde cada valor corresponde al parámetro de balance de blanco de cada componente. Se inicia con la matriz [0.5 0.5] que corresponde a un valor 1 de *GBalbR* y *GBalB* que, como se ha explicado anteriormente, al multiplicar por el valor de las componentes de la imagen, no se produce ningún cambio en la imagen. Cuando se presiona el botón *MemA* para guardar un balance de blancos, en cada índice de la matriz se guarda el valor del balance de blanco de cada componente.
- *MemB*: Esta variable funciona exactamente igual que la *MemA*, lo único que corresponde al segundo banco de memoria de datos.
- *MemPST*: Esta variable corresponde a resetear los valores del balance de blancos. Por lo que, al querer volver a los ajustes de la imagen original, la matriz asociada a dicho parámetro tendrá un valor constante de [0.5 0.5], lo que corresponde a no ejecutar ningún ajuste de balance de blanco a la imagen.

Después de crear la estructura *Params,* en donde se irán almacenado los valores correspondientes a los ajustes que ofrece la aplicación, se carga la imagen Stockholm para visualizar una imagen de inicio al arrancar la aplicación. Para poder realizar ajustes sobre esta imagen, se asocia un matriz de 1080x1920 a cada componente R, G o B de la imagen. Estas 3 matrices se guardan en una variable llamada RGB1. Finalmente, la aplicación muestra dicha imagen tanto en 'Imagen de entrada' como en 'salida CCU' y muestra por el WFM el valor de cada muestra de las tres componentes.

Load Image:

Este bloque se encarga de cargar una imagen que el usuario seleccione de los archivos del ordenador en la aplicación. Su funcionamiento es igual que el de cargar la imagen Stockholm en *startupFcn,* solo que se añade un explorador de archivos del PC para seleccionar la imagen deseada.

Ajustes:

En esta parte es donde aparece por primera vez el usuario de la aplicación. En ella, el usuario puede hacer los siguientes ajustes: ajustes R y B en balance de blancos, dos botones A y B para guardar en memoria dos distintos ajustes de este tipo, un regulador de apertura de iris, un

botón regulador de pedestal, un regulador de ganancia general con valores discretos y un filtro de temperatura de color con 3 valores. Al realizar algunos de estos ajustes, la callback asociada a dicho ajuste calcula y cambia la variable de la estructura *Params* correspondiente al ajuste. Finalmente, se ejecuta la función externa *CCUProcesador*.

CCUProcesador:

La función *CCUProcesador* es la que se encarga de ejecutar y aplicar los ajustes realizados sobre las matrices RGB correspondientes a la imagen. Para ello, dependiendo de cada ajuste, realiza la operación matemática correspondiente sobre los valores de las componentes RGB de la imagen. Devuelve también 3 matrices de 1080x1920 correspondientes a cada componente de la imagen y las guarda en una variable llamada RGBS, que es la única variable de salida de esta función.

Actualiza:

Esta función simplemente muestra la imagen resultante tras los ajustes por "Salida CCU" y manda a la función *WFMShowfcn* las nuevas componentes.

WFMShow:

Muestra en el WFM las componentes resultantes recibidas tras los ajustes.

5. Actualizaciones de la aplicación

Como se ha comentado anteriormente, la incorporación de la nueva CCU al laboratorio implica nuevas funcionalidades de la herramienta. Dichas funcionalidades ya se han comentado en el apartado de objetivos. En este apartado se dispone a implementar dichas actualizaciones en la aplicación, incluyendo la nueva botonería en la interfaz.

5.1 Cambios en la interfaz

Con la renovación de la Unidad de Control de Cámara (CCU) en el laboratorio, la interfaz de usuario experimentó un cambio significativo. Originalmente, la aplicación presentaba una disposición de botones similar a la que se encuentra en las cámaras JVC estándar de televisión. En la primera versión de la aplicación, se intentó replicar esta disposición lo más fielmente posible. La interfaz inicial, como se muestra en la figura 4, presenta los controles de ajuste en la parte inferior de manera horizontal, al igual que en la cámara JVC. En la parte superior de la interfaz se visualizan la imagen de entrada, la imagen de salida y el monitor de forma de onda (WFM).

Para mejorar la interfaz y hacerla más intuitiva, se decidió transformarla en una nueva versión que se asemejara más a la CCU BlackMagic. Este cambio implicó una reestructuración completa de la disposición de los controles. En lugar de la disposición horizontal, los controles se organizaron verticalmente en el lado derecho de la aplicación. El WFM se movió a la parte inferior izquierda, situándose debajo de las imágenes de entrada y salida. Se utilizó una imagen de la CCU BlackMagic como referencia visual para colocar los botones en la aplicación programada de manera más coherente.

Uno de los mayores desafíos fue implementar la palanca que ajusta simultáneamente la apertura de iris y el pedestal, debido a la complejidad de la codificación. Para resolver esto, se dividió la función en dos controles separados: un botón giratorio grande para el pedestal y un deslizador vertical para el iris, situados uno al lado del otro.

La nueva CCU BlackMagic también incluye dos pequeñas pantallas digitales junto a la palanca de apertura de iris y pedestal, que muestran los valores aplicados de apertura de iris (en pasos de distancia focal) y pedestal (en milivoltios). En la aplicación, esto se implementó añadiendo dos cuadrados pequeños que muestran estos valores. El valor de pedestal se integra directamente ya que la estructura *Params* proporciona este valor en milivoltios. Para el valor de apertura de iris, se crea una función que asigna un valor de paso, entre f1 y f32, en función del valor de apertura del iris (entre 0 y 1).

Además, se cambian los colores de los botones y el fondo de la aplicación para que se fusionen adecuadamente con el color negro predominante en la CCU. El resultado final de estas modificaciones se puede observar en la figura 6.

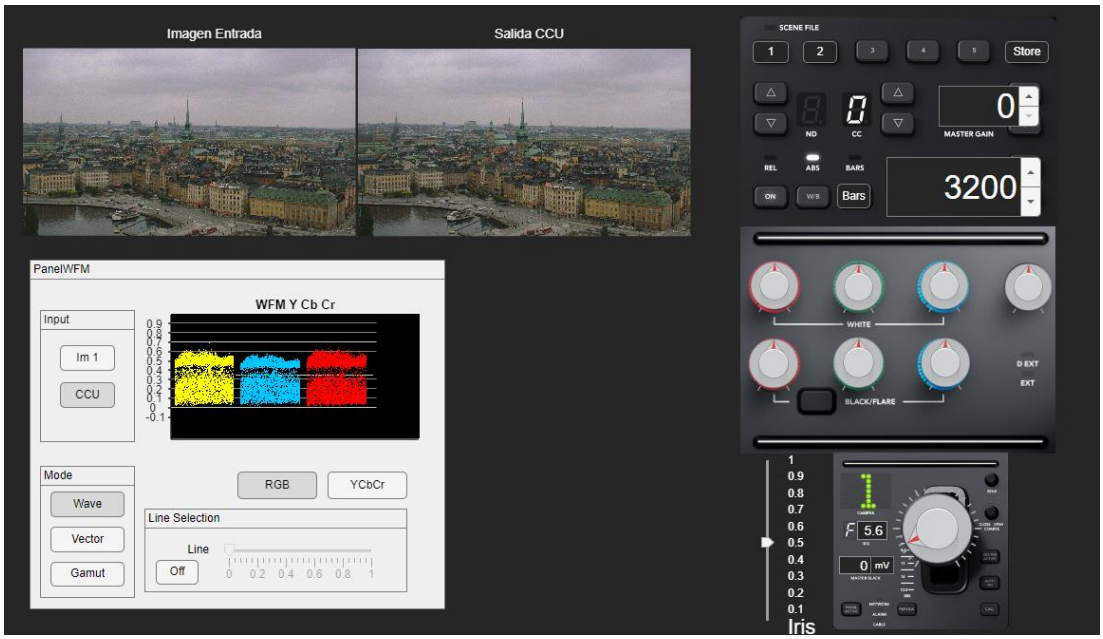

Figura 6: Interfaz de la aplicación VirtualCCU 2.0

La transformación de la interfaz no solo se centró en la estética, sino también en mejorar la funcionalidad y la experiencia del usuario. Al mover los controles a una disposición vertical en el lado derecho, se liberó espacio para una visualización más clara de las imágenes de entrada y salida, así como del WFM. Esta reorganización permite una interacción más natural y eficiente con los controles de la CCU.

5.2 Filtro de temperatura de color de valores continuos

En la versión anterior, para el desarrollo del botón de filtro de temperatura de color, se crea una función de callback que modifica el parámetro de ganancia de la componente azul (*GBalB*) de la estructura *Param* en función del valor del filtro de temperatura de color. Sin embargo, este desarrollo no es del todo correcto, ya que el filtro de temperatura de color no afecta solo a la componente azul de la imagen, sino que actúa como un filtro con valores diferentes para cada una de las tres componentes de color (rojo, verde y azul). Ademas, dicho filtro se implementa con un botón giratorio que solo ofrece la posibilidad de 3200K o 5600K.

Para abordar este problema, se ha creado la función *luzK2FiltRGB*, que toma como entrada una temperatura de color en grados Kelvin y devuelve los valores de un filtro de color complementario en el espacio de color RGB [4]. Finalmente, el valor de cada componente de este filtro se suma a las componentes correspondientes de la imagen. A lo largo de esta función, se realizan varias conversiones entre diferentes espacios de color y se llama a la función externa *kelvin2rgb* para obtener el resultado final.

La función *luzK2FiltRGB* toma un argumento de entrada, *luz_kelvin*, que representa una temperatura de color en grados Kelvin. La función comienza sumando al valor de Kelvin introducido en la aplicación 3400K, ya que el valor del filtro de temperatura de color referencia en la CCU debe ser 3200K, y en las fórmulas que se usan a continuación están adaptadas para un valor de referencia de 6600K. Despues, se llama a *kelvin2rgb* para convertir este valor de temperatura de color a valores RGB.

La función *kelvin2rgb* convierte una temperatura de color en grados Kelvin a valores RGB utilizando fórmulas específicas para cada componente de color. El proceso implica ajustar la temperatura a un rango válido (entre 1500 y 7000 grados Kelvin), dividirla por 100 para simplificar los cálculos, y luego aplicar diferentes fórmulas para calcular las componentes roja, verde y azul. Estas fórmulas están basadas en aproximaciones matemáticas optimizadas para ofrecer resultados visualmente precisos. Los valores RGB resultantes se aseguran de estar dentro del rango de 0 a 255 y se redondean al final para obtener valores enteros.

Una vez obtenidos los valores RGB de la temperatura de color, la función *luzK2FiltRGB* convierte estas componentes a YCbCr. La conversión a YCbCr se realiza utilizando la fórmula 1, 2 y 3 [5].

$$y\_luz = 0.2126 * \mathrm{r\_luz} + 0.7152 * g\_luz + 0.0722 * b\_luz \quad (1)$$

$$cb\_luz = \frac{b\_luz - y\_luz}{1.8556} \quad (2)$$

$$cr\_luz = \frac{r\_luz - y\_luz}{1.5748} \quad (3)$$

Para obtener el color complementario, se invierten las componentes de crominancia (*cb_luz* y *cr_luz*), mientras que se mantiene la luminancia (*y_luz*) sin cambios [6]. Esto da como resultado las variables *cb_filt* y *cr_filt*.

El color complementario en YCbCr se convierte de nuevo a RGB utilizando las fórmulas 4, 5 y 6.

$$r\_filt = \text{y\_filt} + 1.4025 * cr\_filt \quad (4)$$

$$g\_filt = \text{y\_filt} - 0.3443 * cb\_filt - 0.7144 * cr\_filt \quad (5)$$

$$b\_filt = \text{y\_filt} + 1.7730 * cb\_filt \quad (6)$$

Después de la conversión, se asegura que las componentes RGB no exceden el valor máximo de 255. Si alguna componente supera este valor, todas las componentes se escalan proporcionalmente para que la componente mayor se ajuste a 255.

Finalmente, los valores RGB se escalan al rango [0, 1] para que sean adecuados para usos posteriores en gráficos o visualización. Este escalado se realiza dividiendo cada componente por 255.

Para la interfaz de la aplicación se ha pasado de un botón giratorio de valores discretos a un *spinner* como el de la figura 7, de tal manera que el filtro de temperatura de color podrá tomar valores enteros continuos entre 1500K y 10000K.

Figura 7: Tipo de botón utilizado para el filtro de temperatura de color

5.3 Botón giratorio para componente verde en balance de blancos

En la aplicación VirtualCCU 2.0, como es normal para la gran mayoría de CCUs, solo se implementa botones giratorios para ajustar la ganancia de las componentes en balance de blanco para las componentes roja y azul.

En la aplicación VirtualCCU se implementa ademas un botón giratorio para ajustar la ganancia de la componente verde. La función responsable de dicho ajuste es *GKnob_ValueChanged*. Esta función se ejecuta cuando se cambia el valor del botón giratorio correspondiente al balance de blancos para la componente verde. Para su funcionamiento, la función utiliza varias variables globales definidas fuera de ella, a saber:

- *Y1, Cb1, Cr1*: Matrices que representan componentes de color de la imagen en formato YCbCr.
- *RGB1*: Matriz de la imagen en formato RGB.
- *esperar*: Indicador que evita la ejecución simultánea de ciertos bloques de código.
- *Params*: Estructura de parámetros de la aplicación.

En la estructura *Params*, siguiendo con la misma estructura de código de la aplicación VirtualCCU 1.0, se crea previamente la variable *GBalG*, destinada a almacenar el ajuste correspondiente a la componente verde. Al igual que las otras componentes del balance de blancos, esta variable se inicia con el valor 1, lo que indica que no se realiza ningún ajuste inicial.

Para implementar este ajuste en AppDesigner, se utiliza un botón giratorio como el ilustrado en la figura 8. El valor de este botón se iniciará en 0.5, puede variar entre 0 y 1, y se almacenará dentro de la función en la variable *valorBoton*. Para traducir el valor de *valorBoton* al valor de *GBalB*, se usa la fórmula 7.

$$GBalB = \frac{\text{valorBoton} + 0{,}3}{0{,}8} \tag{7}$$

Con la fórmula 7, el valor de *GBalB* varía entre 0.375 (atenuación del 62.5%) y 1.625 (amplificación del 62.5%). Cuando el valor inicial del botón giratorio es 0.5, la variable *GBalB* tendrá un valor de 1.

Una vez obtenido el valor *GBalB*, se llama a la función *CCUProcesador* que aplica el ajuste de balance de blancos a la componente verde de la imagen. Esto se logra multiplicando cada valor de la matriz RGB1 correspondiente a la componente verde por el valor calculado de *GBalB*.

Finalmente, se llama a la función actualizar que se encarga de mostrar la imagen ajustada en la interfaz de la aplicación y de actualizar los valores de las componentes en la pantalla del

WFM. Esto asegura que los usuarios de la aplicación puedan ver en tiempo real cómo se aplican los ajustes de balance de blancos.

Figura 8: Botón giratorio tipo Knob

5.4 Balance de negro con botón giratorio para las componentes RGB

Se puede entender mejor esta funcionalidad haciendo una analogía con los tres botones giratorios ya implementados para el balance de blancos. A diferencia de estos últimos, esta vez la finalidad es crear una variable para cada componente que, en vez de multiplicar, se sumará al valor de cada componente de la imagen. En otras palabras, mientras que los botones giratorios de balance de blancos modifican las componentes de color mediante la multiplicación, los nuevos botones giratorios para el balance de negro ajustarán las componentes sumando un valor específico a cada una.

Para implementar esto, se han añadido tres botones giratorios como el de la figura 8, uno para cada componente de color que se quiere ajustar para el balance de negro. Los botones giratorios correspondientes a las funciones se llaman *RKnob_BlackValueChanged*, *GKnob_BlackValueChanged* y *BKnob_BlackValueChanged*. Estas funciones utilizan varias variables globales que se definen fuera de la función, lo que asegura una integración coherente con el resto de la aplicación. Las variables globales son:

- *Y1, Cb1, Cr1*: Matrices que representan componentes de color de la imagen en formato YCbCr.
- *RGB1*: Matriz de la imagen en formato RGB.
- *esperar*: Indicador que evita la ejecución simultánea de ciertos bloques de código.
- *Params*: Estructura de parámetros de la aplicación.

Para almacenar los valores de estos botones giratorios, se crean previamente tres nuevas variables en la estructura *Params*: *Black0R* (para la componente roja), *Black0G* (para la componente verde) y *Black0B* (para la componente azul). Estas variables se inician con un valor de 0.11, permitiendo que el ajuste sea neutro al principio, y pueden variar entre 0 y 1. Dicho valor se transformaría a la variable de la estructura *Params* correspondiente siguiendo la siguiente fórmula 8 (se toma como ejemplo la componente azul).

$$Black0B = 0.45 * \text{valorBoton} - 0.05 \quad (8)$$

Así, las variables correspondientes al ajuste de balance de negro varían entre -0.05 V y 0.4 V, comenzando con un valor neutral de 0 V. Esta fórmula permite una gama de ajuste precisa que puede ser fácilmente controlada por el usuario.

Previamente, se añade código en la función *CCUProcesador* para que las variables Black0R, Black0G y Black0B se sumen a los valores de la matriz RGB1 correspondiente a las componentes R, G y B respectivamente. De esta manera, las funciones *RKnob_BlackValueChanged*, *GKnob_BlackValueChanged* y *BKnob_BlackValueChanged* llaman a la función *CCUProcesador*, esta aplica el ajuste de balance de negro a cada componente de la imagen sumando el valor de las variables *Black0R*, *Black0G* y *Black0B* a los valores correspondientes de la matriz *RGB1*.

Finalmente, la función *actualiza* se encarga de mostrar la imagen ajustada en la interfaz de la aplicación y de actualizar los valores de las componentes en la pantalla del WFM. Esto asegura que los usuarios puedan ver en tiempo real cómo se aplican los ajustes de balance de negro.

5.5 Ganancia general para las 3 componentes en el balance de blancos

Para regular la ganancia general de las tres componentes durante el proceso de balance de blancos, se ha desarrollado un botón giratorio similar a los utilizados para el balance de cada componente por separado. Se ha añadido una nueva variable en la estructura *Params* llamada *GBalRGB*, la cual se inicia en 0,5 y puede variar entre 0 y 1. El valor del botón giratorio se traducirá a *GBalRGB* usando la fórmula 9.

$$GBalRGB = \frac{\text{valorBoton} + 0{,}3}{0{,}8} \tag{9}$$

Con esta fórmula, el valor de *GBalRGB* oscila entre 0,375 (lo que implica una atenuación del 62,5%) y 1,625 (una ganancia del 62,5%). El valor inicial será 0.5, lo que significa que no hay ni ganancia ni atenuación aplicada. Para la interfaz de la aplicación en AppDesigner se empleará un botón similar al mostrado en la figura 8.

5.6 Implementación de las memorias de almacenamiento 1 y 2

Como se ha mencionado en el apartado 2, la nueva CCU BlackMagic dispone de una función de almacenamiento que permite cargar y guardar ajustes de balance de blanco y balance de negro en dos memorias distintas: memoria 1 y memoria 2. Esto ya existe en la aplicación VirtualCCU 1.0, pero con un funcionamiento algo distinto al de la nueva CCU por lo que se decide cambiar el código para llegar a un resultado fiel al funcionamiento de la actual CCU. Esta funcionalidad se implementa mediante dos botones de memoria (memoria 1 y memoria 2) y un botón de almacenamiento (store). El diagrama de flujo del código desarrollado para esta funcionalidad se detalla en el diagrama de la figura 9.

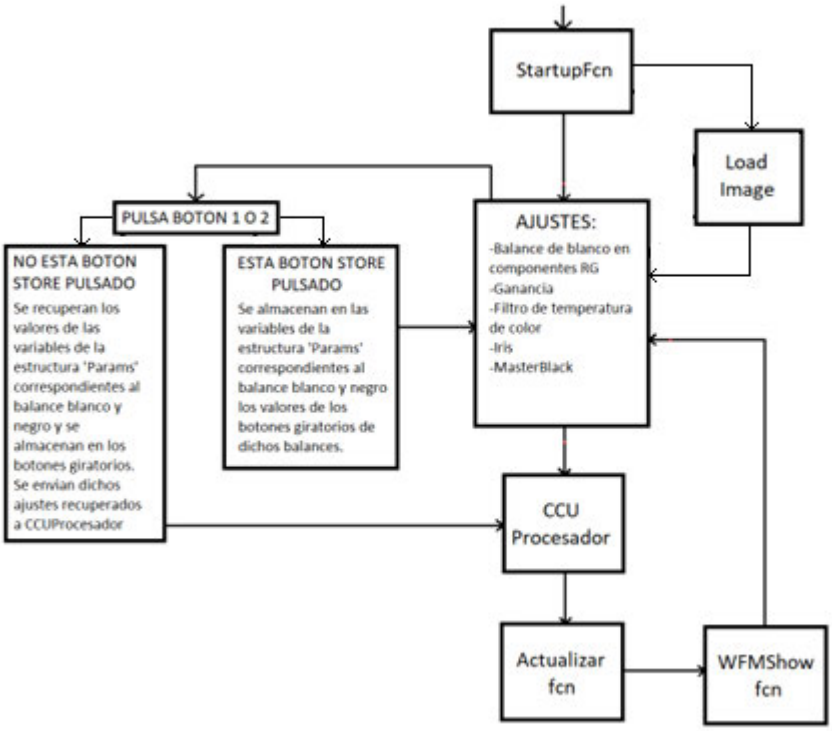

Figura 9: Diagrama del funcionamiento de los botones de almacenamiento

El código de la función se divide en cuatro procesos distintos:

1. **Inicialización de Variables:**

   Se crean cuatro nuevas variables en la estructura *Params*: *MemA*, *MemB*, *MemABlack* y *MemBBlack*. Cada una de estas variables es un array de tres elementos que representan el valor de los botones giratorios de las componentes de color R, G y B.

Estas variables se inicializan con el valor [0.5, 0.5, 0.5], lo que indica que no se ha aplicado ningún ajuste inicial a las componentes de color.

2. **Función de Almacenamiento:**

Cuando se pulsa el botón de almacenamiento (store), el sistema queda en espera de que se seleccione una de las memorias (memoria 1 o memoria 2).

Al pulsar el botón de una memoria específica, los valores actuales de los botones giratorios de balance de blanco y negro se almacenan en las variables correspondientes de esa memoria.

3. **Guardado de Ajustes:**

Los valores de los botones giratorios de balance de blanco (*GBalR*, *GBalG*, *GBalB*) y de balance de negro (*Black0R*, *Black0G*, *Black0B*) se guardan en las variables *MemA* y *MemB* para balance de blanco, y en *MemABlack* y *MemBBlack* para balance de negro.

4. **Recuperación de Ajustes:**

Cuando se pulsa uno de los botones de memoria (memoria 1 o memoria 2) sin pulsar previamente el botón de almacenamiento (store), el sistema recupera los valores almacenados en la memoria seleccionada.

Estos valores se cargan en los botones giratorios correspondientes, restableciendo los ajustes de balance de blanco y negro a los valores previamente guardados en la memoria seleccionada.

Esta funcionalidad de memoria permite a los usuarios guardar configuraciones personalizadas de balance de blanco y negro y recuperarlas rápidamente cuando sea necesario.

6. Añadir opción de importar vídeos a la aplicación

Una de las funcionalidades que se plantea añadir a lo largo de este proyecto es la opción de realizar los ajustes de la aplicación tanto a imágenes como a vídeos. Esto presenta una serie de retos significativos a la hora de programarlo, pero el objetivo es permitir que los usuarios puedan aplicar los mismos ajustes que se hacen en imágenes fijas a secuencias de vídeo. A continuación, se detallan los desafíos y soluciones implementadas para lograr esta funcionalidad.

6.1 Importar vídeo

En Matlab existe la opción de leer archivos en formato vídeo con el comando *videoreader('nombredelarchivo').* Este comando transforma un vídeo en una sucesión de imágenes (cada imagen corresponde a un fotograma del vídeo) cada una de ellas transformada en un conjunto de tres matrices de 1920 filas por 1080 columnas. Planteando así el problema, se propone tratar independientemente cada imagen con los ajustes de la aplicación y mostrarlas por pantalla sucesivamente, dando la sensación de estar viendo el vídeo con los ajustes aplicados.

El problema es que cada vídeo suele contener 32 imágenes por segundo, y cada imagen contiene 2 073 600 valores por cada componente. Además de esto, por cada ajuste realizado en la interfaz, la aplicación tiene que realizar un cálculo por cada uno de estos valores, sumado a que cada componente después es transformada para ser presentada por el WFM. Todo esto hace que el número de operaciones por segundo sea muy elevado y, pese a realizarlo con ordenadores de gran capacidad de procesamiento, la velocidad de reproducción del vídeo en la aplicación era muy lenta (más de 3 segundos por cada fotograma).

Se piensa en reducir la resolución de cada imagen, bajándola a una resolución de 480x240 usando la función externa *redimensionarImagen* (lo que reduce 18 veces el número de valores) pero aun así la velocidad de reproducción de video es lenta (aproximadamente 2 segundos por fotogramas). El inconveniente no reside en el número de valores, si no en que el comando *videoreader* crea una matriz de cuatro dimensiones, lo que agota la capacidad de procesado de memoria temporal de Matlab.

Gracias al profesor Enrique Rendón, que tuvo un problema similar en uno de sus proyectos, se decide cambiar la perspectiva de la solución. Esta consiste en dividir los fotogramas del vídeo en imágenes independientes previamente a ser cargados en la aplicación [7]. De esta manera, la aplicación carga una carpeta que contiene todas las imágenes (fotogramas) del vídeo y guarda cada una de ellas en una matriz de tres dimensiones independiente, correspondiendo cada una de ellas a las capas RGB de la imagen. Esta función sigue la siguiente estructura:

- Se abre un cuadro de diálogo para seleccionar el directorio que contiene las imágenes. Se guarda el directorio actual, se cambia al directorio seleccionado y se obtiene una lista de todos los archivos en ese directorio.
- Se inicializan varias estructuras de datos (*cell arrays*) para almacenar las imágenes y sus componentes transformadas en matrices de 1920x1080x3, que finalmente son redimensionadas a matrices de 480x240x3.
- Se inicia un bucle que recorre la lista de archivos en el directorio seleccionado, buscando archivos .bmp. Si se encuentra un archivo válido, se carga la imagen, se convierte al formato YCbCr y se guarda en las estructuras de datos correspondientes.
- Se verifica si se encontraron y cargaron imágenes válidas. Si no, se muestra un mensaje y la función termina.
- Una vez cargadas las imágenes, se reproducen en bucle. Para cada una de ellas:
  1. Se redimensiona y se muestra por la interfaz en 'Imagen de entrada'.
  2. Se recuperan las componentes YCbCr.
  3. Se convierten de nuevo a RGB para su posterior tratamiento.
  4. Se procesan los datos mediante la función *CCUProcesador* y se actualiza la interfaz mediante la función *actualizar*.

De esta manera, se aumenta considerablemente la velocidad de reproducción, pese a ser todavía algo lenta. Si bien es una solución para el problema de reproducción, no permite cargar cualquier archivo de vídeo como se pensó inicialmente, si no una carpeta que contenga los fotogramas del vídeo en formato bmp.

6.2 Reproducción en bucle de los vídeos y ajustes simultáneos en reproducción

Cuando se propone al inicio del proyecto la posibilidad de añadir vídeos en la aplicación, se idea que fueran vídeos cortos que se reproduzcan en bucle en la aplicación. El vídeo original se reproduce por la salida "Imagen Entrada" y el vídeo con los ajustes realizados por "Imagen Salida". Cuando el usuario hace un ajuste en alguno de los parámetros de la aplicación, los dos vídeos siguen reproduciéndose desde el fotograma en el cual se hizo el ajuste.

La reproducción en bucle se programa de manera sencilla creando un bucle *for* en el cual una variable independiente lleva el índice de número de fotograma y, cuando este llega al último fotograma del vídeo, empieza de nuevo por el primer fotograma. El problema que esto genera es que, cuando el usuario toca algún botón de la interfaz, la *callback* asociada a dicho botón no se activa, ya que esta acción pasa a la cola de acciones de Matlab, que hasta que no acaba el bucle *for* de la reproducción del vídeo en bucle, no da paso a dicha acción.

Se lleva a cabo un trabajo de investigación sobre cola de acciones e interrupciones en Matlab. Efectivamente, sí se puede dar un rango de interrupciones dentro de AppDesigner, pero solo entre *callbacks*. Esto quiere decir que se puede estar ejecutando una *callback* determinada y

que, si el usuario toca un botón que activa otra *callback*, esta última puede interrumpir la primera y pasar a ejecución. Pero esto no es posible entre un bucle y una *callback*, al menos en AppDesigner.

Por lo que se decide eliminar la opción de reproducción en bucle. Cuando un vídeo es cargado en la aplicación, este se reproduce solo una vez. Cuando el usuario realiza algún ajuste de la interfaz de la aplicación, este no se aplica hasta que el vídeo termina dicha reproducción.

Como se ha comentado en el apartado anterior, cuando el usuario realiza un ajuste durante la reproducción del vídeo por la aplicación, este no se efectúa hasta que dicha reproducción finaliza. Esto se puede realizar ya que Matlab tiene una cola de espera de acciones. Es decir, mientras se está ejecutando el bucle *for*, Matlab detecta que hay una *callback* activa a la espera y la deja en cola hasta que el bucle finaliza. Pero esta memoria es muy limitada y solo tiene espacio para una acción. En definitiva, si el usuario realiza dos o más ajustes durante la reproducción del vídeo, solo la última acción es la que se guarda en cola, perdiendo las acciones anteriores.

Para evitar este problema y limitar el número de acciones mientras se reproduce el vídeo a una sola acción, se decide que cuando el usuario realiza algún ajuste de la interfaz que ejecuta una *callback* y, por tanto, reproduce el vídeo por la aplicación, el resto de botones de la aplicación se bloquean hasta que la reproducción termina. De esta manera, el usuario solo podrá tener una *callback* pendiente en la cola de acciones.

6.3 Implementación de los ajustes sobre vídeo

Una vez superados los problemas de los apartados anteriores, la resolución del código para poder implementar los ajustes que se realizan sobre imágenes a vídeo es relativamente sencilla. Al guardar los fotogramas del vídeo en matrices de 480x240x3, que a su vez se guardan en un array de matrices cuyo índice corresponde al número de fotogramas del vídeo, la implementación de los ajustes se realiza de manera análoga a como se hacía con las imágenes, con la diferencia de que se realiza secuencialmente para todas las imágenes del video.

Es decir, cuando se carga el vídeo en la aplicación, cada fotograma se guarda en una matriz que ocupa un índice del array, y se guarda en una variable independiente el número de fotograma del video o, lo que es lo mismo, el número de índices del array de matrices. Además, se activa la variable *hayVideo,* que indica si hay un video cargado en la aplicación. De esta manera, cuando se realiza un ajuste en la aplicación, este ajuste carga la *callback* correspondiente. En dicha *callback*, si la variable *hayVideo* está desactivada, actúa como se ha descrito en apartados anteriores para imágenes. Si la variable *hayVideo* esta activa, se inicia un bucle *for* en el cual, para cada fotograma del vídeo, se realiza el mismo procedimiento que para las imágenes. Como se ha comentado en el apartado anterior, durante este bucle *for* se

desactivan el resto de botones de la aplicación para evitar conflicto en las peticiones de *callbacks*.

Procediendo de esta manera, se consigue que cada fotograma del vídeo se ejecute en el tiempo que tarda en procesar una imagen independiente (menos de 1 segundo), dando una sensación de fluidez en las imágenes de salida y, por tanto, asemejándose a una reproducción de vídeo tanto en la secuencia de fotogramas originales como en la de los fotogramas ajustados.

La implementación de la funcionalidad para importar y ajustar vídeos en la aplicación representó un desafío significativo debido a la gran cantidad de datos y procesamiento requerido. Sin embargo, mediante la división de los fotogramas en imágenes individuales y la gestión cuidadosa de las colas de acciones y *callbacks* en Matlab, se logra una solución eficiente que permite a los usuarios aplicar ajustes de color tanto a imágenes fijas como a vídeos. Aunque la solución final no permite la carga directa de archivos de vídeo y requiere que los fotogramas se almacenen en una carpeta, esta restricción es una carencia razonable y asumible dado el aumento en la velocidad de procesamiento y la fluidez en la reproducción del vídeo ajustado.

7. Resultados

Para analizar los resultados de la aplicación VirtualCCU 2.0 se realizan las pruebas correspondientes a una práctica tipo del laboratorio de la asignatura de Ingeniería de video ya que es la asignatura en la cual los alumnos utilizarán dicha aplicación.

Primero, se prueba la visión general de la app al abrir la aplicación (figura 10), al cambiar el WFM a componentes YCbCr (figura 11) y al cambiar el modo del monitor a modo vector (figura 12) y modo gamut (figura 13).

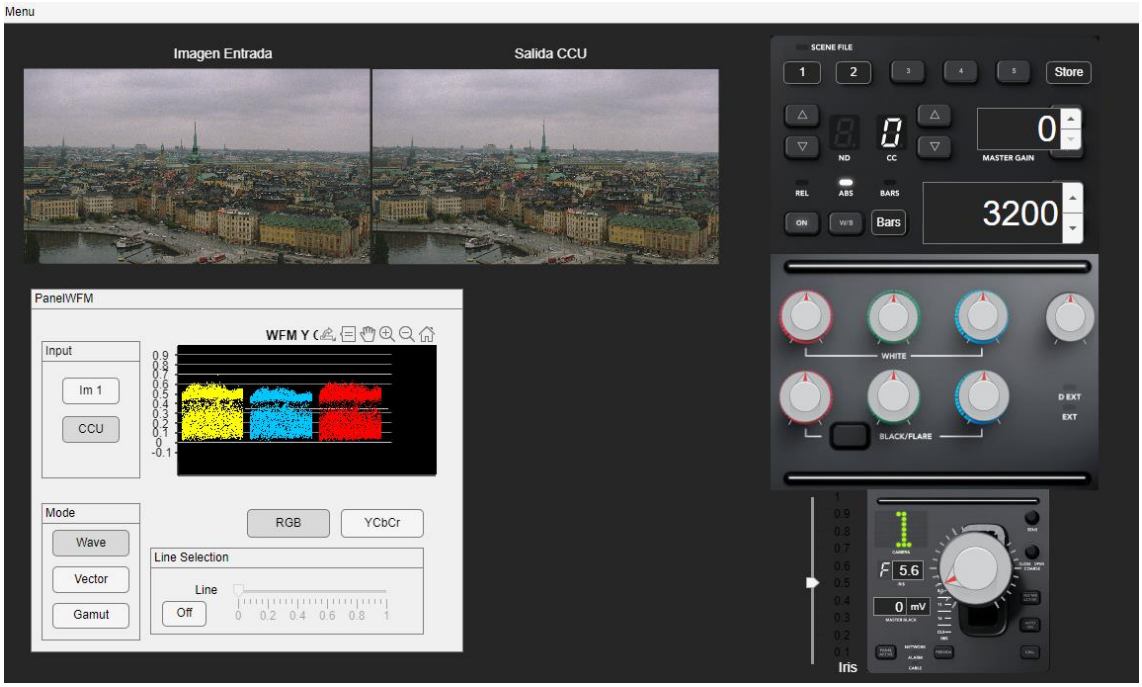

Figura 10: interfaz de usuario al iniciar la aplicación con el WFM en componentes RGB

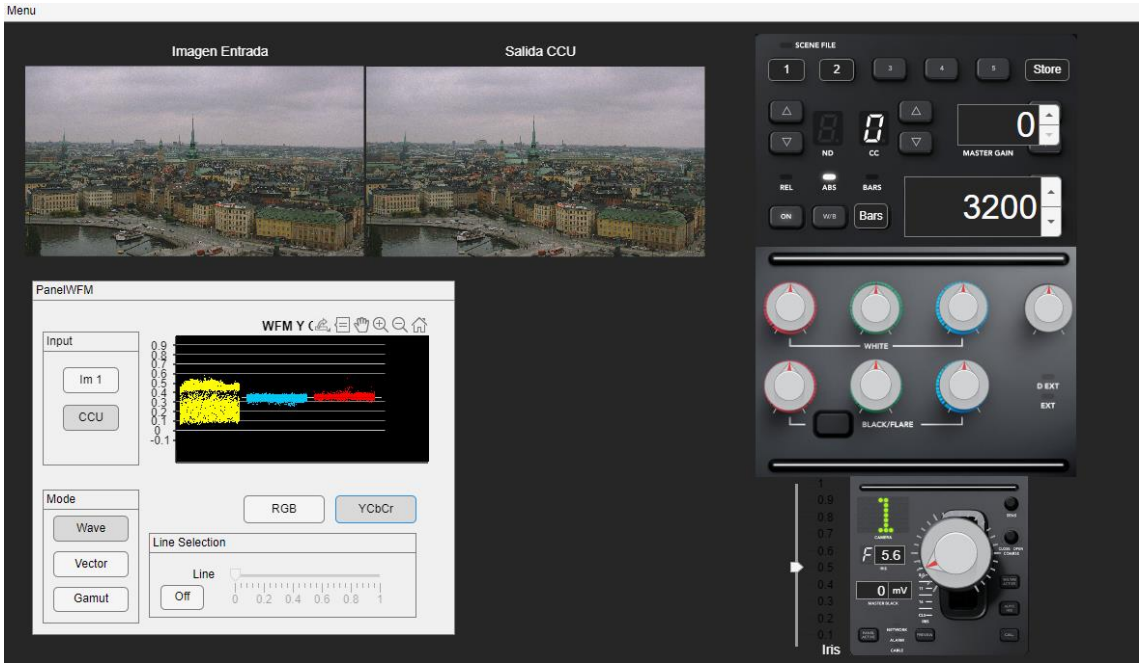

Figura 11: Interfaz de usuario de la aplicación al cambiar el WFM a componentes YCbCr

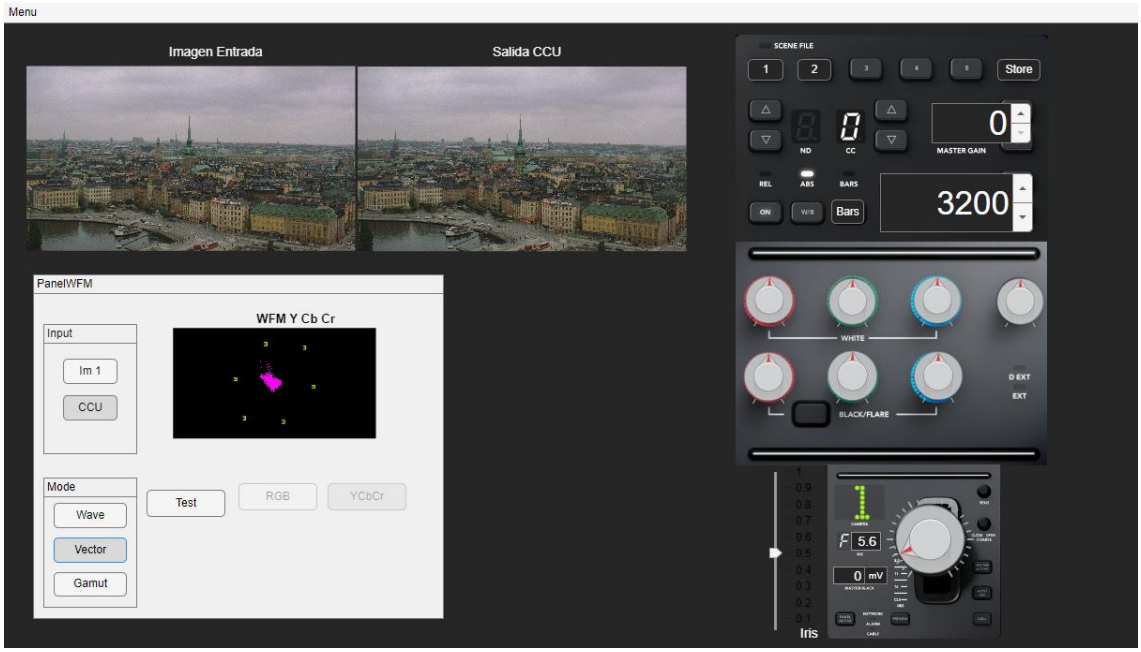

Figura 12: Interfaz de usuario de la aplicación al cambiar el WFM a modo vector

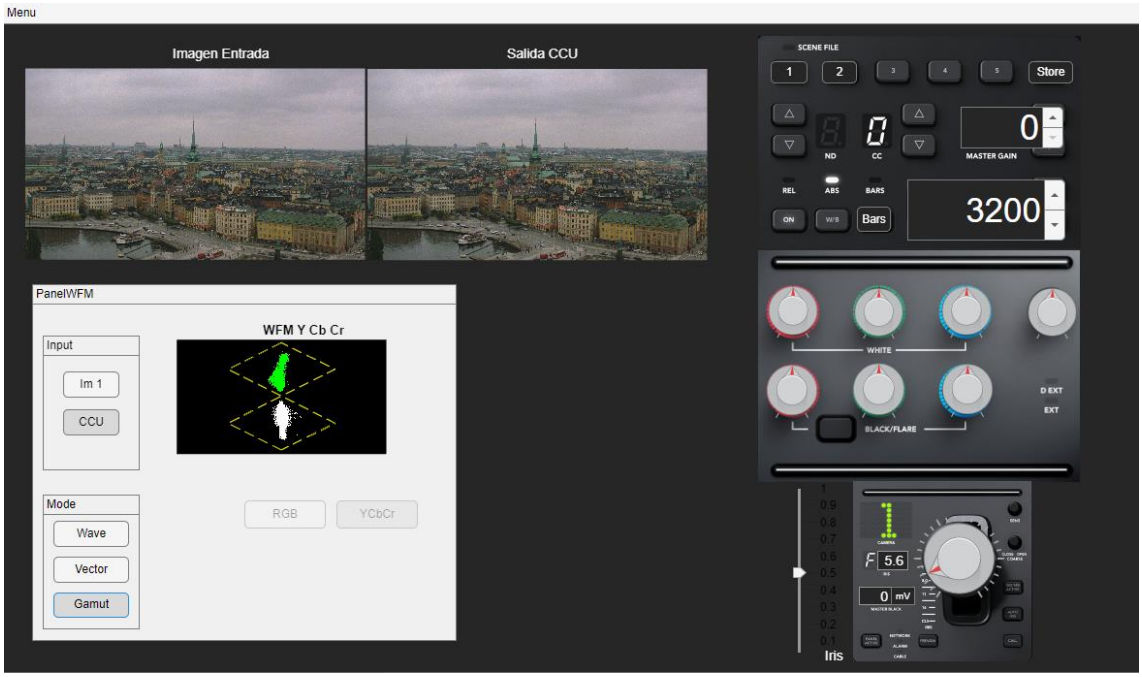

Figura 13: Interfaz de usuario de la aplicación al cambiar el WFM a modo gamut

Ahora, se prueban las funcionalidades de los botones de almacenamiento. Para ello, se realizan unos ajustes aleatorios como se ve en la figura 14 y se guardan los cambios en la memoria 1. Después, se realizan otros ajustes como se observa en la figura 15 y se guardan en la memoria 2.

Figura 14: Interfaz de usuario de la aplicación al guardar ajustes en memoria 1

Figura 15: Interfaz de usuario de la aplicación al guardar ajustes en memoria 2

Una vez guardados los ajustes en las memorias correspondientes se procede a cambiar de una memoria a otra (figura 16 e figura 17) y se observa como los ajustes guardados se aplican bien en la imagen.

Figura 16: Interfaz de usuario de la aplicación al recuperar los ajustes de la memoria 1

Figura 17: Interfaz de usuario de la aplicación al recuperar los ajustes de la memoria 2

Ahora se procede a hacer una prueba del balance de blanco. Para ello, se carga en la aplicación la imagen “pared blanca” (imagen que se utiliza en la práctica de la asignatura para dicho balance) como se observa en la figura 18.

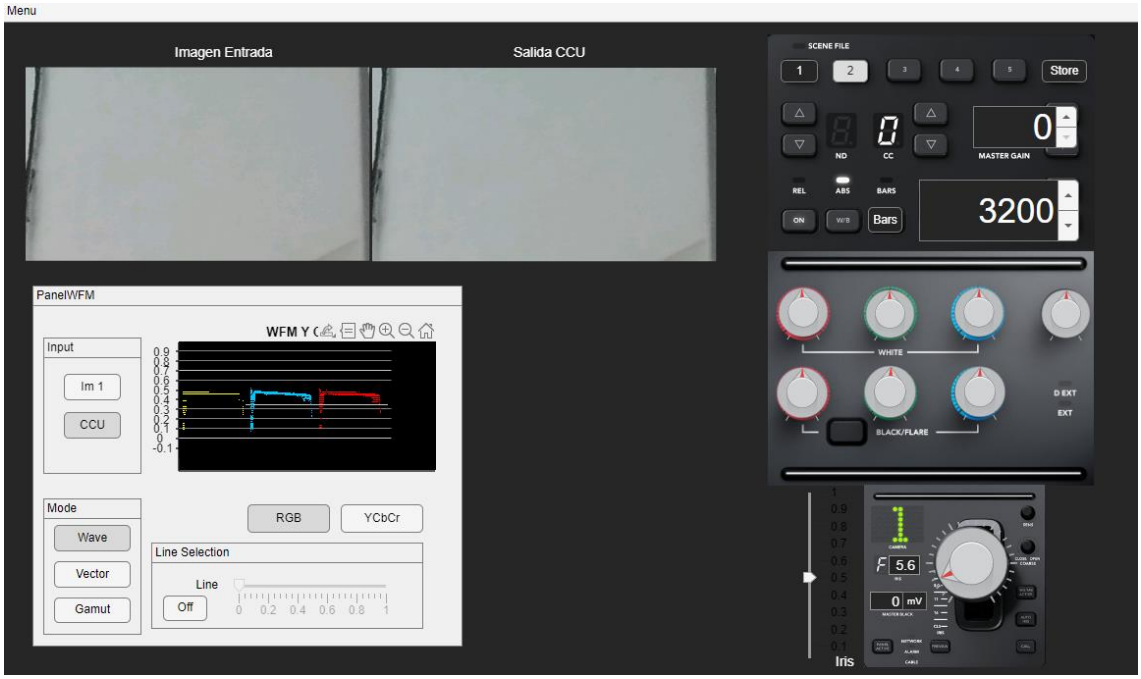

Figura 18: Interfaz de usuario de la aplicación al cargar la imagen "pared blanca"

Se realiza un balance de blanco ajustando la apertura de IRIS y la ganancia de cada componente para que las tres componentes estén igualadas y a 700 mV y se guarda en la memoria 2 (figura 19).

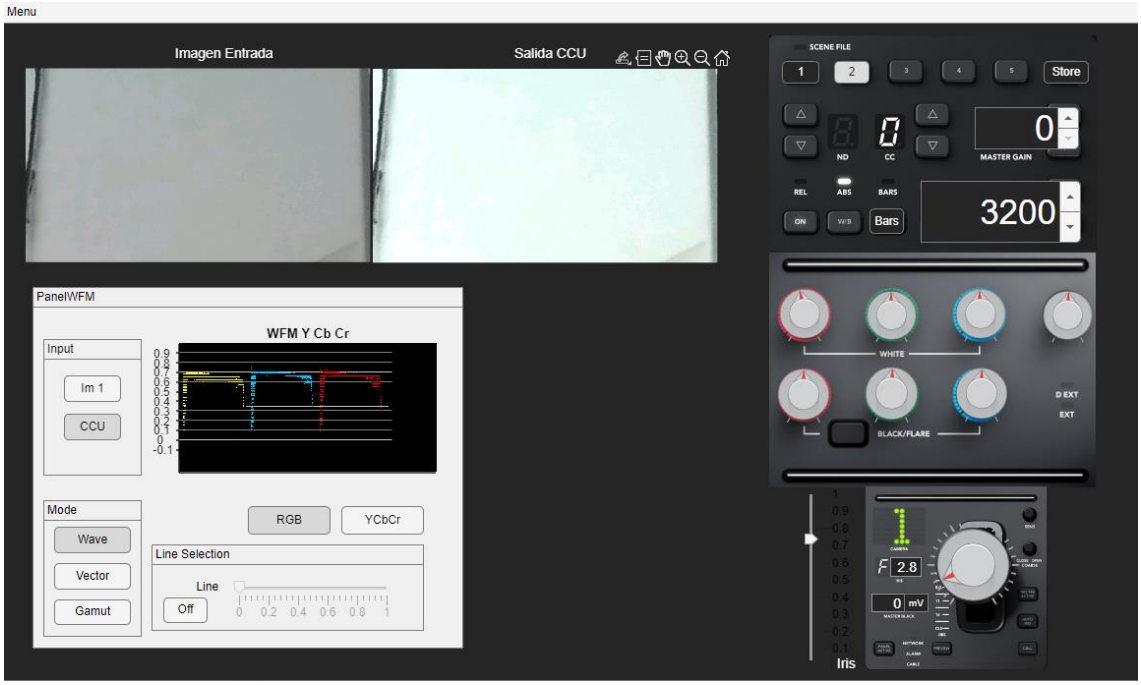

Figura 19: Interfaz de usuario de la aplicación tras realizar balance de blancos sobre la imagen "pared blanca"

Se han realizado tambien diversas pruebas independientes para comprobar el funcionamiento de todos los botones de la aplicación. Sobre la imagen “pared blanca” se realiza un ajuste de ganancia de +6dB (figura 20), despues se sube el pedestal 400mV (figura 21) y finalmente se aumenta la ganancia de la componente roja en balance de blanco (figura 22).

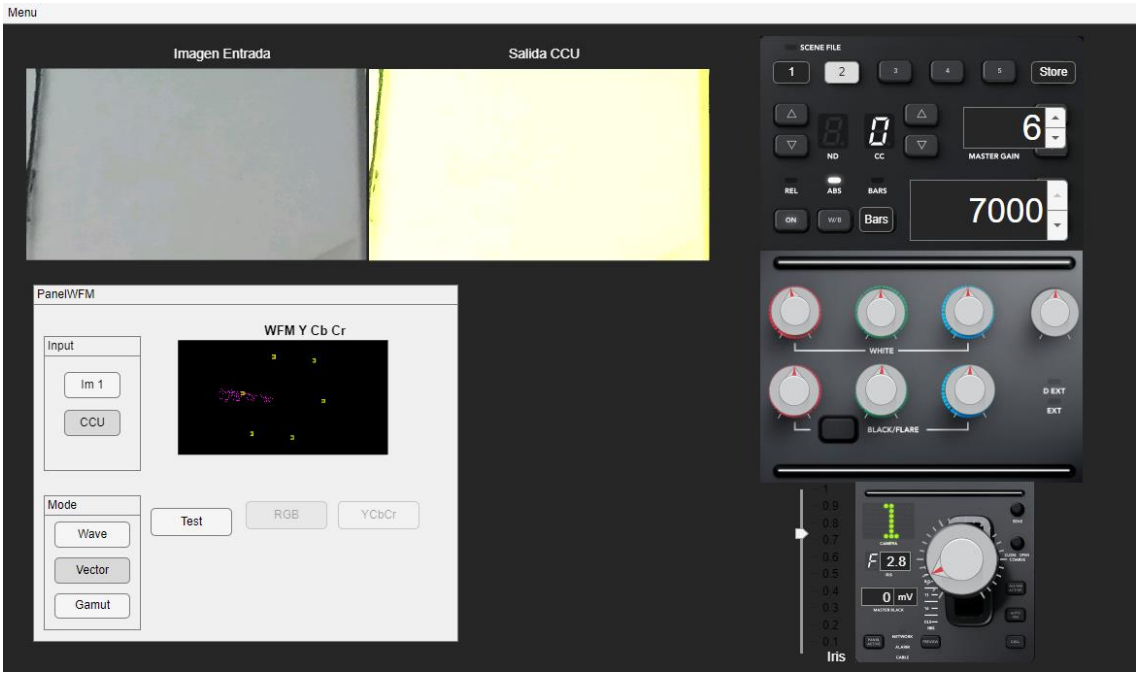

Figura 20: Interfaz de usuario de la aplicación al aplicar 6dB de ganancia sobre la imagen “pared blanca”

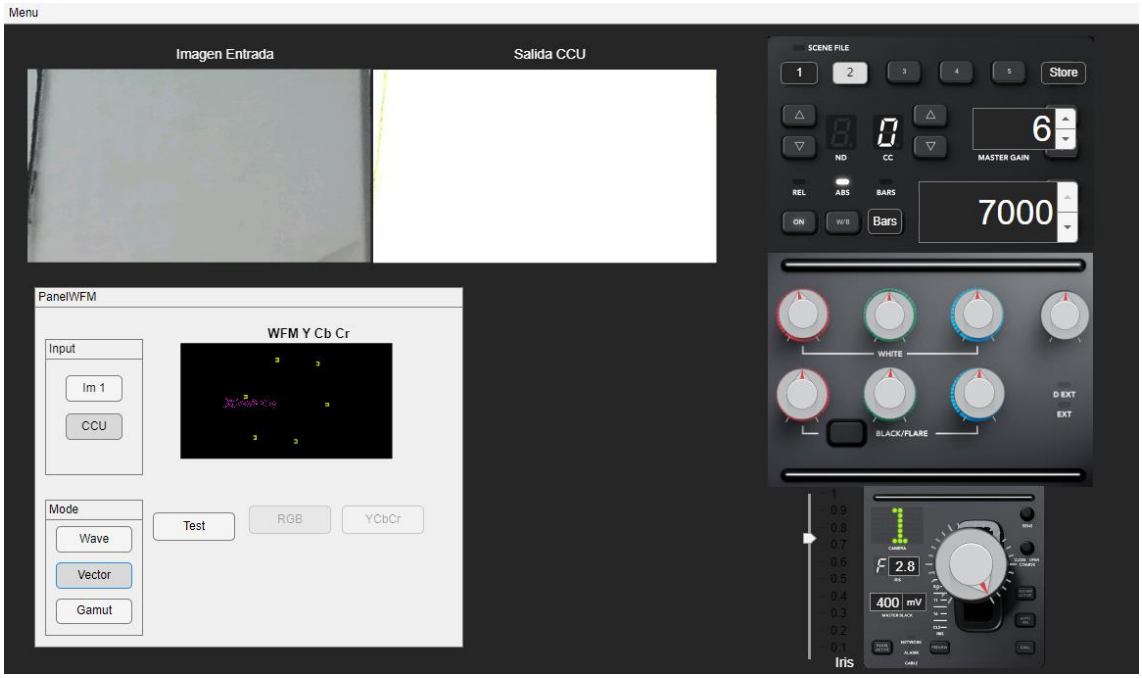

Figura 21: Interfaz de usuario de la aplicación al aplicar 400mV de pedestal sobre la imagen “pared blanca”

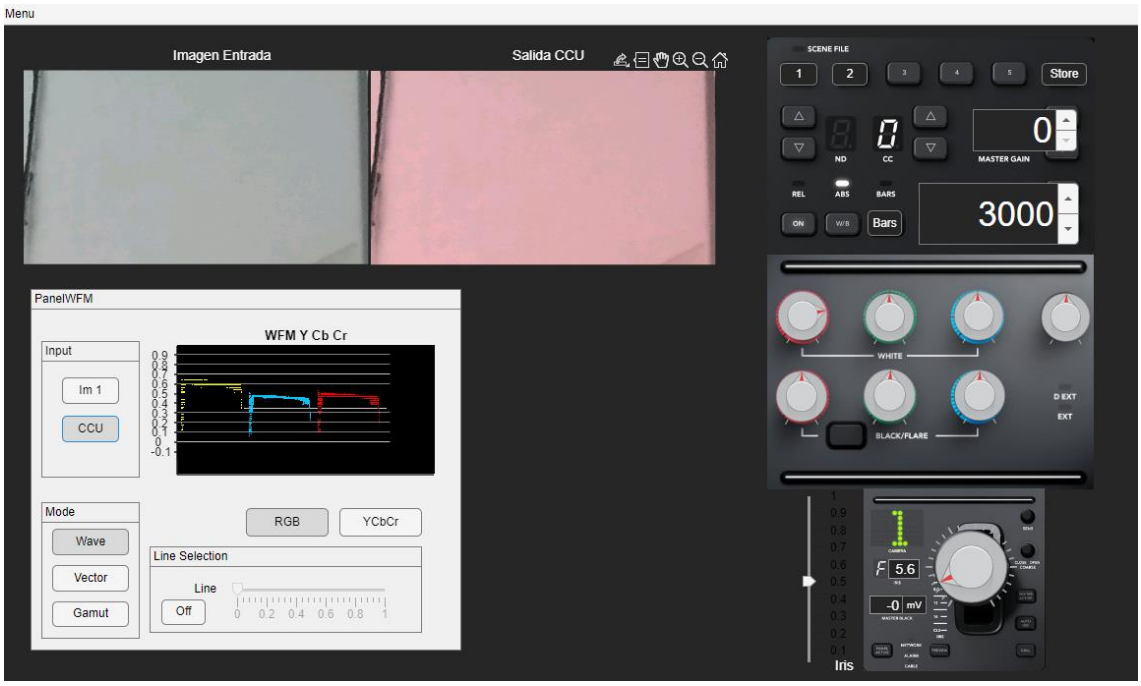

Figura 22: Interfaz de usuario de la aplicación al aplicar una ganancia de balance de blanco en la componente roja sobre la imagen "pared blanca"

Finalmente, se decide probar la función de barras de color de la aplicación. Se dispone a visualizar las componentes en el WFM en modo wave (figura 23) y gamut (figura 24).

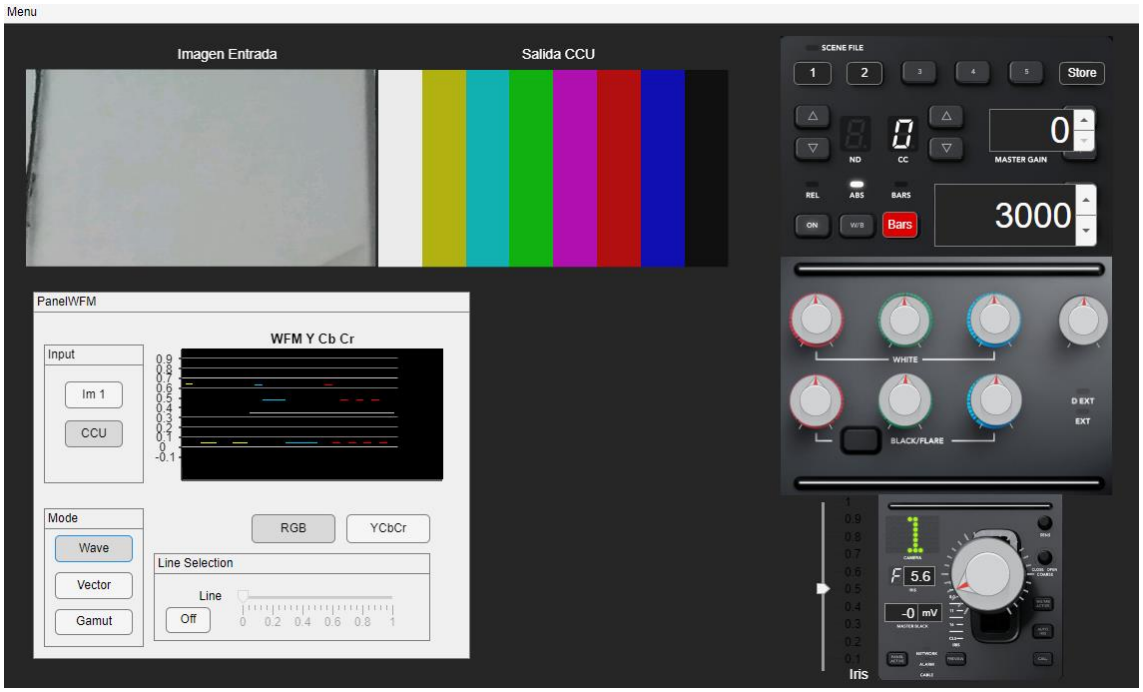

Figura 23: Interfaz de usuario de la aplicación al cargar barras de color y con el WFM en modo wave

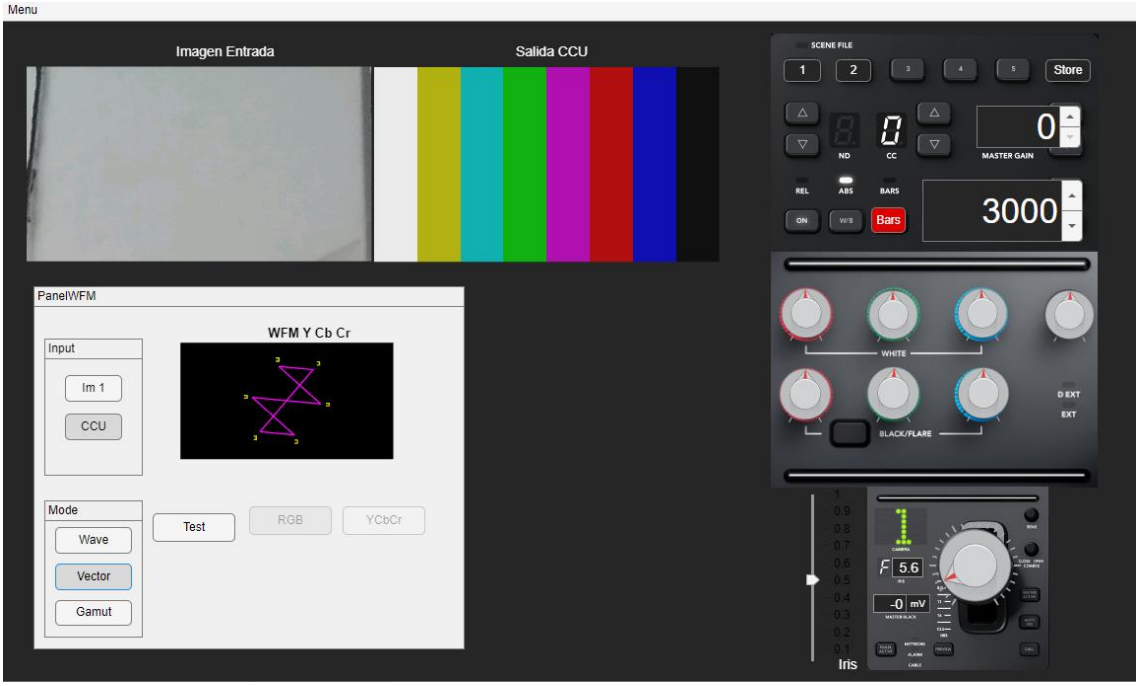

Figura 24: Interfaz de usuario de la aplicación al cargar barras de color y con el WFM en modo gamut

8. Conclusiones

En este apartado se realiza un balance general del proyecto, objetivos propuestos al inicio de este, objetivos realizados y resultados.

8.1 Conclusiones

Tras la realización de este proyecto, se puede afirmar que la renovación de la aplicación VirtualCCU 1.0 se ha realizado exitosamente. La interfaz de usuario realizada sigue una representación fiel de la ATEM BlackMagic que el alumno de la carrera se va a encontrar en el laboratorio. Así mismo, se han implementado todas las nuevas funcionalidades que incluye dicha CCU y que se propusieron al principio del proyecto, como el balance de blanco para componente verde, filtro de temperatura mejorado y con valores continuos, balance de negros, bancos de memoria, gestión de memoria, representación de componentes RGB en el WFM etc.

Además, se ha logrado también incorporar una nueva funcionalidad independiente a la CCU BlackMagic, como es la opción de cargar vídeos en la aplicación para su tratamiento digital. Esta última mejora convierte la aplicación VirtualCCU 2.0 no solo en una representación virtual de la herramienta digital del laboratorio, sino también en una herramienta útil para otros temas de la asignatura. Dicha aplicación ha sido probada siguiendo un guion de practica del laboratorio y los resultados han sido los esperados.

Por último, se ha logrado depurar y ordenar el código, reduciendo redundancias innecesarias y simplificando funciones, lo que ha logrado que la ejecución de dicho código se agilice.

En definitiva, todos los objetivos planteados al inicio del proyecto han sido realizados con éxito y la aplicación VirtualCCU 2.0 ya es una herramienta virtual lista para ser utilizada por los alumnos de la asignatura de “Ingeniería de vídeo”

8.2 Trabajos futuros

Pese haber cumplido los objetivos iniciales del proyecto, la aplicación VirtualCCU 2.0 sigue teniendo limitaciones que pueden ser fruto de un trabajo futuro. Entre dichas limitaciones se encuentra la carga de archivos de vídeo. Como se ha expuesto a lo largo del apartado 5, la carga de vídeos en la aplicación se ha logrado con una función que solo permite cargar un archivo de imágenes (fotogramas del video) previamente separados del vídeo original, sin llegar a lograr cargar vídeos de cualquier formato de archivo de vídeo. Ademas, una vez realizada esta mejora, se puede incorporar tambien la función de shutter, la cual solo se puede practicar sobre vídeo.

9. Referencias

[1] J. Chen, «Implementación de herramientas virtuales de aprendizaje para control de camara y mezcla/efecto de video.,» Madrid, 2021.

[2] [En línea]. Available: https://documents.blackmagicdesign.com/UserManuals/ATEM_Production_Studio_Switchers_Manual.pdf?_v=1663225210000.

[3] «Mathworks,» [En línea]. Available: https://es.mathworks.com/products/matlab/app-designer.html.

[4] «https://tannerhelland.com/2012/09/18/convert-temperature-rgb-algorithm-code.html,» [En línea].

[5] [En línea]. Available: https://web.archive.org/web/20131004221733/http://neutron.ing.ucv.ve/comunicaciones/Asignaturas/DifusionMultimedia/T-REC-H.262-200002-I!!PDF-S.pdf.

[6] «https://rgbcolorpicker.com/complementary,» [En línea].

[7] E. Rendón, Escritor, *reproduce.* [Performance].

Anexos

A.1 Presupuesto

Se procede a realizar un cálculo estimado del presupuesto de este proyecto. Para ello, se tiene en cuenta tanto los recursos tecnológicos utilizados durante la realización de dicho proyecto, como las horas de trabajo llevadas a cabo.
Para los recursos materiales se tienen en cuenta los materiales reflejados en la tabla 1. A pesar de que la licencia del software utilizado es gratis debido al convenio con la universidad, se refleja el precio de dicho software durante el tiempo del proyecto.

Tabla 1: Resumen de costes en recursos tecnológicos durante el proyecto

Recurso	Valor (€)	Tiempo de uso	Coste final (€)
Matlab R2023b	320	1 año	320
Signal Processing Toolbox	100	1 año	100
Image Processing Toolbox	100	1 año	100
COSTE TOTAL			520

Para realizar una estimación del coste de la mano de obra del proyecto, teniendo en cuenta que se han empleado 320 horas y que según la documentación consultada en la Comunidad de Madrid el salario de un ingeniero de telecomunicaciones es de 14,82€/hora, se estima un coste de 4742,4€.

Finalmente, se refleja en la Tabla 2 los costes totales del proyecto.

Tabla 2: Desglose de los costes totales del proyecto

Actividad	Coste
Recursos tecnológicos	520€
Mano de obra	4742,4€
COSTE TOTAL	**5262,4€**

A.2 Cronogramas

Fase	Objetivos Especificos	ID	Nombre de la Tarea	Descripcion de la tarea	Duracion (horas)	Fecha Inicio	Fecha Final
Fase 1	Estudio del proyecto anterior y comprension del codigo a mejorar.	1	Probar la aplicación existente	Se usará la aplicación ya existente para ver que funcionalidades presenta y cuales son sus limitaciones.	12	01/09/2023	08/09/2023
		2	Definir las mejoras a realizar	Una vez probada la aplicación, se definirán las funcionalidades a actualizar y posibles mejoras a incluir.	10	05/09/2023	13/09/2023
		3	Analisis y estudio de Matlab AppDesigner.	Se darán los primeros pasos en Matlab AppDesigner, descubriendo y entendiendo su estructura de programación y multiples funcionalidades.	18	13/09/2023	20/09/2023
		4	Comprensión y depuración del codigo existente.	Se estudiarán las lineas de codigo de la aplicación existente con el objetivo de saber que partes habrá que modificar, añadir o eliminar para la versión actualizada.	20	13/09/2023	23/09/2023
Fase 2	Actualizar funcionalidades ya existentes y pruebas.	5	Filtro de temperatura de color de rango amplio + pruebas.	Sobre el filtro de temperatura de color ya existente, se modificará el codigo para que el rango sea continuo en pasos de 50K	14	24/09/2023	26/09/2023
		6	Añadir boton de Green para el balance de blanco + pruebas.	Sobre el codigo ya existente para el balance de blanco con Red y Blue, se añadirá el boton Green.	16	27/09/2023	30/09/2023
		7	Añadir balance de negro con botones Red, Green y Blue + pruebas.	Se añadirá un panel exactamente igual que el de balance de blanco pero para realizar balance de negro.	20	01/10/2023	13/10/2023
		8	Añadir boton de ganancia para balance de blanco + pruebas.	En el panel de balance de blanco que ya habremos modificado, se añadira una rueda para ajustar la ganancia de los tres colores.	20	14/10/2023	25/10/2023
Fase 3	Creación de nuevas funcionalidades de la aplicación.	9	Actualizar la interfaz y el diseño de la CCU.	Se modificará el diseño de la CCU para que se parezca a las nuevas CCUs de BlackMagic de laboratorio.	20	26/10/2023	07/11/2023
		10	Incluir pequeños videos.	Se añadirá una nueva funcionalidad que permita incluir pequeños videos para su posterior tratamiento en la app.	35	08/11/2023	15/12/2023
		11	Incluir capturas desde webcam.	Se añadirá una nueva funcionalidad que permita incluir la señal en directo de una webcam y tratar capturas que se hagan desde ella.	35	16/12/2023	31/01/2024
Fase 4	Procesos documentales.	12	Ultimas pruebas, depuración del codigo y ultimos cambios en la interfaz	Se realizarán pruebas de las ultimas mejoras añadidas. Ademas se buscarán posibles optimizaciones de codigo para reducir el tiempo de procesado.	20	01/02/2024	10/02/2024
		13	Redacción de la memoria.	Se redactará la memoria del PFG. Este paso se hirá haciendo durante el resto de fases pero tom ará su forma final en la ultima fase.	45	01/02/2024	29/02/2024
		14	Preparacion de la defensa.	Se preparará la defensa del PFG ademas de realizar los ultimos tramites documentales.	35	01/03/2024	20/03/2023
					320		

A.3 Diagrama de Gantt

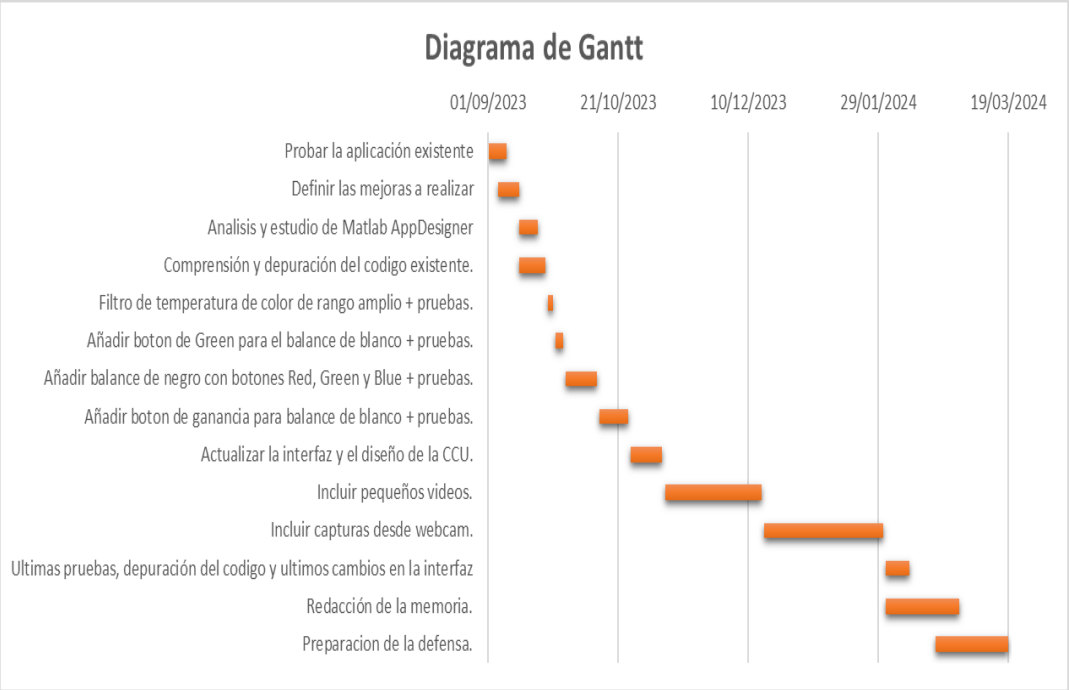

Imagen 26: Diagrama de Gantt del proyecto

A.4 Manual de usuario

La aplicación Virtual CCU 2.0 ofrece las siguientes funcionalidades:

- Load Image: Situado en la barra de acciones “Menu” en la parte superior izquierda, al pulsar sobre este botón, se abre un explorador de archivos para seleccionar la imagen que se quiere cargar. Acepta todos los formatos de archivos de imagen.
- Load Video: Situado en la barra de acciones “Menu” en la parte superior izquierda, al pulsar sobre este botón, se abre un explorador de archivos para seleccionar la carpeta que contiene las imágenes correspondientes a los fotogramas del video.
- Barras de color: Situado en la parte superior de la interfaz de la CCU, al pulsar sobre este botón se carga una imagen de barras de color al 75%.
- Memoria 1: Situado en la parte superior de la interfaz de la CCU. Tiene dos funciones: si primero se pulsa el botón STORE y después el botón de memoria 1, los ajustes actuales de balance de blanco y negro se guardan en el banco de memoria 1, si no se pulsa previamente el botón STORE y se pulsa solo el botón de memoria 1, la aplicación cargará los ajustes de balance de blanco y negro guardados previamente en el banco de memoria 1.
- Memoria 2: Situado en la parte superior de la interfaz de la CCU. Tiene dos funciones: si primero se pulsa el botón STORE y después el botón de memoria 2, los ajustes actuales de balance de blanco y negro se guardan en el banco de memoria 2, si no se pulsa previamente el botón STORE y se pulsa solo el botón de memoria 1, la aplicación cargará los ajustes de balance de blanco y negro guardados previamente en el banco de memoria 2.
- Filtro de temperatura: Situado en la parte superior derecha, encima del indicador “Shutter”. Dispone de unas flechas para subir o bajar la temperatura del filtro de color de temperatura. Se puede introducir directamente también el valor deseado.
- Ganancia general: Situado en la parte derecha superior de la interfaz de la CCU, justo encima del filtro de color de temperatura, dispone de unas flechas para subir o bajar el valor. Solo maneja 3 valores: 0, 6 y 12.
- Balance de blanco RGB: Son los botones giratorios que se encuentran en la primera fila de la parte central del interfaz de la CCU. El color que rodea dicho botón es el color de la componente correspondiente a dicho botón. Se puede regular hacia la derecha y hacia la izquierda para aumentar o disminuir la ganancia de la componente correspondiente. El cuarto botón giratorio contando desde la derecha corresponde al regulador de ganancia general de las 3 componentes para balance de blanco.
- Balance de negro RGB: Son los botones giratorios que se encuentran en la segunda fila de la parte central del interfaz de la CCU. El color que rodea dicho botón es el color de la componente correspondiente a dicho botón. Se puede regular hacia la derecha y hacia la izquierda para aumentar o disminuir la ganancia de la componente correspondiente.

- Apertura de Iris: corresponde a la barra deslizadora vertical situada en la parte inferior de la interfaz de la CCU. Regula la apertura de iris y se puede desplazar verticalmente para aumentar o disminuir dicha apertura.
- Pedestal: corresponde a la rueda situada en la parte inferior de la interfaz de la CCU y ajusta en nivel de pedestal en milivoltios. Se puede regular dicho valor girando la rueda hacia la izquierda o hacia la derecha.
- RGB: Situado en el panel de WFM, al pulsar este botón se mostrará por el WFM el valor las componentes RGB de la imagen cargada.
- YCbCr: Situado en el panel de WFM, al pulsar este botón se mostrará por el WFM el valor las componentes YCbCr de la imagen cargada.
- Wave: Situado en el panel de WFM, al pulsar este botón se mostrará por el WFM el valor las componentes de la imagen cargada.
- Vector: Situado en el panel de WFM, al pulsar este botón se mostrará por el WFM el valor las componentes en modo vector de la imagen cargada.
- Gamut: Situado en el panel de WFM, al pulsar este botón se mostrará por el WFM el valor las componentes en modo gamut de la imagen cargada.