ESCUELA DE INGENIERÍA DE
FUENLABRADA

GRADO EN INGENIERÍA EN SISTEMAS
AUDIOVISUALES Y MULTIMEDIA

TRABAJO FIN DE GRADO

DESARROLLO DE UN VECTORSCOPIO PARA DISPOSITIVOS MÓVILES ANDROID

Autor: Elena María del Río Galera

Tutor: David Gualda Gómez

Curso académico 2022 / 2023

Agradecimientos

Hay veces en la vida que las cosas ocurren de una manera que nunca llegamos a imaginar. Hoy es uno de esos días, donde la realidad supera la ficción y se cumple un sueño que durante mucho tiempo no sabía ni que tenía.

En primer lugar, quería dar las gracias a mi tutor, David, que me ha acompañado en este proyecto, me ha apoyado y animado a seguir a cada momento, especialmente cuando yo no creía que fuera capaz de sacar esto adelante. Gracias por enseñarme e inspirarme.

Gracias a todos y cada uno de los profesores que he tenido en cada asignatura y que han sido maestros. De todos he aprendido algo. Algunos de ellos me han enseñado con muchísima ilusión y otro me han retado y han conseguido enseñarme el valor de esforzarse cada día un poco más.

Especial agradecimiento a Obijuan, por ver algo en mí, cuando yo todavía no lo veía. Por enseñarme, y hacerme querer mejorar. Por ser mi maestro jedi, espero mantenerme siempre en el lado luminoso de la fuerza.

Gracias a todos los compañeros que han formado parte del viaje, porque con ellos los malos momentos han sido menos malos, y los buenos han sido aún mejores.

Mención especial a mi Taburete, que es lo mejor que me llevo de estos años. Gracias María, gracias Yolanda, por ser equipo siempre. Gracias por vuestra generosidad, vuestros ánimos, vuestra paciencia, pero sobre todo gracias por ser esas manos a las que agarrarse cuando todo se tambalea y esas amigas con las que celebrar, y bailar, aunque no sepamos.

Gracias a mi hermana y a mi madre, por animarme a estudiar esta carrera sabiendo que podía con ello, escuchándome en los días malos y dándome alas cuando quería rendirme. Gracias también a mi padre, por ser ejemplo de resiliencia. Y a toda mi familia, por ser apoyo, por estar unidos ante las dificultades, por ser hogar.

Gracias a la ingeniería, por apasionarme y enseñarme, por convertirse en mi profesión. El camino no ha sido nada fácil, pero eso solo ha significado que todo el esfuerzo ha merecido la pena. Lo difícil que ha sido llegar hasta aquí, es lo que lo hace grande. Y ahora… ¡A por la próxima aventura!

"I'm hoping to do some good in the world!"

Resumen

En este Trabajo Fin de Grado (TFG) se realiza el desarrollo completo de una aplicación *Android* que emula el comportamiento de un vectorscopio, el cual se utiliza para conocer la información relacionada con los colores que componen una imagen o un *frame* de video.

Previamente al desarrollo en *Android*, se ha llevado a cabo el desarrollo del vectorscopio en *Matlab*, con el objetivo de disponer de una referencia realizada mediante una herramienta de programación de alto nivel, con menor dificultad de desarrollo al hacer uso de funciones de alto nivel y poder comparar posteriormente los resultados con la versión de *Android.*

Aunque la aplicación está más orientada a profesionales de la edición de imagen y video, la sencillez de la interfaz desarrollada permite que cualquier usuario pueda manejar la aplicación, aunque la interpretación de lo que se observa requiera de ciertos conocimientos técnicos.

Por último, los resultados obtenidos en este proyecto han sido satisfactorios, ya que se han comparado los resultados del vectorscopio desarrollado en *Android* utilizando diversas imágenes con el vectorscopio realizado en *Matlab* y con la función de vectorscopio de una herramienta profesional de análisis y edición de video, produciendo resultados similares.

Palabras clave

Android, Matlab, aplicación, Vectorscopio, color, imagen.

Abstract

In this Final Degree Project, the full development of an Android application that emulates the behavior of a vectorscope has been realized, which is used to know the information related to the colors that compose an image or a video frame.

Prior to the Android development, the vectorscope has been developed in Matlab, in order to have a reference using a high-level programming tool, with less development difficulty by using high-level functions and to be able to later compare the results with the Android version.

Although the application is more oriented to image and video editing professionals, the simplicity of the developed interface allows any user to handle the application, although the interpretation of what is observed requires some technical knowledge.

Finally, the results obtained in this project have been satisfactory, since the results of the vectorscope developed in Android using different images have been compared with the vectorscope developed in Matlab and with the vectorscope function of a professional video analysis and editing tool, producing similar results.

Keywords

Android, Matlab, app, Vectorscope, color, image.

Índice de contenidos

Índice de figuras

Índice de tablas

Acrónimos

App	Application
API	Application Programming Interface
APK	Android Application Package
ARIB	Association of Radio Industries and Businesses
CMYK	Cyan, Magenta, Yellow, Key
DTD	Definición de Tipo de Documento
GPS	Global Positioning System
HTML	Hypertext Markup Language
IDE	Integrated Development Environment
iOS	iPhone Operating System
IU	Interfaz de Usuario
IRE	Institute Radio Engineers
JVM	Java Virtual Machine
Lab	Lightness, green / magenta, blue / yellow
Linux	Lovable Intellect Not Using XP
MathML	Mathematical Markup Language
MATLAB	Matrix Laboratory
NTSC	National Television System Committee
PAL	Phase Alternating Line
Pixel	Picture Element
PLUGE	Picture Lineup Generating Equipment
RGB	Red, Green Blue
RMA	Radio Manufacturers Association
SECAM	Séquentiel Couleur à Mémoire
SMPTE	Society of Motion Picture and Television Engineers
SVG	Scalable Vectors Graphics
UER	Unión Europea de Radiodifusión
URI	Uniform Resource Identifier
UX	User Experience
XML	Extensible Markup Language
XHTML	Extensible HyperText Markup Language
W3C	World Wide Web Consortium
WORA	Write once, run anywhere
YIQ	Luminance; In-Phase; Quadrature

1. Introducción

En este primer capítulo se incluye una presentación del proyecto, se abordan las motivaciones que han llevado a elegir este tema y se explica la estructura de la memoria, desglosando los capítulos.

1.1. Presentación

A lo largo de esta memoria se describe en detalle el proyecto realizado para el Trabajo de Fin de Grado (TFG) de Ingeniería en Sistemas Audiovisuales y Multimedia. El proyecto consiste en la creación de una aplicación *Android* que emula el comportamiento de un vectorscopio, utilizado para observar y medir la información de color de imágenes. Además, se presenta en una interfaz sencilla para poder estudiar cualquier tipo de imagen de la galería del dispositivo móvil. Para poder llegar a tener éxito creando la aplicación, en un primer momento se ha realizado una aplicación piloto en *Matlab* (*Matrix Laboratory*). La capacidad computacional de esta herramienta a la hora de procesar imágenes fue la razón principal de empezar el proyecto por ahí. Tras disponer de un programa funcional y atractivo en *Matlab* y entender el código, con *Android Studio*, se ha realizado la segunda parte y gran objetivo de este proyecto.

Durante el desarrollo de la aplicación se ha llevado a cabo un trabajo continuo de investigación para poder profundizar en los campos relacionados con el proyecto. Toda esta investigación se refleja en los siguientes capítulos para ayudar a la comprensión final en conjunto del proyecto.

1.2. Motivación personal

Hoy en día es extraño que alguien no disponga de un *smartphone* y lo lleve consigo en todo momento. Esta tecnología ha cambiado nuestras vidas y la forma en que nos relacionamos.

Los dispositivos móviles contienen aplicaciones que facilitan la vida diaria y la conectividad con nuestros seres queridos. En pocos segundos se puede contactar con una persona al otro lado del mundo mediante una videollamada, realizar cálculos complejos o consultar el estado del tráfico en tiempo real.

Este proyecto nace de la inquietud de querer conocer cómo funcionan las aplicaciones por detrás y cómo se puede llegar a desarrollar una, cuáles son los lenguajes de programación que se usan, cómo se organiza la parte visual…

La elección de programar en *Android* y no en *iPhone Operating System* (*iOS*), es en primer lugar porque *Android* es el sistema más utilizado a nivel mundial**,** según las estadísticas, la cuota de mercado de *Android* e *iOS* es del 72,2 % y el 26,99 %, respectivamente.

La segunda razón es porque antes de sumergirse en este proyecto, ya se tenían conocimientos de *Swift*, que es el lenguaje creado por *Apple* enfocado para el desarrollo de aplicaciones en *iOS* (tanto para móviles como para ordenadores.), y por el contrario no se tenía ningún conocimiento de las herramientas que se utilizan a nivel técnico para desarrollar aplicaciones *Android.*

En cuanto a la decisión de desarrollar un vectorscopio en vez de cualquier otra aplicación, existen varias razones:

1. La primera es que, en las búsquedas realizadas en la *Play Store*, se han encontrado muchas aplicaciones de edición de fotografía, pero no se ha encontrado ninguna aplicación que sea o contenga un vectorscopio. Es por eso por lo que el reto era mayor.
2. La segunda razón de desarrollar un vectorscopio, es que en su día en la asignatura de Equipos de Audio y Vídeo se usaron dispositivos de este tipo, pero no se profundizó, por lo que surgió cierta inquietud en conocer cómo funcionan estos sistemas a bajo nivel.
3. La tercera razón fue poder profundizar en temas de tratamiento de imágenes, espacios de color, y otros conceptos relacionados con el tema audiovisual para ver cómo se aplican en el día a día y por qué son importantes.

Con todas estas variables presentes, se procede a realizar este proyecto.

1.3. Estructura de la memoria

En esta sección se describen cada uno de los capítulos para facilitar su orden y comprensión.

1. **Introducción**: a través de varios apartados se presenta el proyecto, así como la motivación personal que ha llevado a desarrollar este proyecto con las herramientas que se ha realizado y como se desarrollará estructuralmente la memoria.

2. **Objetivos**: se concretan los objetivos que se pretenden conseguir, se detallan las fases en las que se ha estructurado el proyecto, y se muestra un diagrama de Gantt en el que consultar el tiempo aproximado que se ha dedicado a cada fase.
3. **Definiciones y conceptos:** este apartado trata de describir las herramientas utilizadas para el proyecto, así como de explicar conceptos que en capítulos posteriores se van a usar y pueden ser desconocidos, tales como: espacios de color, la importancia del vectorscopio en la industria, el uso de los lenguajes de programación *Java* y *Matlab,* etc.
4. **Desarrollo del trabajo:** este capítulo se divide en dos apartados: la implementación realizada en *Matlab* y en *Android*. En cada capítulo se explica el código utilizado para lograr los resultados, utilizando como apoyo los flujogramas de las correspondientes aplicaciones y/o funciones específicas.
5. **Resultados:** se han elegido varias imágenes y han sido probadas en *Matlab, Android* y una herramienta profesional (*Adobe After Effects*). Además, en este capítulo se muestran y comentan los resultados obtenidos.
6. **Costes:** se presenta el coste aproximado de la realización de la aplicación.
7. **Conclusiones:** se recogen las conclusiones obtenidas del proyecto, las competencias empleadas, competencias adquiridas y por último se sugieren algunas líneas de mejora para la aplicación en un futuro.
8. **Bibliografía:** se referencian las fuentes consultadas.

2. Objetivos

En este capítulo se detallan los objetivos, las fases del proyecto, y el diagrama de Gantt para mostrar la organización temporal de las fases que componen el TFG.

2.1. Objetivos del proyecto

Los objetivos concretos de este proyecto son:

- Recopilación de la información sobre el funcionamiento del vectorscopio y los conceptos relacionados con el mismo.
- Desarrollo de un vectorscopio en *Matlab* con el objetivo de disponer de una referencia sencilla de implementar y poder comparar los resultados con el desarrollo en *Android*.
- Desarrollo del vectorscopio en *Android* y comparativa de resultados con la versión de *Matlab* y con una herramienta de edición y análisis de imagen y video profesional.

2.2. Fases de desarrollo del proyecto

2.2.1. Fase de investigación

Esta fase ha sido constante y recurrente a lo largo del desarrollo del proyecto. Se centra en la obtención de la información y conceptos relacionados con el funcionamiento interno del vectorscopio, las funciones de *Matlab* para poder llevar a cabo la primera versión, así como el manejo de *Android* y las funciones necesarias para realizar el desarrollo del proyecto.

2.2.2. Fase de desarrollo

Esta fase se subdivide en desarrollo en Matlab y en Android.

- **Desarrollo en Matlab**

Para facilitar el trabajo posterior con *Android*, y debido a la facilidad que presenta *Matlab* para el cálculo computacional, se comienza a desarrollar un vectorscopio en *Matlab*. De esta forma, se procesan de forma más simple las matrices asociadas a las imágenes que se pretenden analizar y servirá como referencia comparativa cuando se realice el desarrollo en *Android.*

Una vez terminada esta fase, siendo los resultados en *Matlab* satisfactorios se comienza la segunda fase de desarrollo.

- **Desarrollo de la aplicación en Android**.

Partiendo de la lógica desarrollada en *Matlab*, en esta fase se realiza el desarrollo completo del vectorscopio en *Android*, comenzando con la interfaz gráfica y realizando posteriormente la algoritmia relacionada con la obtención de la información de color de la imagen deseada.

2.2.3. Fase de evaluación de resultados

En esta fase, se utilizan diversas imágenes para comprobar el funcionamiento correcto de la aplicación desarrollada en *Android*. Para validar los resultados, con cada imagen se comparan los resultados obtenidos en *Android* con los que se obtienen en *Matlab* y con los obtenidos con la herramienta de edición y análisis profesional *Adobe After Effects*.

2.2.4. Fase de documentación

Esta fase de documentación ha sido llevada a cabo en diferentes tramos del proyecto:

- al comienzo, con la estructura de la memoria y la definición de los objetivos.
- En el tramo del desarrollo del vectorscopio en *Matlab* y *Android*, documentando los algoritmos realizados de cada una de las funciones implementadas.
- En la parte final del proyecto, documentando los resultados obtenidos, las conclusiones y líneas futuras, así como el resto de los apartados.

2.3. Diagrama de Gantt

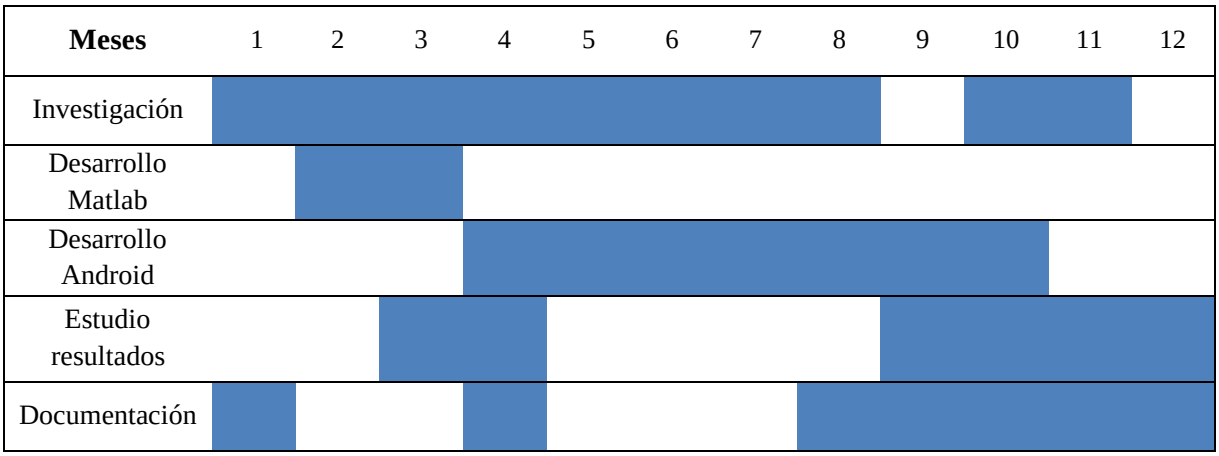

Tabla 1. Diagrama de Gantt.

3. Definiciones y conceptos

En este capítulo se describen las definiciones y conceptos que se han utilizado para el desarrollo de este trabajo.

3.1. Píxeles

Las imágenes digitales están compuestas por píxeles (*Picture Element*), los elementos más pequeños que forman parte de una imagen para su representación digital. Un píxel, se divide en subpíxeles, rojo, verde y azul, RGB (*Red, Green, Blue)*, estos subpíxeles se distribuyen en distintas proporciones para generar un color u otro. Los subpíxeles se suman entre ellos (mezcla aditiva) y se obtienen como resultados diferentes valores relativos a un color concreto. Por ejemplo, si los tres colores primarios presentan el valor máximo de 255, la salida es en blanco, en caso de que el valor sea el mínimo toman el color negro, si uno de los subpíxeles está en su máximo valor y los otros a cero, R = 255, G = 0, B = 0, se obtiene un tono puro, en este ejemplo el rojo. [1].

En la Figura 1, se puede ver como se forma un píxel de color anaranjado, los subpíxeles toman cada uno un calor (R=255, G=175, B=97) y estos valores se suman, para formar el color de la imagen.

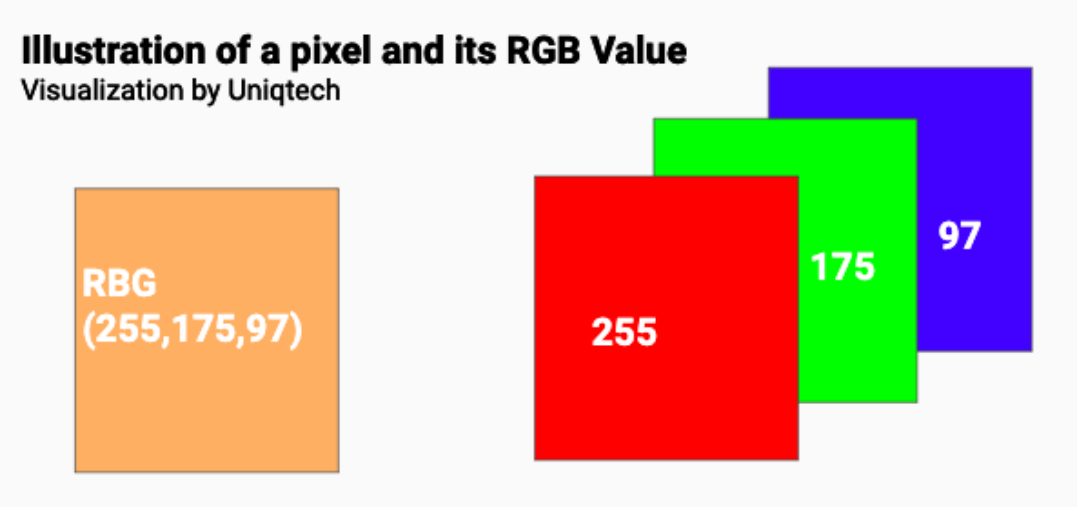

Figura 1. Distribución de los valores RGB de un píxel [2].

Además, cada píxel tiene información de luma y croma (como refleja la Figura 2):

1. **luma**, la parte de la luz de una imagen, los tonos grises.
2. **Croma**, aporta el color a la imagen.

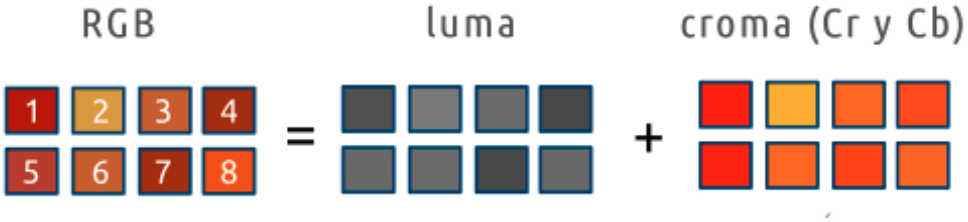

Figura 2. Luma y croma en píxeles [3].

3.2. Espacios de color

Los diferentes espacios de color influyen en el rendimiento de la imagen. Los sistemas de interpretación del color son representaciones matemáticas abstractas que permiten ordenar y asignar valores numéricos a los colores. Esto ayuda a interpretar y representar los colores de manera precisa y coherente.

Hay diversos espacios de color, RGB, CMYK (*Cyan, Magenta, Yellow, Key*), Lab (*Lightness, green / magenta, blue / yellow),* YIQ *(Luminance; In-Phase; Quadrature), National Television System Committee* (NTSC)…

RGB

El espacio de color RGB, es un espacio de color aditivo, es decir se obtienen los valores a partir de la suma de los componentes de color. Cada color se crea mediante la emisión de diferentes intensidades de luz roja, verde y azul

Este modelo de color es frecuentemente utilizado en gráficos por ordenador debido a su similitud con el sistema visual humano. Para cada uno de los tres colores, rojo, verde y azul, se almacenan valores individuales de 16 bits.

Figura 3. Esquema de color aditivo [4].

YIQ

El espacio de color YIQ, o NTSC, es el que se usa en los sistemas de televisión en color que utilizan el sistema de codificación NTSC. Este sistema es utilizado en países como Japón y está bastante extendido por la mayoría de los países de América del norte.

Si se analizan las iniciales de YIQ, se puede comprobar que la Y, se refiere al valor de la luminancia (el componente que se utiliza en las emisiones de televisión en blanco y negro), mientras que la I y la Q, se refieren a la fase y a la cuadratura respectivamente y son los valores que representan la crominancia.

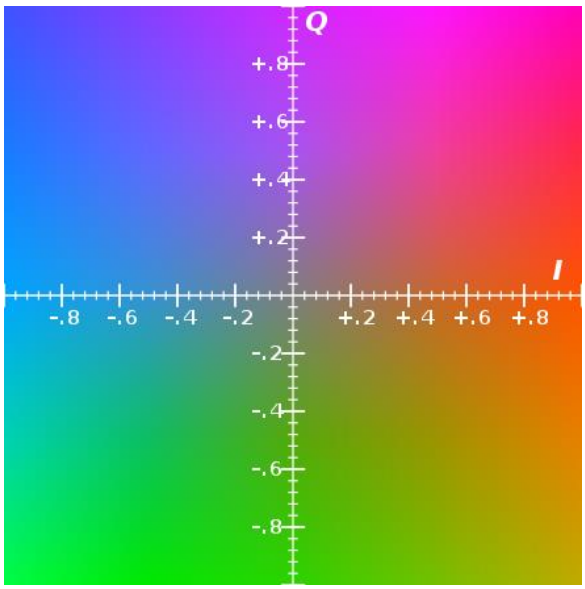

Figura 4. Espacio de color YIQ en Y = 0.5 [5].

El modelo de color YIQ está diseñado para explotar las propiedades perceptuales de la visión humana. El ojo es más sensible a los cambios en el rango naranja-azul (I) que en el rango púrpura-verde (Q); por lo tanto, se requiere menos ancho de banda para Q que para I [6].

3.3. Barras de color

En la producción de televisión de señales de vídeo es habitual utilizar la mira o carta de barras de color para calibrar los sistemas de producción y realizar los ajustes oportunos si fuera necesario.

En ámbitos profesionales ante una señal de prueba preliminar, con las barras de color se puede calibrar la señal para que sea correcta. Cuando la señal llega a los usuarios finales lo que

están enviando coincide exactamente con la señal que se recibe. El propósito de las barras es preservar la integridad de la señal.

Las barras de color representan los colores clave, gris, amarillo, cian, verde, magenta, rojo y azul. En trabajos profesionales es habitual incluir un minuto de metraje de barras para facilitar el trabajo a los técnicos en postproducción.

A continuación, se van a comentar brevemente los tres estándares de barras más utilizados mundialmente.

SMPTE

Las barras de color *Society of Motion Picture and Television Engineers* (SMPTE) son un patrón de prueba de televisión de marca registrada que se utiliza donde se utiliza el estándar de video NTSC. Se utiliza para configurar monitores y ajustarla crominancia NTSC y la información de luminancia correctamente.

Como se pude ver en la Figura 5, la imagen de barras SMPTE representa en siete barras verticales proporcionales los colores primarios y complementarios, así como el gris en los dos tercios superiores. En la parte restante se representan los tonos blancos y negros así como el púrpura.

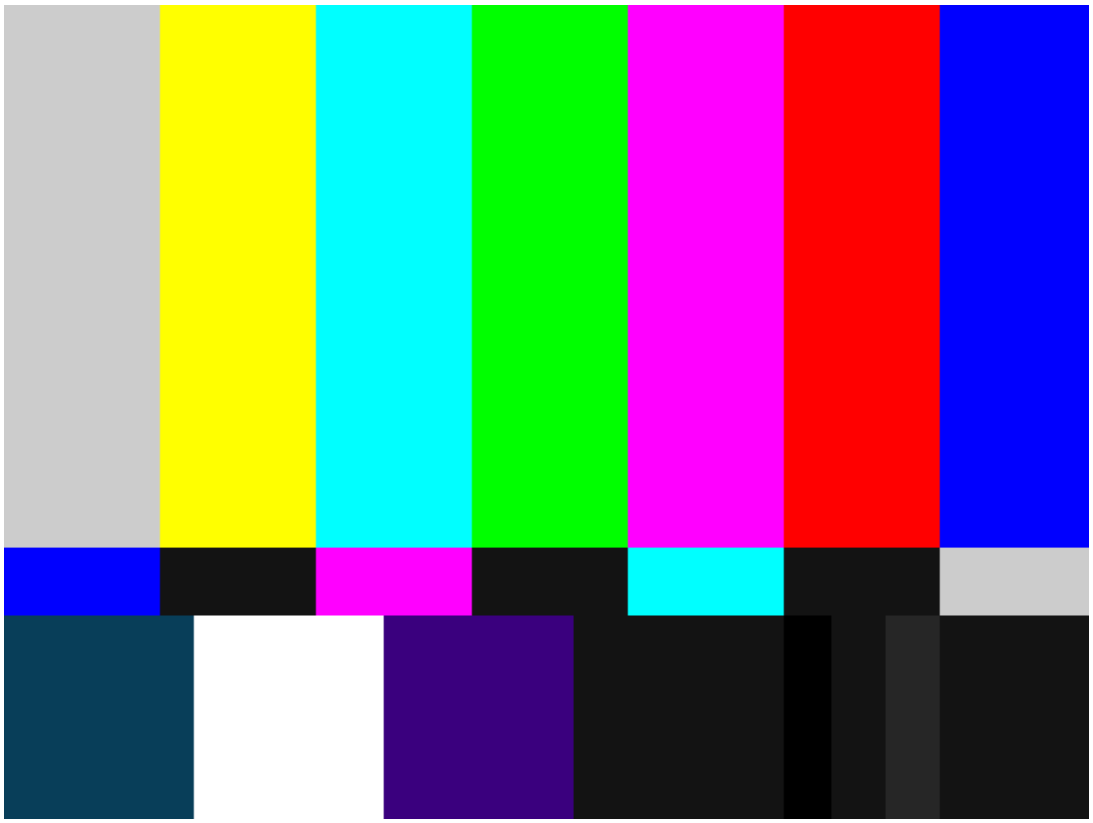

Figura 5. Imagen de barras SMPTE [7].

UER

La Unión Europea de Radiodifusión (UER) creó una carta de color que ordena de manera decreciente las barras de color con relación al nivel de luminancia, es decir, comienza en el blanco y termina en el negro, estos dos colores carecen de información de color.

La carta de barras que se genera es fácilmente identificable con una escala de grises, y es utilizada para calibrar sistemas de vídeos que utilizan el sistema de codificación *Phase Alternating Line* (PAL).

En la Tabla 2, se encuentran representados los niveles de los componentes de la señal de las barras de color.

Color	Luminancia Y	R-Y	B-Y	S	Y+S	Y-S
Blanco	1	0	0	0	1	1
Amarillo	0,89	0,11	-0,89	0,9	1,79	-0,01
Cian	0,70	-0,70	0,30	0,76	1,46	-1,06
Verde	0,59	-0,59	-0,59	0,83	1,41	-0,24
Magenta	0,41	0,59	0,59	0,83	1,24	-0,42
Rojo	0,30	0,70	-0,30	0,76	1,06	-0,46
Azul	0,11	-0,11	0,89	0,90	1,01	-0,79
Negro	0	0	0	0	0	0

Tabla 2. Nivel mira UER [8].

Los valores indicados en la tabla se aplican en el caso en que la crominancia está al 100% de saturación al igual que la luminancia.

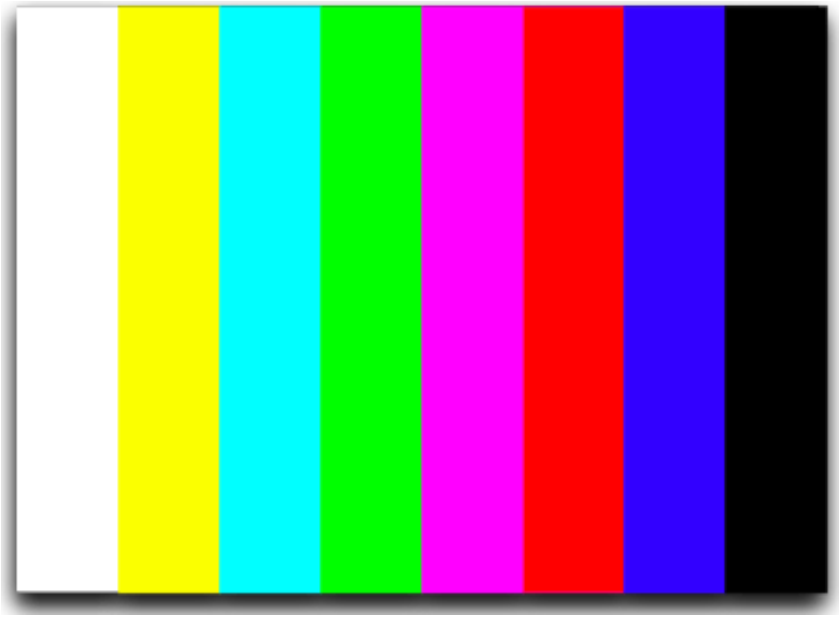

Figura 6. Barras de color UER [9].

ARIB

En las barras de color *Association of Radio Industries and Businesses* (ARIB), la señal de prueba presenta una mayor variedad de componentes en comparación a la señal SMPTE. Contiene las mismas barras de color de contraste al 75%, con la adición de áreas de grises neutros con diferentes niveles de luminancia, y un *Picture Lineup Generating Equipment (Pluge)* con más pasos (-2%, 2% y 4%) que supone un mayor nivel de precisión para el establecimiento de los niveles de negro y brillo [10].

Figura 7. Barras de color ARIB [11].

3.4. Sistemas de codificación

Al surgir la televisión en color, se crean dos sistemas de codificación y transmisión de señal. Estos sistemas se aplican tanto en vídeo analógico como en vídeo digital:

- **NTSC:** en países como América del Norte y Central, en la mayor parte de América del Sur, y en Japón la norma de codificación que se utiliza es el sistema de codificación NTSC, Comité de Sistema de Televisión Nacional. Debido a la presencia de la televisión surge la necesidad de crear estándares para las emisiones comerciales, por eso la *Radio Manufacturers Association* (RMA), estadounidense crea el NTSC, que a su vez crea unos estándares para las emisiones comerciales de televisión.

En este sistema de codificación 525 líneas entrelazadas componen las imágenes, de las cuales 486 componen el cuadro visible, y se muestran en una tasa de 29,97 *frames* por segundo.

- **PAL**, abreviatura de *Phase Alternate Line*. En la mayoría de los países europeos se utiliza este estándar. Una imagen PAL se compone de 625 líneas entrelazadas de las cuales 576 componen el cuadro visible, y muestra las imágenes a 25 *frames* por segundo [12].
- **SECAM** abreviatura de *Sequential Color and Memory*, de la misma forma que PAL una imagen SECAM tiene 625 líneas entrelazadas que se muestran a 25 *frames* por segundo. Sin embargo, este sistema de codificación, procesa la información de color de una forma incompatible con PAL [13].

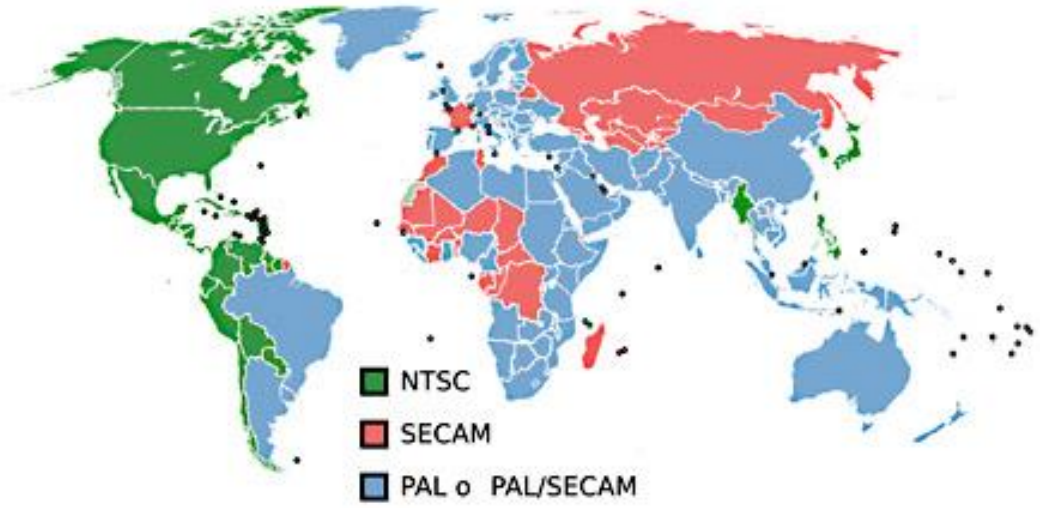

Figura 8. Mapa de distribución de Sistemas de codificación utilizados mundialmente [14].

3.5. Vectorscopio

El vectorscopio es un osciloscopio, que se basa en la representación de la información de color de la señal de vídeo. Se le conoce también como *Video Scope* o Monitor de Campos de Video. No se puede hablar profesionalmente de corrección de color sin concebir este aparato. En el monitor encontramos puntos de referencia de los colores primarios y complementarios dispuestos en un círculo. Según la distribución de los valores se interpretan cualidades como el tono y la saturación.

El vectorscopio tiene referencias para los niveles de saturación «legales» para difusión. Los valores legales varían según el país de emisión y están relacionados con los sistemas de codificación de vídeo. Es por eso por lo que un vectorscopio que se utilice para medir el color

de sistemas NTSC tiene los valores de referencia en posiciones distintas al vectorscopio para sistemas PAL. En adelante en este trabajo se centra el foco en los vectorscopios para sistemas NTSC.

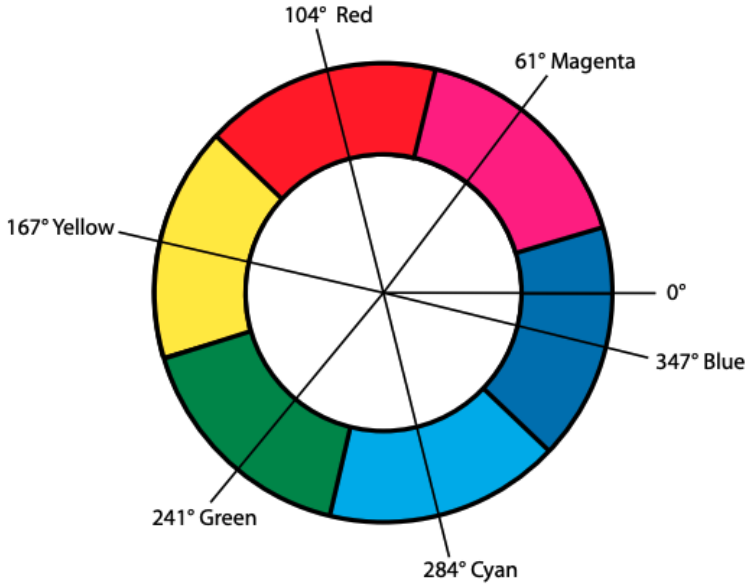

Figura 9. Rueda de colores en la que se basa el vectorscopio [15].

Aunque en este proyecto la investigación y el desarrollo se centran en vectorscopios de vídeo, es interesante conocer que también existen los vectorscopios de audio, los cuales permiten diferenciar entre canales de señales de audio estéreo.

Mientras que el vectorscopio analiza la información de color, el monitor de onda representa gráficamente la información de luminancia, es decir la luz de una imagen.

3.5.1. Monitor de forma de Onda

El monitor de forma de onda muestra la luminancia y la crominancia de una imagen, en términos muy sencillos; muestra el brillo y el contraste de la imagen.

La escala en esta herramienta es de 0 a 100, va de Negro/Sombras (0) a Blanco/Altas luces (100). El nivel inferior representa la zona de sombras, el nivel medio los tonos medios y el nivel superior las luces. Si se pasa de 0 o 100, se producirá un recorte y la imagen no mostrará todos sus detalles e información.

En la Figura 10, el círculo verde representa las cortinas de la izquierda, se puede ver en la forma de onda todas las "ondas" creadas por el pliegue de las cortinas y la luz diferente que rebota en ellas. Luego se tiene toda la información general dentro del gráfico, las zonas oscuras y las claras, se puede discernir la barandilla blanca (círculo azul) dentro de las zonas claras. En

la parte superior, el círculo rojo muestra todas las zonas sobreexpuestas en las ventanas de la derecha, luces fundidas que no son recuperables [16].

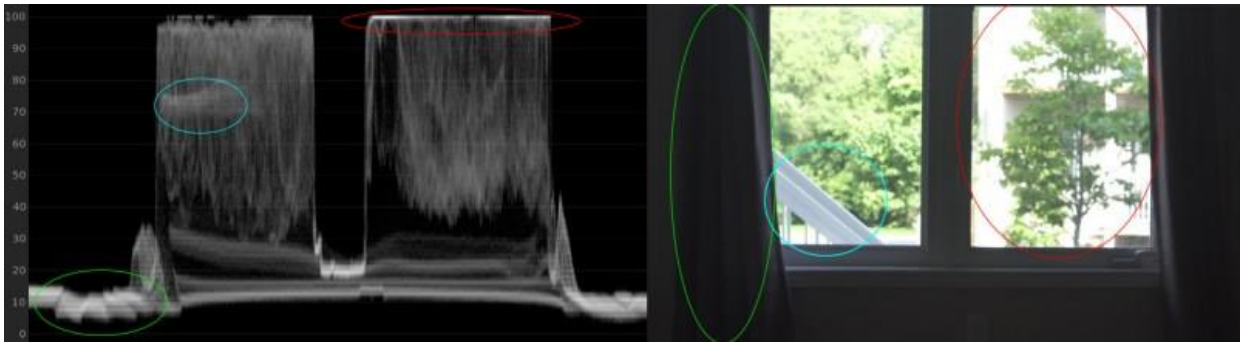

Figura 10. Monitor de forma de onda. (a) Representación. (b) Imagen de estudio [17].

3.5.2. Interpretación del vectorscopio

Los valores que representa el vectorscopio se leen o interpretan de la siguiente manera: [R] Rojo, [Mg], Magenta, [B] Azul, [Cy] Cian, [G] verde y [Yl] amarillo y se corresponden con los valores de los tonos puros.

La forma de rueda del vectorscopio ayuda a medir distancias. La distancia a la que se encuentra un punto representativo de la imagen de estudio respecto del centro del vectorscopio se corresponde con la saturación de dicho píxel en la imagen. Mientras que, si se dibuja una línea desde el centro hasta dicho punto representativo, el ángulo que forma esa línea se refiere al tono o matiz.

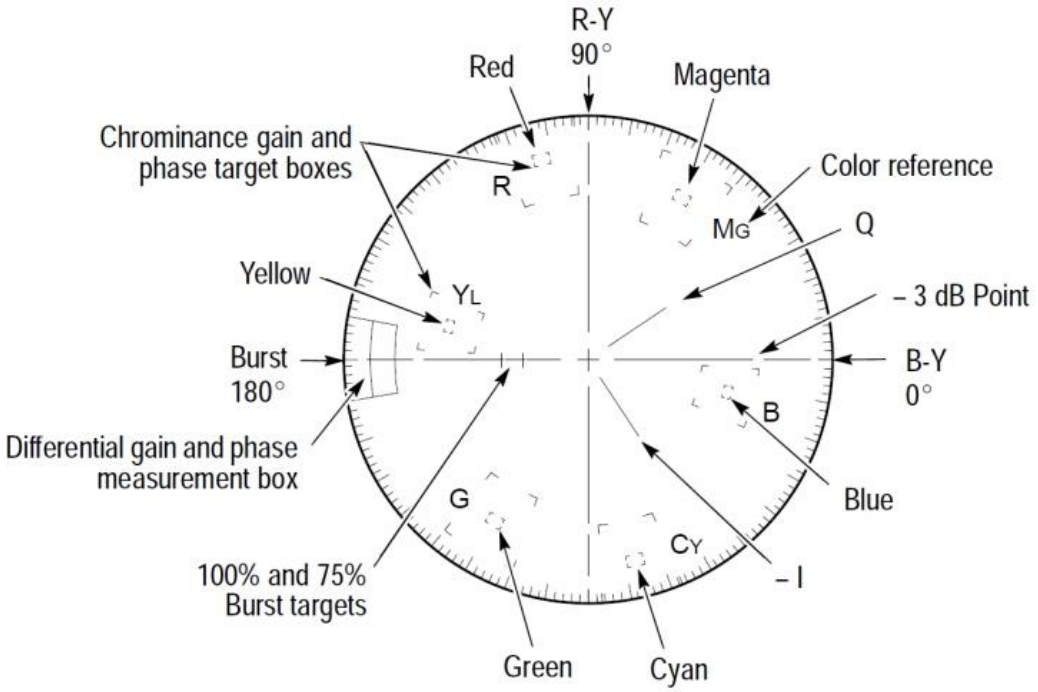

Figura 11. Explicación del Vectorscopio NTSC [18].

Un vectorscopio muestra gráficamente la crominancia de una imagen. En el campo de vídeo, el vectorscopio muestra 525 líneas (un fotograma de vídeo) 30 veces cada segundo.

La rejilla representa todos los colores dentro de un círculo, el gris neutro se sitúa en el centro. Para un punto de color determinado, la saturación se representa por su distancia al centro del círculo, mientras que el ángulo alrededor del centro representa el tono del color. Por esta razón, la diferencia de ganancia entre un color y su cuadro de destino indica un fallo de saturación y un ángulo de fase diferente insinúa un tono erróneo [19].

Si nos preguntamos cuáles son los valores de un color correcto es algo subjetivo, ya que es una decisión de la dirección de fotografía de la producción audiovisual. Hay producciones en las que los colores se ven más verdes, en otras más amarillos…. Por ejemplo, si se habla de la serie de televisión *CSI Miami o Hawái 5.0*, es claro que estas producciones buscan tonalidades más expresivas.

3.5.3. Usos del vectorscopio

Algunos de los usos más importantes del vectorscopio se van a comentar a continuación.

- **Corrección de los tonos de piel**

Una de las mejores características del vectorscopio, es la línea del tono de la piel, se encuentra a medio camino entre el amarillo y el rojo. Esta línea está diseñada para representar el flujo sanguíneo que atraviesa la piel humana y se asegura de que la piel de las personas se observe correctamente en la cámara [20].

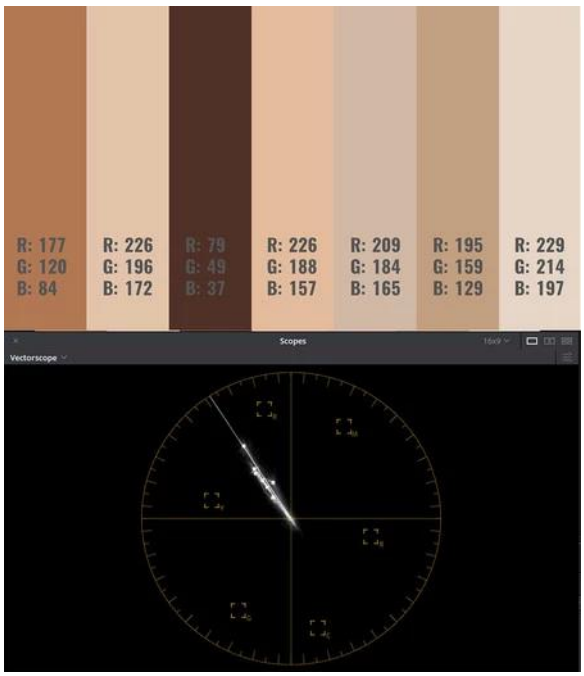

Figura 12. Representación de barras de tonos de piel en *Davinci Resolve* [21].

- **Balance de blancos**

En el ámbito fotográfico y de procesado de imágenes electrónicas, el "balance de blancos", "equilibrio de color" o "equilibrio de blancos" es un arreglo realizado por software que consigue una generación correcta de color. De esta manera se evita la predominancia de unos colores respecto a otros, que se hace especialmente sensible cuando se trata de tonos como el blanco o los grises.

El correcto balance de blancos de una videocámara se indica mediante el punto centrado en la pantalla del vectorscopio cuando el objetivo o la señal que se visualiza es un objeto blanco o gris neutro. Si el balance de blancos de la cámara está desactivado el punto se desplazará fuera del centro.

- **Ajuste de la pantalla verde**

Una pantalla verde es un fondo verde que se utiliza en producción de vídeos para generar efectos visuales en la etapa de postproducción.

Las pantallas verdes deben ser verdes, por supuesto, pero también es importante que los tonos de piel y el resto de los colores presentes en la toma sean correctos. Se utiliza el vectorscopio para asegurar que la pantalla verde está bien iluminada, tiene la exposición adecuada y de que los demás colores de la toma tienen un aspecto natural. La línea del vectorscopio debe ir en diagonal desde el centro entre el verde y el rojo.

3.6. Histograma

En el procesamiento digital de imágenes, un histograma es una representación estadística de los datos de una imagen. Normalmente se representa como un gráfico de barras que ilustra cada nivel de intensidad en el eje x y su frecuencia de aparición en la imagen de estudio en el eje y. Para crear un histograma, se calcula la cantidad total de píxeles de cada nivel de intensidad de la imagen. Como una imagen RGB es multicanal, se suele pasar la imagen a escala de grises para no tener que representar cada canal por separado.

Los histogramas no contienen ninguna información espacial sobre la imagen.

Están divididos en secciones: a la izquierda están las sombras, en el centro los tonos medios y a la derecha las altas luces. La información de la imagen se muestra dentro de ese gráfico. El borde de la izquierda es el negro verdadero y el de la derecha es el blanco puro. Si se superan los bordes, las sombras o altas luces estarán recortadas, lo que significa que no se grabará toda la información y se perderán los matices sin posibilidad de recuperar esa información posteriormente. Una imagen correctamente expuesta dicta que se necesita

mantener el gráfico o la información capturada dentro de esos límites, cuanto más cerca se está de los bordes más oscuro o claro será el color y la imagen capturada [22].

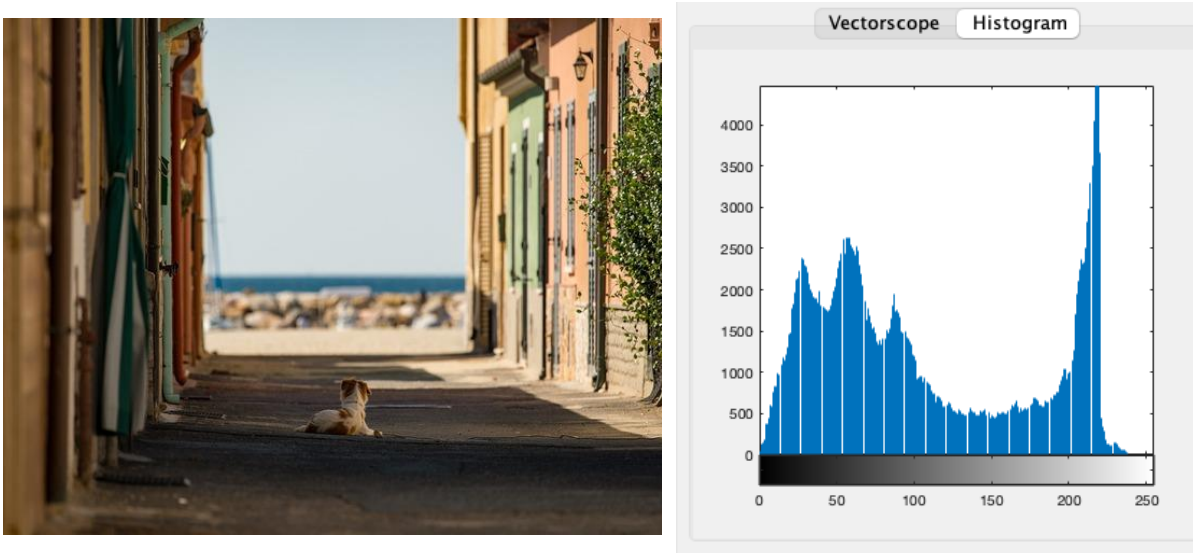

Figura 13. Representación del histograma. (a) Imagen original. (b) Histograma.

Observemos la imagen de la Figura 13 y leamos el histograma. De izquierda a derecha, en las sombras, las sombras proyectadas por el edificio en la calle. Para los tonos medios, en el centro, todo lo que se puede ver claramente desde las sombras claras. A continuación, tenemos las altas luces, que es la información de recorte, esta parte de la imagen está sobreexpuesta, y, no se puede salvar, es el pico más alto.

3.7. MATLAB

MATLAB (*MATrix LABoratory*) es un entorno de programación utilizado en su mayoría por matemáticos, ingenieros y científicos. El lenguaje de programación que se utiliza también se llama *Matlab,* este lenguaje está basado en vectores y matrices permitiendo realizar cálculos técnicos y complejos con aparente sencillez.

Al abrir *Matlab*, se encuentra una ventana de trabajo como la que se puede ver en la Figura 14.

En la parte superior de la ventana de trabajo de *Matlab,* se encuentra la Barra de menús. Desde donde se pueden crear nuevos archivos, guardar, ejecutar…

La parte inferior se divide en varios espacios, algunos relevantes son:

- ***current Folder,* directorio de trabajo**. Se encuentra a la izquierda del todo. Es el directorio en el que se leen y se guardan todas las funciones y programas que se van a crear durante la sesión de trabajo. Se pueden consultar los archivos que hay en el

directorio abierto, cambiar a otros directorios o cargar el contenido de un archivo pinchando dos veces sobre su el archivo.

- **Ventana de Edición de archivos.** Aquí se abren los nuevos archivos para poder escribir y modificar el código. Este código se guarda en archivos con extensión m. Se encuentra al lado de *Current Folder*.

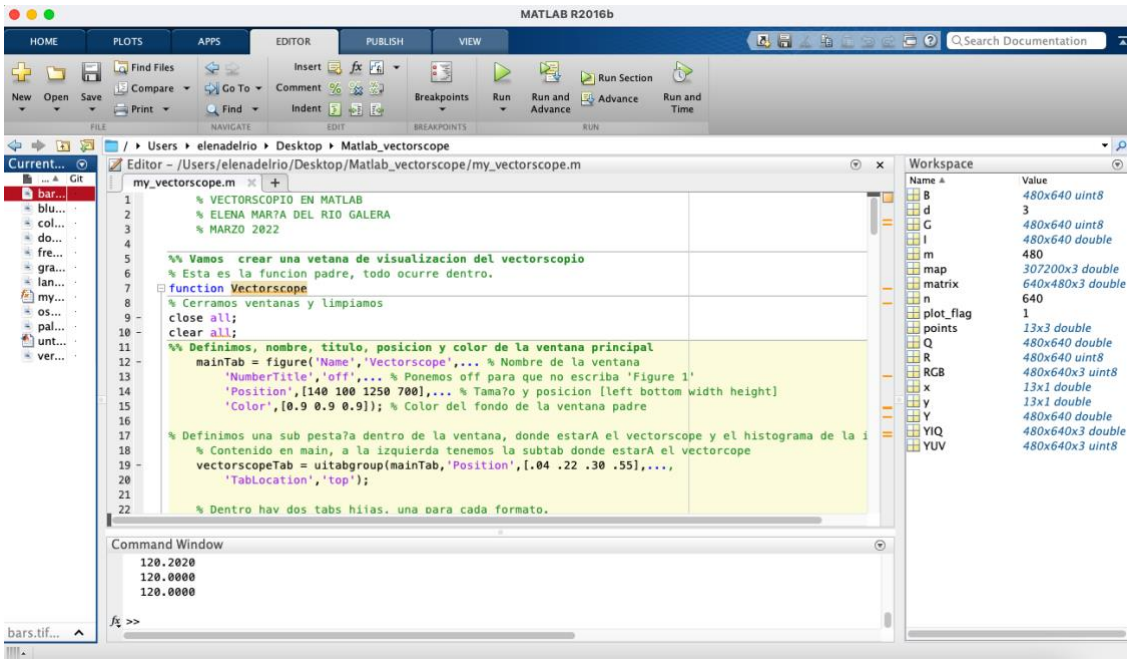

Figura 14. Ventana de trabajo de *Matlab*.

- ***Workspace*, espacio de trabajo en memoria**. Es la sección de la pantalla (situada por defecto a la derecha) donde *Matlab* muestra las variables creadas en la ventana de comandos. Todas las variables son consideradas como matrices. Incluso los valores simples, se almacenan del modo 1x1.
- ***Command Window*, ventana de comandos**. Es la sección de la pantalla dónde se ejecutan todas las operaciones y comandos. El comportamiento por defecto de *Matlab* una vez que se escribe algo, es mostrar a continuación el resultado de lo que se ha escrito. Si no se quiere que muestre el resultado, hay que terminar cada uno de los comandos con punto y coma (;) [23].

3.8. Android

Android es un sistema operativo de dispositivos móviles de código abierto, basado en el *kernel* de *Linux (Lovable Intellect Not Using XP*).

La ventaja que ofrece *Android* es la aproximación unificada de un desarrollo o dicho de otra manera, la orientación a la multiplataforma, es decir en este sistema operativo, se desarrolla una única vez y la aplicación es apta para múltiples dispositivos.

3.8.1. Versiones de Android

Desde la aparición de la primera versión, la evolución de *Android* ha sido enorme, igualando el nivel de desarrollo de la tecnología que nos rodea.

Figura 15. Versiones de *Android.* a. *Android 1.0* [24] b. *Android 12.0* [25].

Cada versión de *Android* ha metido cambios para adaptarse a los dispositivos del mercado, mejorar la conectividad, la usabilidad, la interfaz gráfica… En la imagen superior se ve una comparativa de la primera versión de *Android* y la última versión.

Hoy en día muchas versiones ya están en desuso, en la tabla se pueden consultar todas las versiones de *Android* con su fecha de lanzamiento y el porcentaje de usuarios que tiene cada versión no desfasada [26].

Versión Android	**Fecha de lanzamiento**	**% mayo 2022**
Android Apple Pie 1.0	23/09/2008	-
Android Banana Bread 1.1	09/02/2009	-
Android Cupcake 1.5	30/04/2009	-
Android Donut 1.6	15/09/2009	-
Android Eclair 2.0-2.1	29/10/2009	-
Android Froyo 2.2	20/05/2010	-
Android Gingerbread 2.3	06/12/2010	-
Android Honeycomb 3.0	22/02/2011	-
Android Ice Cream Sandwich 4.0	12/10/2011	-
Android Jelly Bean 4.1-4.2-4.3	09/06/2012	0,3%

Android KitKat 4.4	31/10/2013	1,0%
Android Lollipop 5.0	03/11/2014	3,0%
Android Marshmallow 6.0	05/10/2015	3,9%
Android Nougat 7.0	22/08/2016	5,0%
Android Oreo 8.0	21/08/2017	11,6%
Android Pie 9.0	06/08/2018	16,2%
Android 10.0	03/09/2019	23,9%
Android 11.0	08/09/2020	28,3%
Android 12.0	04/10/2021	6,2%
Android 13.0	15/08/2022	Versión preliminar

Tabla 3. Versiones de *Android* [27].

En el siguiente gráfico circular se puede ver porcentualmente la cantidad de usuarios de cada versión de *Android.*

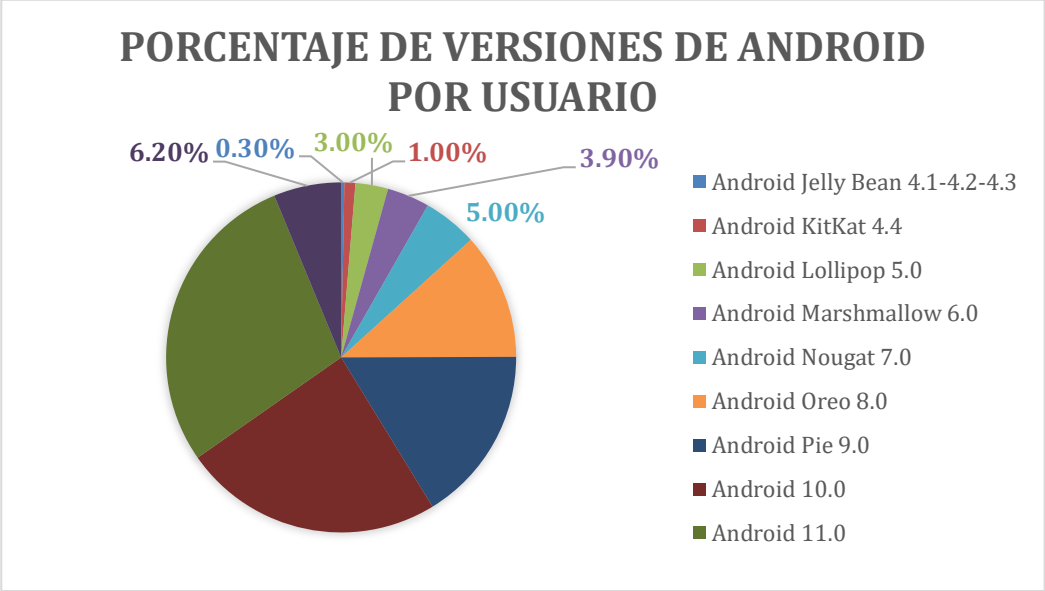

Figura 16. Gráfico circular de versiones *de Android.*

3.8.2. Aplicaciones nativas

Las aplicaciones nativas están desarrolladas para un sistema operativo móvil único. En la actualidad, en referencia a dispositivos móviles la mayoría de las aplicaciones se desarrollan para sistemas *Android* o *iOS*. Una aplicación programa para *iOS*, no es funcional en un sistema *Android*, y viceversa.

Aunque las aplicaciones nativas presentan un gran beneficio debido a su alto rendimiento y a la excelente Experiencia de Usuario, *User Experience* (Ux), también tienen un inconveniente importante relativo al coste. Para la creación y el mantenimiento de aplicaciones

para *Android* o *iOS*, hay que contar con dos equipos de desarrollo y esto provoca que los gastos de desarrollo y mantenimiento de una aplicación se vean incrementados.

En *Android* las aplicaciones se programan en dos lenguajes: *Java* o *Kotlin*.

Kotlin [28] es un lenguaje de programación de código abierto diseñado para funcionar en casi cualquier plataforma, principalmente *Android*, *Java Virtual Machine* (JVM), *JavaScript* y *Native*. Este lenguaje está inspirado en *Java*. Además, es 100% interoperable con *Java*: todos los marcos y bibliotecas de Java son compatibles con *Kotlin*, por lo que ambos pueden coexistir. Las razones por las que cada día más empresas optan por desarrollar las aplicaciones *Android* con *Kotlin* en vez de con *Java* son dos:

- reduce el tiempo necesario para el desarrollo, ya que necesita menos líneas de código para realizar la misma operación que en *Java*.
- Disminuyen los *crashes* de la aplicación porque el código es más conciso, lo que facilita el mantenimiento a largo plazo.

3.8.3. Android Studio

Android Studio es el *Integrated Development Environment* (IDE), independientemente del lenguaje de programación que se escoja, este IDE nos permitirá construir aplicaciones que corran sobre la JVM

En *Android Studio,* desde que se lanza el programa, en la pantalla de inicio se pueden crear nuevos proyectos, abrir proyectos ya existentes, importar proyectos de otros programas…

A la hora de crear nuevos proyectos es necesario poner nombre al proyecto, escoger la versión mínima de *Android* a partir de la cual el proyecto va a ser compatible y escoger la plantilla deseada para la parte gráfica.

En la Figura 17 se puede observar cómo es el espacio de trabajo en este programa. En la parte en la que aparecen las carpetas del proyecto, hay algunas muy relevantes como:

- ***manifest*:** fichero de configuración del proyecto.
- ***java***: código *Java/Kotlin*.
- ***res*:** recursos de la aplicación, como pueden ser vistas, imágenes, iconos, *layouts*…
- ***Gradle Scripts***: gestión del ciclo de vida de la aplicación *(app*).

En la parte horizontal arriba del proyecto se puede observar la ruta del archivo que está seleccionado. También se gestionan desde aquí las ejecuciones del programa o los dispositivos donde se quiere ejecutar.

En la parte de arriba, a la izquierda del todo se puede ver el árbol de carpetas y archivos del proyecto. En la parte de abajo se aprecia que está seleccionado el archivo *MainActivity.java*

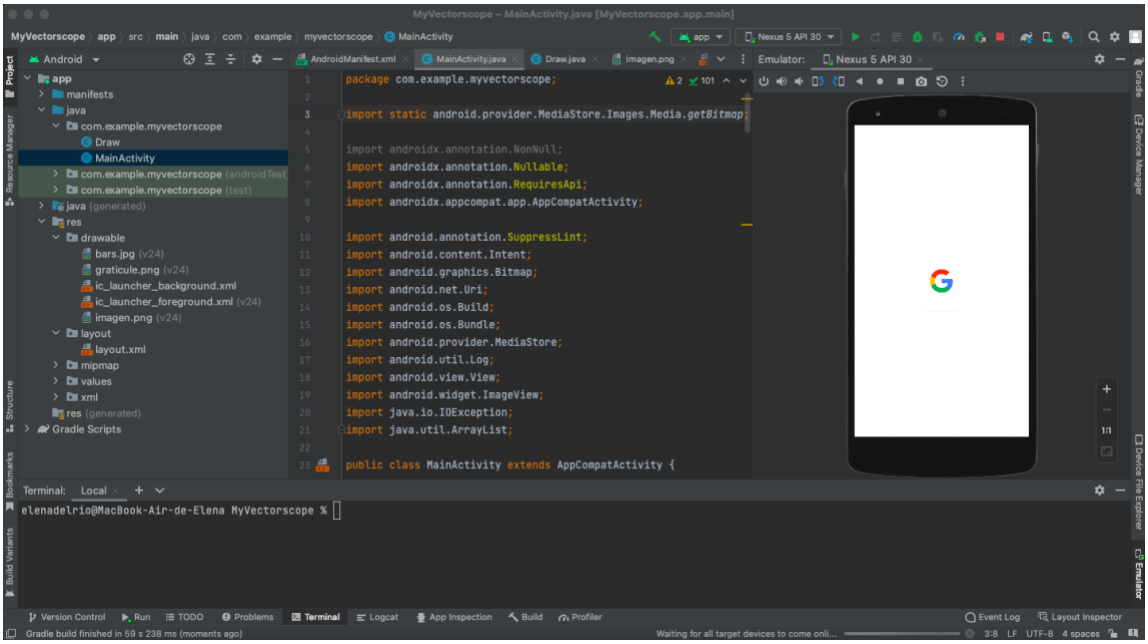

Figura 17. Ventana de trabajo de *Android Studio*.

Al lado, la ventana que destaca es la parte donde se abren los archivos del proyecto. Desde aquí se escriben las líneas de código para la parte visual como para la de bajo nivel, editando los archivos correspondientes.

En la parte derecha está abierto el espacio del emulador ya que la aplicación *Android Studio* permite crear dispositivos virtuales para probar las aplicaciones. En estos dispositivos, podremos utilizar los botones de navegación, encendido/bloqueo, volumen y rotación. También tendremos la posibilidad de configurar el *Global Positioning System* (GPS), nuestra conectividad, recursos, capturar la pantalla, etc.

Desde *Android Studio* también se pueden conectar dispositivos físicos al ordenador y lanzar las pruebas de la aplicación directamente en el dispositivo.

En la parte de debajo de la ventana de trabajo de *Android Studio* se encuentra abierto un terminal, aunque también hay otras ventanas seleccionables como *Version Control*, para consultar el estado del proyecto en *GitHub*, o *logcat*, donde se pueden ver los *logs* del código para depurar la aplicación.

Android Studio tiene muchas más funcionalidades disponibles, aunque aquí solo se han descrito las funcionalidades más relevantes [29].

3.9. Java

Java es un lenguaje de programación y plataforma de software.

Al igual que *Android*, este lenguaje es multiplataforma. Aproximadamente 6,8 millones de programadores utilizan *Java* para escribir código [30].

Java es un lenguaje de programación orientado a objetos lo que hace habitual que los programas desarrollados en este lenguaje contengan 'clases'. Las clases son subconjuntos de uno o más objetos.

El objetivo de *Java* es poder crear aplicaciones una vez y ejecutarlas en cualquier lugar. Este concepto se llama *Write Once, Run Anywhere* (WORA). Si un código ya ha sido compilado y el dispositivo donde se quiere utilizar admite *Java* no es necesario compilarlo de nuevo, directamente puede ser ejecutado.

Detrás de *C, Java* es el segundo lenguaje de programación más utilizado en el mundo y además tanto las descargas como las actualizaciones se ofrecen de manera gratuita [31].

```
if (savedInstanceState != null) {
    //Saca el estado de salida con el key guardado en putparceable
    imagenFinal = savedInstanceState.getParcelable( key: "bitmap");
    Log.d( tag: "STATE-RESTORE",  msg: "bitmap created");
    //Lo pone en el imageview.
    image.setImageBitmap(imagenFinal);
    Log.d( tag: "RESTORING...",  msg: "onRestoreInstanceState()");
}
```

Figura 18. Extracto de código escrito en *Java* para la aplicación *My Vectorscope.*

3.10. Extensible Markup Language

Extensible Markup Language (XML), Lenguaje de Marcado Extensible, es un lenguaje de marcado similar a HTML(*Hypertext Markup Language*) popularmente conocido por su extendido uso en páginas web. XML es una especificación de *World Wide Web Consortium* (W3C) como lenguaje de marcado no predefinido, lo cual implica que se puede ajustar a las necesidades del usuario que va a definir sus propias etiquetas a medida.

Hay diversos lenguajes populares que están basados en XML; Algunos de los más conocidos son: *Extensible HyperText Markup Language* (XHTML), *Mathematical Markup Language* (MathML) o *Scalable Vectors Graphics* (SVG). Como se ha comentado, también se puede crear un lenguaje de este tipo según las necesidades del usuario o del proyecto.

Los documentos XML, deben cumplir una serie de reglas sintácticas para ser válidos, estas reglas se recogen o se consultan en una Definición de Tipo de Documento (DTD).

En *Android* se utiliza XML para representar cada elemento de la Interfaz de Usuario (IU), ya que este lenguaje de marcado se utiliza de manera habitual para intercambio de datos entre aplicaciones [32].

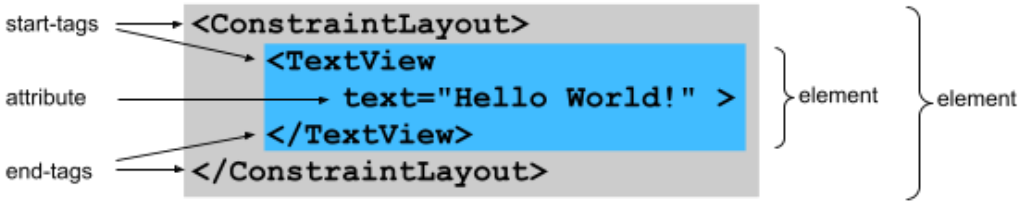

Figura 19. Ejemplo de archivo XML en *Android* [33].

En la Figura 19 se puede observar un archivo XML en *Android*, el elemento padre es *Constraint Layout*, y los demás elementos se organizan dentro del elemento padre. Por ejemplo, en este caso un *View* de tipo *text*, donde pone: *"Hello World!"*. En la siguiente figura se puede ver, como queda visualmente el archivo XML, en la pantalla del móvil:

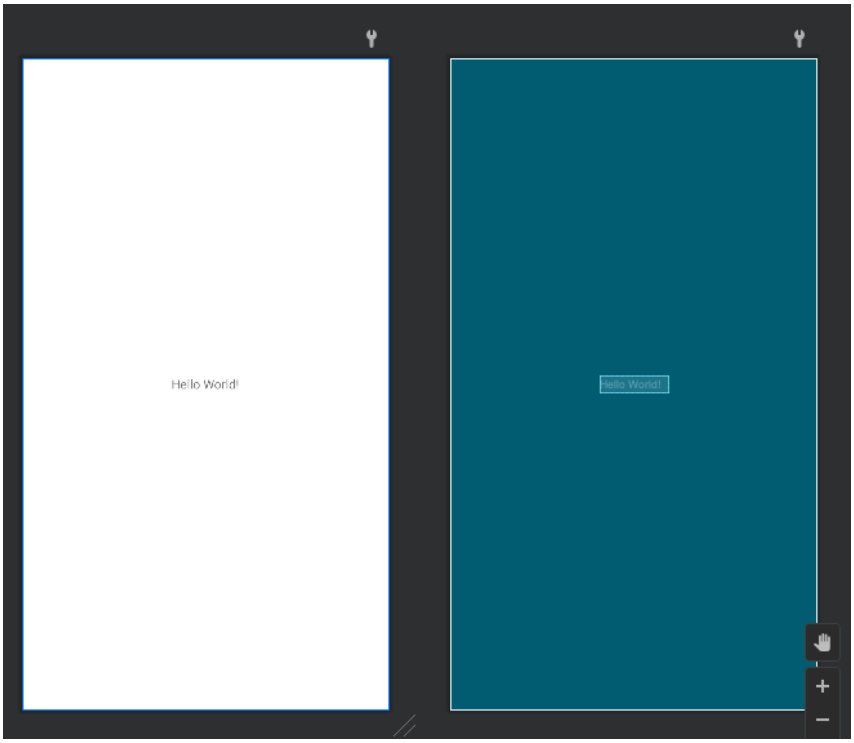

Figura 20. Parte visual del archivo XML.

3.11. Adobe After effects

Adobe After Effects es uno de los mejores *softwares* de diseño audiovisual y composición digital. Como su propio nombre indica es propiedad de *Adobe*. Es usado mayoritariamente para la posproducción de gráficos animados, así como para diseños de gráficos 2D y 3D. El sistema de edición que es utilizado por *Adobe After effects* es no lineal, además de la misma forma que *Photoshop* trabaja con el sistema de capas. La suma de filtros y efectos que se pueden utilizar en este programa es una larga lista que va desde los clásicos filtros de distorsión hasta sofisticados efectos de control de expresiones en tres dimensiones.

Uno de los puntos fuertes de *After Effects* son sus potentes herramientas de gradación y corrección, *Color de Lumetri*.

El espacio de trabajo *Color*, aunque es de especial apreciación para coloristas profesionales, está al alcance de cualquier usuario por su sencilla e intuitiva interfaz. [34].

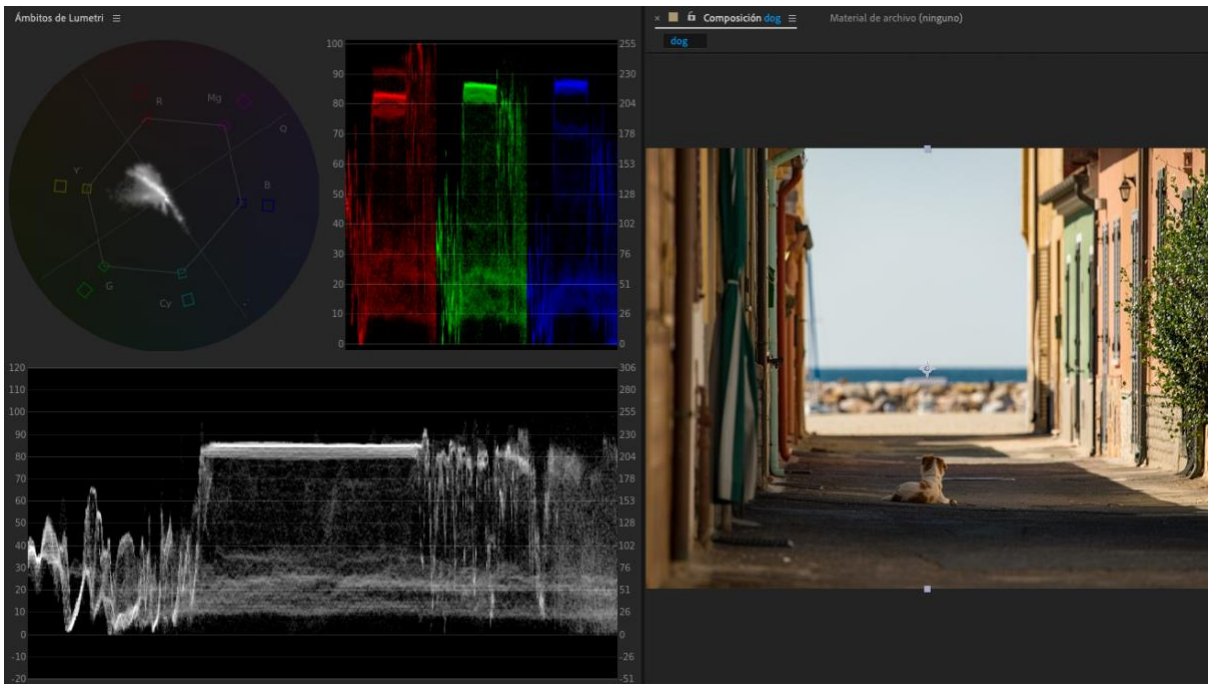

Figura 21. Ventana de corrección de color de imágenes de *After Effects.*

En la Figura 21 se puede ver el espacio de trabajo *Color de Lumetri* de *Adobe After Effects*. A la izquierda de la Figura, se aprecia un vectorscopio, a su lado la exposición RGB separada en canales, en la parte inferior se encuentra el monitor de forma de onda, en el que consultar la información relativa a la luminancia y por último a la derecha se observa la imagen de estudio.

4. Desarrollo del trabajo

En este capítulo se describen con detalle las implementaciones del vectorscopio en *Matlab* y posteriormente en *Android*.

4.1. Implementación en Matlab

A continuación, se describe la implementación del vectorscopio en *Matlab*.

El objetivo final del proyecto es el desarrollo de un vectorscopio en *Android*. Sin embargo, en la fase de desarrollo, se decidió comenzar en *Matlab*, ya que es una herramienta muy apropiada en el manejo de matrices y cálculo de componentes de imágenes, y permite llegar a un entendimiento en profundidad de lo que se pretende realizar
Así, el trabajo en *Matlab* sienta las bases de la aplicación en *Android* [35].

En la Figura 22 se puede observar el flujograma de funcionamiento a alto nivel del vectorscopio en *Matlab*.

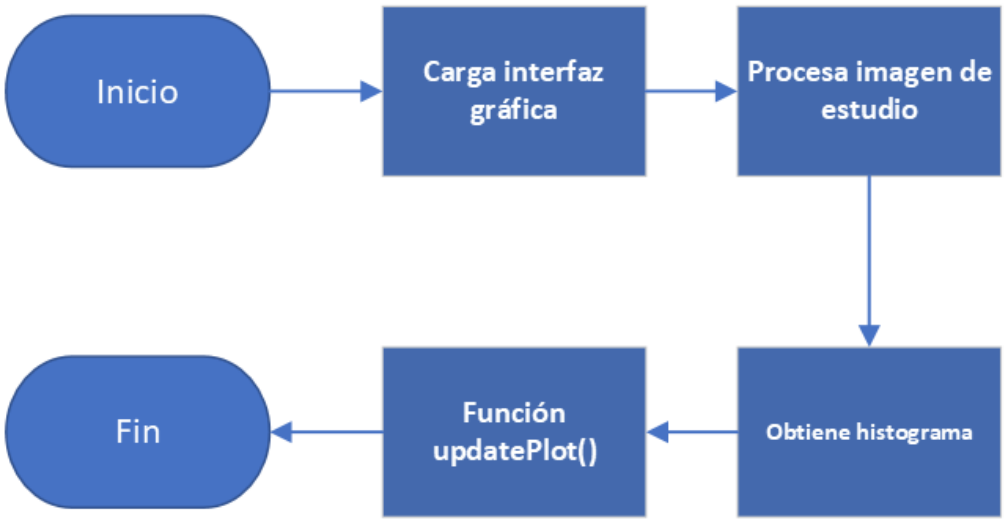

Figura 22. Flujograma de *Matlab* en alto nivel.

Para que el resultado sea más visual, en *Matlab* lo primero que se ha realizado con la función *figure* es crear la ventana principal llamada *Vectorscope*, dónde se ubicarán todos los elementos. A continuación, se define la sub-ventana en la parte derecha dónde se van a ubicar el vectorscopio y la gráfica del histograma. En el código de *Matlab* se especifica que por defecto aparezca marcada la sub-ventana que contiene el vectorscopio.

Se define el espacio que va a ocupar la imagen de prueba y se carga la imagen utilizando la función *imread*. En este caso, para cambiar la imagen de cálculo sería necesario cambiar desde código.

Una vez cargada la imagen, se calculan las filas, columnas y dimensiones, guardando estos valores en distintas variables para utilizarlas más adelante. En la Figura 23 se muestra el resultado de lo descrito anteriormente.

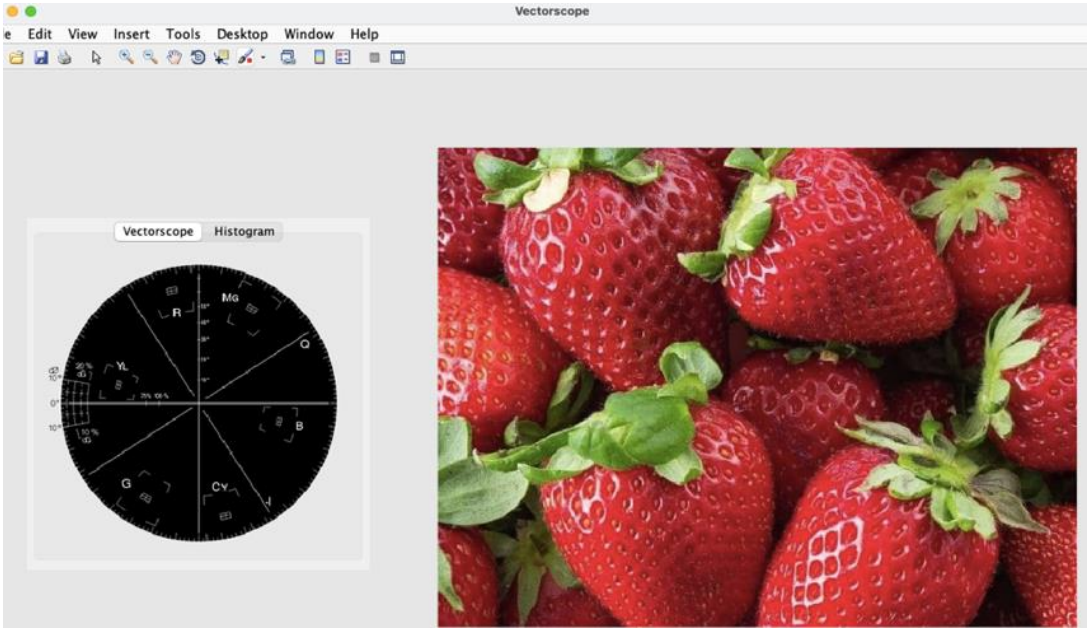

Figura 23. Ventana en *Matlab.*

Para el cálculo del histograma de la imagen, lo primero es convertir la imagen a escala de grises, que se lleva a cabo con la función *rgb2gray*. Para calcular el histograma, se utiliza *imhist*, introduciéndole la imagen y los niveles en los que se quiere normalizar, en este caso 256 (desde 0 hasta 255).

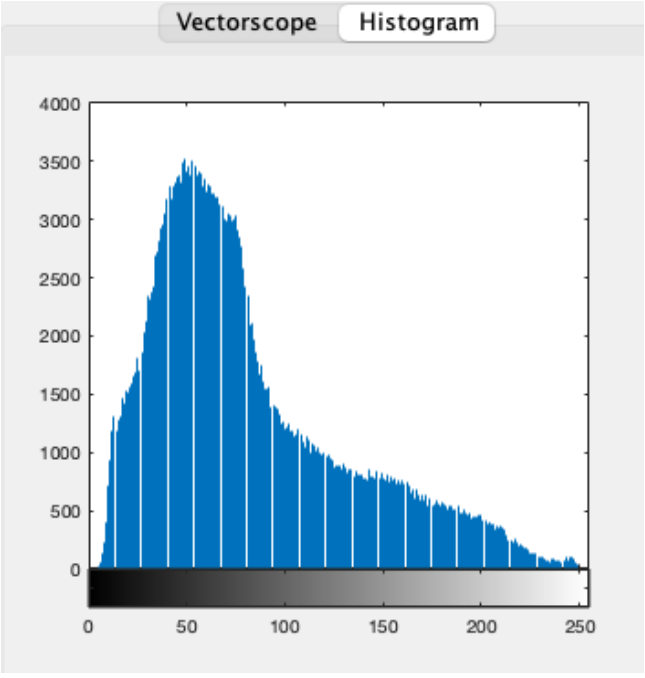

Figura 24. Histograma de la imagen de las fresas.

El histograma ofrece mucha información, como la representación gráfica de la distribución de color de píxeles en una imagen. También es útil para detectar el ruido, los valores de recorte o el fondo en una imagen. En el caso de la imagen de las fresas se observa como predominan los tonos oscuros.

El cálculo matemático relativo al vectorscopio se realiza en la función *updatePlot*, pero antes de esto es necesario definir algunos parámetros como el grado de rotación, tamaño, los ejes x e y, además del color del trazo. Estos parámetros se almacenan en una variable y posteriormente se llama a *updatePlot* para que se dibuje la relación matemática de color de la imagen, según indican los parámetros definidos.

En el flujograma que se muestra en la Figura 25 se puede apreciar cómo funciona *updatePlot.*

- Lo primero es convertir la imagen del espacio de color RGB a NTSC, que se realiza con la función *rgb2ntsc*.
- Con *permute*, se reordenan los valores de la matriz, se pasa de *m x n x d*, a una matriz de *n x m x d.*
- Con *reshape*, se convierte la matriz tridimensional, llamada matrix, en una matriz bidimensional, llamada *map* de m x n filas, y d, columnas, al ser la imagen original, RGB, el valor de d, es tres.
- Se crea una matriz, llamada *points*, del tamaño filas = m x n y columnas = d, y se inicializa a cero.
- Con *unique,* se comprueban las filas repetidas de la matriz bidimensional llamada *map*, y se almacenan en la matriz *points,* las filas únicas, es decir las no repetidas.
- En el eje x, se guardan los valores de la segunda fila de la matriz *points*, estos valores corresponden a los valores de la fase de la imagen.
- En el eje y, se guardan los valores de la tercera fila de la matriz *points,* estos valores corresponden a los valores de la cuadratura de la imagen.

Con esto, está lista la función *updatePlot* para dibujar la relación matemática en el vectorscopio [36].

Se llama a dicha función, pasándole como parámetros el color, tamaño, rotación del trazo. y la propia imagen.

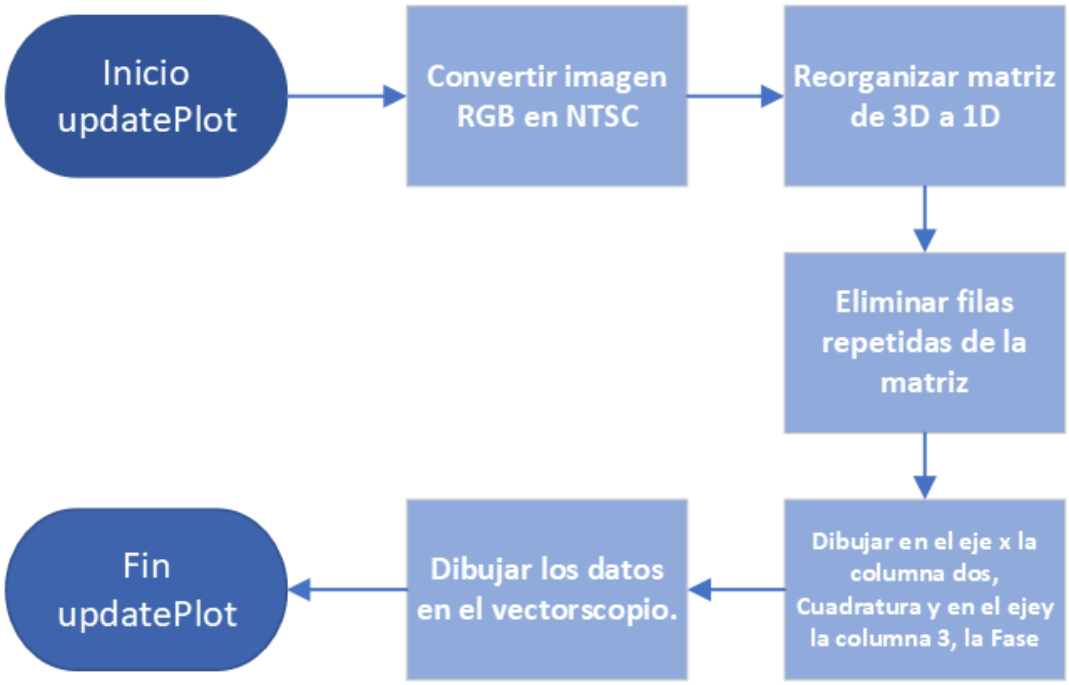

Figura 25. Flujograma de la función *updatePlot.*

En la Figura 26, se puede ver el resultado que muestra el vectorscopio tras analizar la imagen de prueba, que es correcto, ya que cada cuadrado del vectorscopio se corresponde con un tono puro (Rojo, Verde, Azul, Amarillo, Cian, Magenta). Los puntos del vectorscopio caen en el centro del cuadrado, hay otro punto en el centro dónde se ubican los colores que no tienen información de color: gris, negro y blanco.

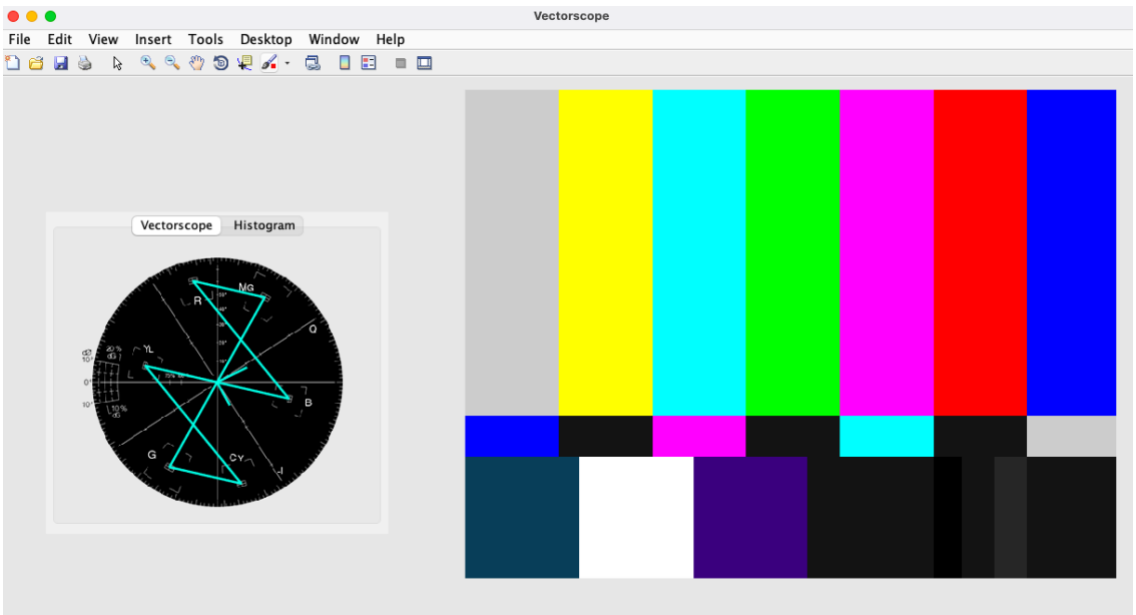

Figura 26. Vectorscopio en *Matlab* con imagen de barras.

4.2. Implementación en Android

Para implementar el vectorscopio en *Android* se ha utilizado el lenguaje de programación *Java* y como entorno de desarrollo *Android Studio*. Los proyectos en *Android Studio* se organizan en carpetas. En los siguientes apartados se mencionarán los archivos que se han manipulado para desarrollar la aplicación, aunque únicamente los más relevantes.

La aplicación que se ha desarrollado se llama *My Vectorscope*. En la Figura 27, se puede apreciar el flujograma de la aplicación, aunque en los siguientes apartados se profundizará en detalles concretos [37].

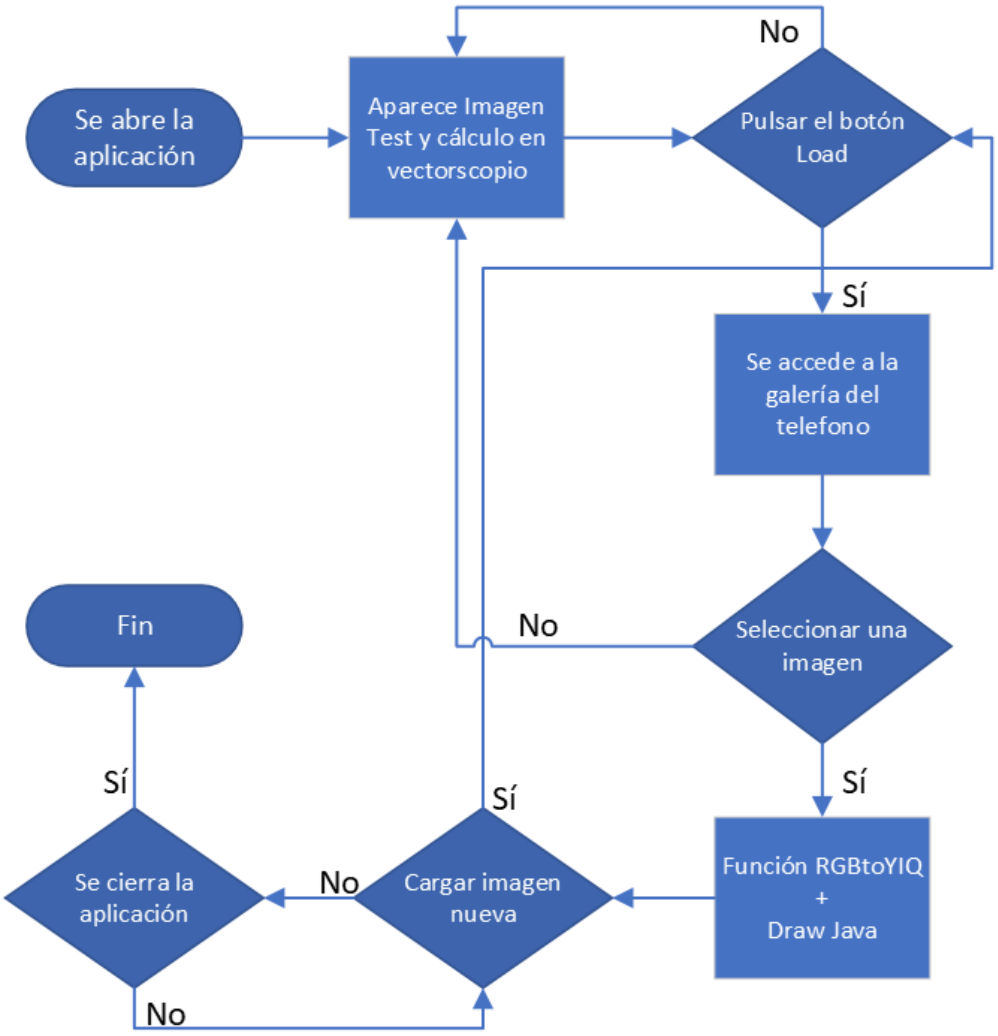

Figura 27. Flujograma de la aplicación en *Android*.

4.2.1. Archivo layout.xml

En cualquier aplicación *Android*, el archivo con extensión XML, determina cómo debe ser la interfaz de usuario global de la Actividad. Pudiera darse el caso de que hubiera varios

documentos con esta extensión. Por ejemplo, para aplicaciones con vista horizontal y vertical, o para aplicaciones con varias actividades, es decir pantallas. En el caso de *My Vectorscope*, existe solo una actividad, y la aplicación se puede visualizar solo en vertical, por lo que únicamente hay un archivo, *layout.xml*

Dentro del archivo *layout.xml* se define el esqueleto sobre el que va a ir todo lo demás. En este archivo se organiza la pantalla de forma visual.

En *Android Studio*, el archivo de extensión XML, presenta varias opciones de visualización: una de código, otra de diseño, y una tercera llamada *Split*, que combina el diseño y el código.

De esta forma el usuario puede trabajar cómodamente, a conveniencia. En la Figura 28, se puede observar el modo *Split* del *layout.xml* de la aplicación *My Vectorscope*. También se puede apreciar la posibilidad de elegir cualquiera de los otros modos de visualización.

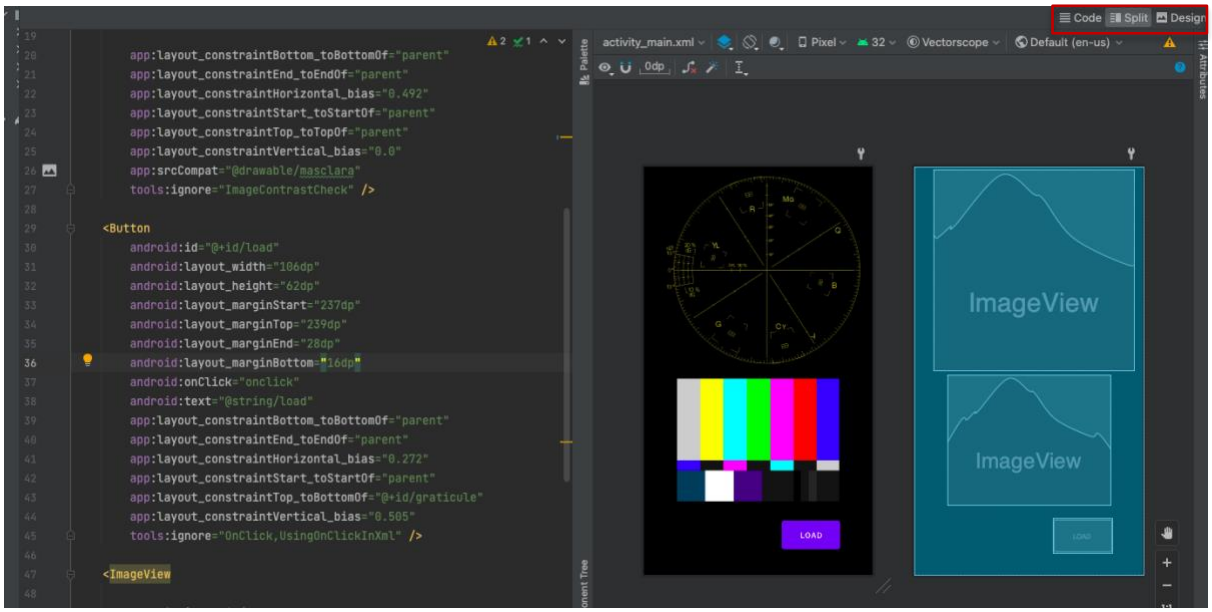

Figura 28. Vista de *layout.xml* en modo *Split*.

Para entender cómo se organiza el archivo XML, en el árbol de archivos del proyecto en *Android Studio*, hay que dirigirse a la carpeta *resources* (recursos), y una vez ahí, seleccionar la carpeta *layout* (diseño). Dentro de *layout*, se encuentra *layout.xml*, y de existir, el resto de los archivos XML también se ubicarían ahí.

Al seleccionar el archivo, y poner la vista de código, lo primero que se observa es un elemento de tipo *Constraint Layout* (Diseño de restricciones). La función de este tipo de elemento es contener elementos de tipo *View*, de una manera coloquial se diría que *Constraint Layout* es el contenedor padre.

En el caso de la aplicación *My Vectorscope*, *Constraint Layout,* contiene tres elementos *View*:

- ***Graticule***: es un *view* especifico del proyecto llamado *Draw*. Se profundizará un poco más adelante, pero es el *view* que muestra la rejilla del vectorscopio y pinta la relación matemática asociada a la imagen.
- ***Load*****:** es un *view* de tipo *Button*, en la aplicación es el botón que se muestra en la esquina inferior derecha y que funciona para cargar imágenes desde el almacenamiento del teléfono.
- ***Test Image:*** es un view de tipo *Image view*, aquí se ubica inicialmente la imagen de barras para calibrar el vectorscopio. Tras usar el botón *Load* y añadir otro tipo de imagen, esta imagen se ubicará en este *view* adaptándose al espacio. Si no se usa el botón *Load*, por defecto se muestra la imagen de barras.

Los objetos *View* se emplean para poder dibujar contenido en la pantalla del dispositivo *Android*. En este caso, dos imágenes y un botón. El esquema del archivo XML sería tal y como se muestra en la Figura 29. Como se ha comentado, *Constraint Layout*, es el contenedor padre, y los tres *views* son los hijos.

Una vez se comprende cómo se organiza la pantalla estructuralmente, y a dónde hay que acudir si se desea cambiar la distribución, se va a seguir profundizando en otros archivos

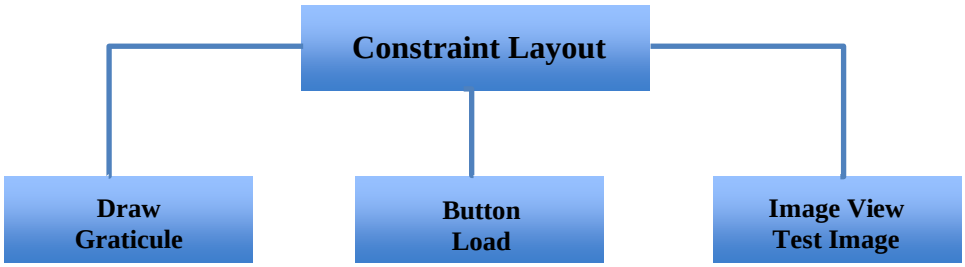

Figura 29. Esquema de la estructura del archivo *layout.xml.*

4.2.2. AndroidManifest.xml

El archivo *AndroidManifest.xml* se ubica en el directorio raíz de la *App.*

Este archivo muestra información imprescindible sobre la *App*. Previamente a la compilación, se consulta el archivo *AndroidManifest.xml*, ya que contiene información importante sobre la aplicación, como actividades, servicios, permisos...

Entre algunas de sus tareas más importantes se encuentra:

- Es responsable de proteger la aplicación, se proporcionan permisos para acceder a las partes protegidas.
- Declara el nivel mínimo de la *Application Programming Interface* (API) que *Android* que se requiere para la que la *app* funcione.

Algunas etiquetas que son comúnmente usadas en el *AndroidManifest.xml,* son:

- ***manifest***: es el elemento raíz del *Android Manifest*, contiene el nombre del paquete.
- ***Aplication*:** subelemento de *manifest* que a su vez contiene atributos como *icon*, *label* o *theme.*
- ***Activity*:** subelemento de *application.* Entre otros atributos, contiene el nombre de la *activity* a ejecutar, la orientación en la que se muestra la aplicación y el bloqueo de la pantalla para que solo se pueda ver en dicha orientación:

  android: name=" Main Activity"

  android: screen Orientation="portrait"

  tools: ignore="Locked Orientation Activity">

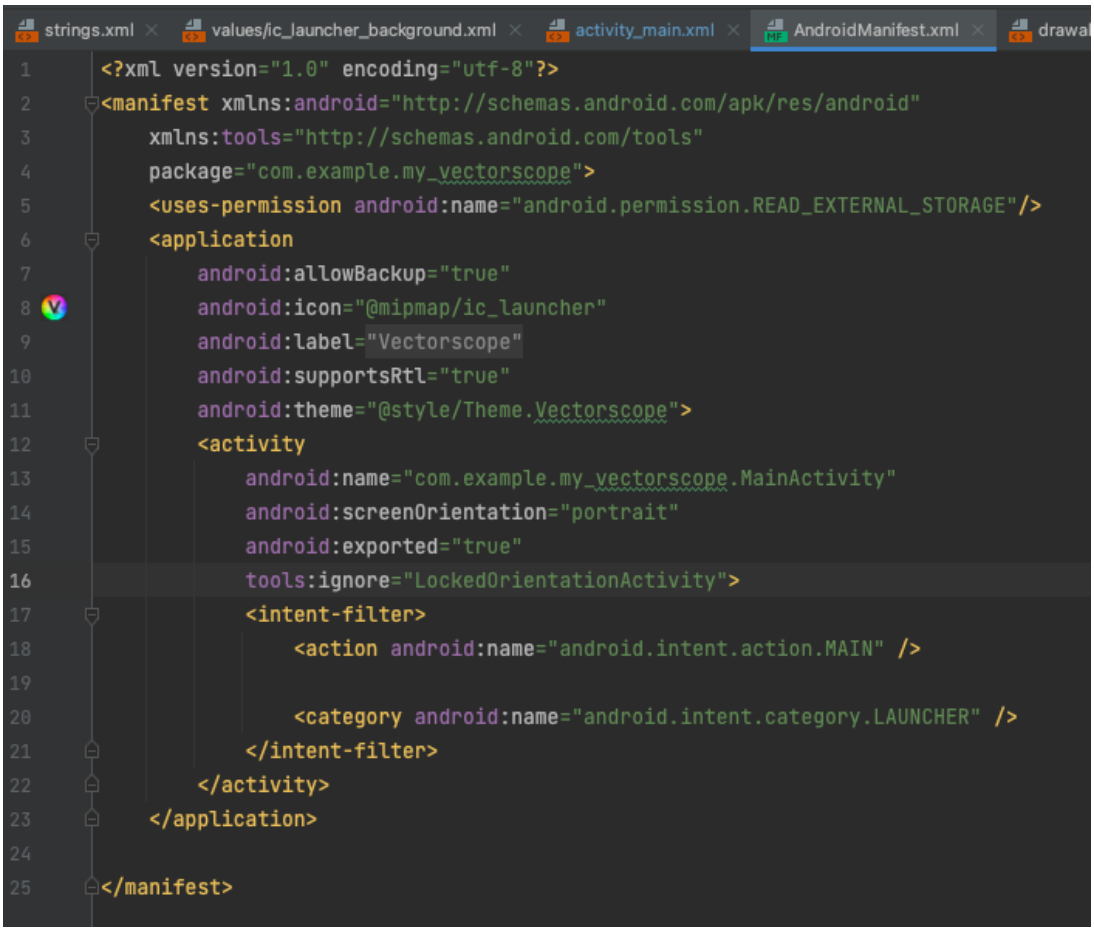

Figura 30. Archivo *AndroidManifest.xml* de la aplicación *My Vectorscope*.

- ***Intent.filter***: subelemento de *activity*.
    - ***Action***: subelemento de *intent*, Añade una acción al *intent-filter*. Al menos tiene que existir un *action* en el *intent filter*.
    - ***Category***: subelemento de *intent filter*. Añade un nombre de categoría al *intent-filter*.

  En el caso de la aplicación *My Vectorscope*, existe una etiqueta más, ***uses-permission***, dónde se definen los permisos especiales que va a tener la aplicación. En este caso el de leer en el almacenamiento externo, para cargar las imágenes de la galería en la *app* [38].

4.2.3. MainActivity.java

Dentro de la carpeta java de la aplicación se encuentra, el archivo *Main.java*. Casi todo el proceso de desarrollo de esta aplicación se ha dedicado en este archivo. Como se puede apreciar en el nombre de la extensión está escrito en el lenguaje de programación *java.*

Lo primero que se observa en este archivo es el nombre del paquete. A continuación, se importan las diferentes clases que van a ser usadas en las siguientes líneas de código.

Para toda la actividad de la superclase *Activity* se debe extender una subclase. Se definen las variables que se necesitan a nivel global. En *java* se definen escribiendo el tipo de variable seguido del nombre que se le otorga. A continuación, se puede inicializar dicha variable o no. Por ejemplo, las variables dónde se va a guardar la altura y la anchura de la imagen de cálculo, se declaran como, *int out Width*, e *int out Height*. Ya que los valores de alto y ancho de la imagen serán enteros.

Public void onCreate

El primer método que se define es el método *onCreate*. Se ejecuta la primera vez que se crea la actividad, cuando esto ocurre, se queda en el estado *created*. Es importante entender que el método *onCreate* solo ocurre una vez porque es el encargado de ejecutar la lógica de arranque básica de la aplicación.

En este método, se recuperan los elementos necesarios del XML, por ejemplo, la rejilla dónde se mostrará la relación matemática del color de la imagen, presenta un id *'graticule'*, y es un *image view* en el archivo *layout.xml*. por lo que, con la línea, *graticule = findViewById (R.id. graticule);* se consigue que en adelante en el archivo *Main.java*, cuando refiera *graticule*,

está asociado con la parte estética de la aplicación con el XML. Exactamente igual se realiza con la imagen de prueba.

Public void onClick

Esta función, es la que se ejecuta cada vez que se pulsa el botón *Load*, se asocia con el *button view* directamente en el archivo XML, desde la línea *Android: onClick=”onclick”*.

El objetivo de esta función es acceder al almacenamiento interno del teléfono cada vez que se pulse. Por ello, al pulsar abre las aplicaciones que pueden acceder a contenidos de tipo imagen, y muestra el mensaje “Seleccione una aplicación” como se especifica en el código. En la Figura 31 se muestra un ejemplo.

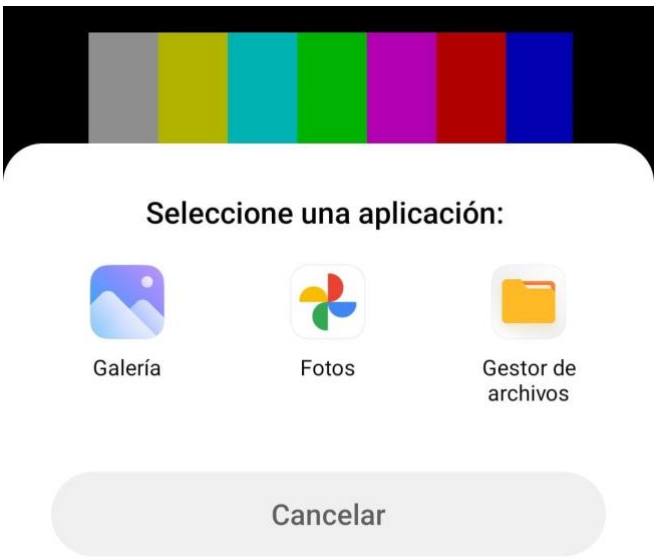

Figura 31. Pantalla que se muestra al pulsar *load.*

Public void onActivityResult

En la función *onClick*, cuando se accede al almacenamiento del teléfono para seleccionar una imagen, también se lanza *staractivityforresult*, lo que quiere decir, que una vez que se selecciona una imagen, se empieza una actividad que espera un resultado. El resultado se encarga de devolverlo *onActivityResult.*

Lo primero que evalúa este método, es si se ha seleccionado una imagen. En caso afirmativo, se sacan los datos de dicha imagen y se convierte de *Uniform Resource Identifier* (URI) a mapa de *bits* (*bitmap),* luego se redimensiona. En función de los valores de la imagen, se calcula una redimensión proporcionada, con un tamaño máximo definido para que la aplicación no consuma demasiados recursos y se interrumpa el proceso de ejecución.

Se crea el *bitmap* relativo a la imagen, usando los valores del *bitmap* de la imagen, junto con los valores del ancho y del alto calculados. *Imagen Final = Bitmap.createScaledBitmap(bitmap, outWidth, outHeight, true);* Después se coloca la imagen en el *image view* para que el usuario pueda verla. *Image. setImageBitmap (imagen Final);*

Por último, dentro de esta función se llama a la función *RGBtoYIQ ()*, que es la que se encarga de realizar todos los cálculos.

Public void RGBtoYIQ

Esta función es de creación propia, con ella se busca, tratando los componentes de la imagen deseada por separado, cambiar al espacio de color YIQ.

Para conseguir cambiar del espacio de color RGB al YIQ, lo que se hace, es con la ayuda de un bucle recorrer, en cada fila, cada columna, y cada dimensión, siendo las dimensiones 3, R, G y B. Con *getPixel (x, y)* se obtiene el píxel asociado a dicha posición de la imagen.

Los píxeles son números enteros de 4 *bytes*. El primer *byte* se corresponde con la transparencia, el siguiente con el valor del color rojo del píxel, el tercer *byte* con el valor de color verde y el último con el color azul.

Por lo tanto, el siguiente paso en el bucle es separar los componentes RGB.

Para hallar el color rojo, hay que moverse 16 *bits* a la derecha *r = (píxel >> 16) & 0xff;* Lo que se consigue con esa línea es que el valor rojo sea el valor más a la derecha, para eliminar la transparencia se usa 0xff.

A continuación, para calcular los valores asociados al color rojo y verde, se realiza algo similar, lo único que cambian son las posiciones que hay que moverse a la derecha. En el caso del verde, 8 *bits*, *g = (píxel >> 8) & 0xff*; y en el caso del azul ninguno. *B = píxel & 0xff*;

Una vez calculados los valores RGB asociados al píxel, se procede a realizar la transformación al espacio YIQ. El cálculo para pasar a este espacio de color se muestra en la Figura 32.

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.275 & -0.321 \\ 0.212 & -0.523 & 0.311 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Figura 32. Cálculo para la conversión de RGB a YIQ [39].

El resultado se almacena en una matriz tridimensional *matrizYIQ[x][y][z]*, donde el valor de x corresponde a cada fila de la imagen, el valor y corresponde a cada columna y el valor z a cada dimensión, YIQ. Por eso tras el cálculo de cada componente, se especifica que el valor de Y, se almacene en la dimensión cero, el valor de I, en la dimensión 1, y el valor de Q, en la dimensión 2. Tras este bucle, se consigue haber pasado de una imagen RGB a una imagen YIQ.

La segunda parte de esta función es convertir una matriz tridimensional a una bidimensional, la nueva matriz 2D, será de la forma *matriz2D [x * y][z]*. Es decir, una matriz que tenga como fila, el número de filas de la *matrizYIQ* por el número de columnas de esta, y como columnas las dimensiones, en principio siempre van a ser 3.

Para formar la matriz de dos dimensiones, es necesario un nuevo bucle, que recorre las dimensiones de la matriz 3D, en cada dimensión se recorren las filas, y en cada fila se recorre cada columna.

Para cada valor recorrido, en el bucle anidado se comprueba en qué dimensión se está iterando, si es la dimensión 0, corresponde con el valor de la luminancia, que se quiere almacenar en la primera columna de la matriz bidimensional. De igual manera valores de la segunda dimensión se almacenan en la segunda columna de la matriz bidimensional, valores de la fase, y los valores de la tercera dimensión, en la tercera columna, que es la cuadratura.

En la figura inferior se puede ver el extracto del código comentado relativo a esta acción.

```
150         //Inicializamos la u, nueva fila de la nueva matriz en cero
151         int u = 0;
152         // Recorremos las dimensiones
153         for (int k = 0; k < dimensiones; k++) {
154             for (int j = 0; j < outHeight; j++) {
155                 for (int i = 0; i < outWidth; i++) {
156                     if (k == 0) {
157                         //Rellenar primera columna con La información relativa a Luminancia
158                         matriz2D[u][0] = matrizYIQ[i][j][k];
159                     } else if (k == 1) {
160                         //Rellenar segunda columna con La información relativa a in-phase
161                         matriz2D[u][1] = matrizYIQ[i][j][k];
162                     } else {
163                         //Rellenar tercera columna con La información relativa a quadratura
164                         matriz2D[u][2] = matrizYIQ[i][j][k];
165                     }
166                     //Una vez relleno dato, saltamos a la siguiente fila
167                     u++;
168                 }
169             }
170             // Para la siguiente dimensión, empezamos en la fila 0, reiniciamos la u
171             u = 0;
172         }
```

Figura 33. Extracto del código. Bucle para crear matriz 2D.

Tras los dos bucles anteriores, se consiguen obtener los valores de Luminancia, Cuadratura y Fase de una forma que se puedan manipular. Es por eso, que el paso siguiente, es utilizar solo lo que se necesita, es decir la cuadratura y la fase, y almacenar estos datos de una manera ordenada. Para eso se crea un nuevo bucle que recorre la matriz bidimensional y almacena los valores de la fase, en una lista llamada Fase, que es un *Array List*. Con la cuadratura se realiza lo mismo, almacenando los valores en un *Array List* llamado Cuadratura.

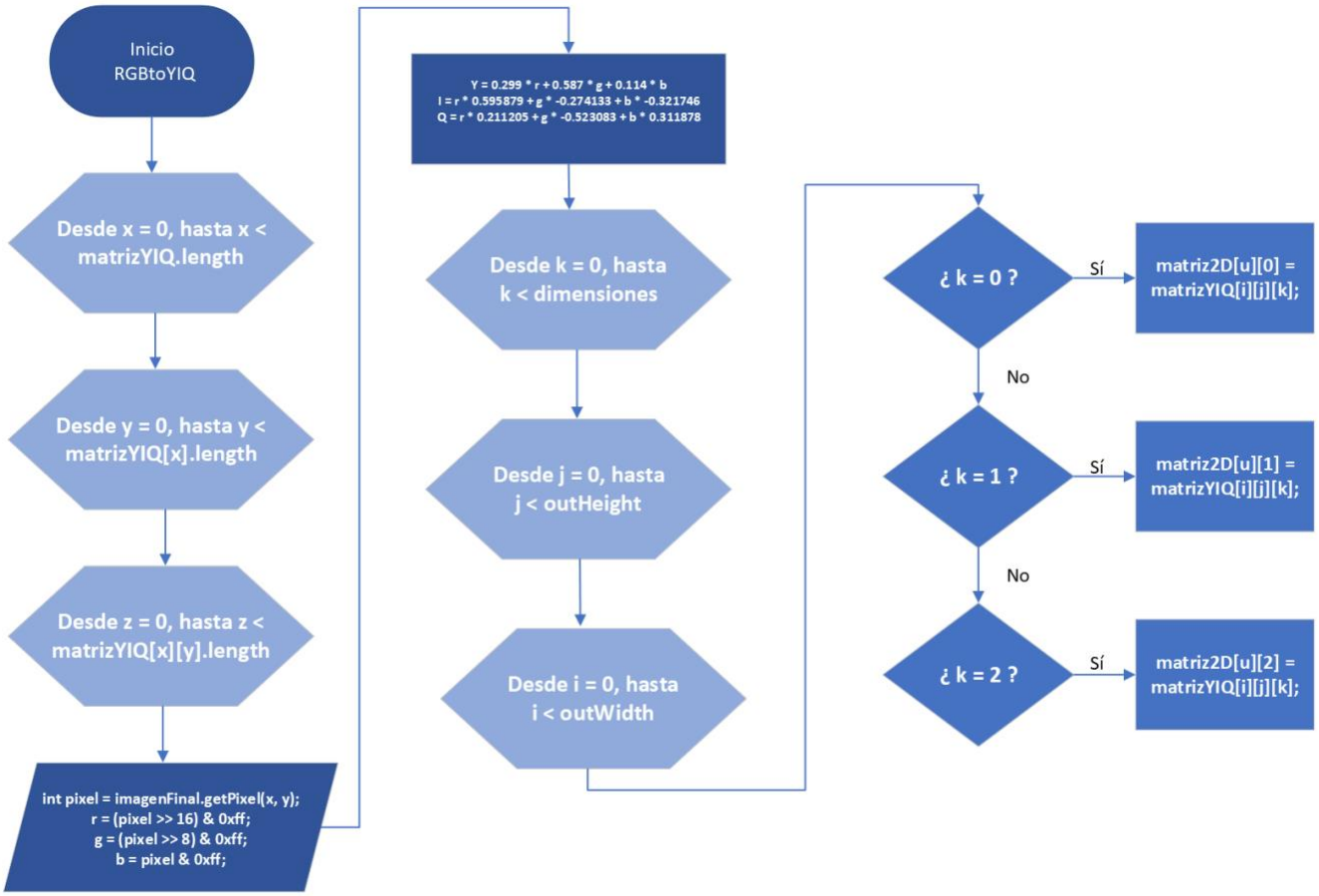

Figura 34. Flujograma de una parte de la función *RGBtoYIQ ()*.

La clase *array list* en *Java*, utiliza un array dinámico para almacenar elementos. La ventaja que ofrece respecto a un *Array* en *Java* es que no es necesario declarar su tamaño.

El último paso en *Main.java*, es crear dos variables de tipo array *float* públicas, para poder usarlas en otra clase. Al final de *RGBtoYIQ*, se calcula el tamaño de los dos *array list* y se programan dos bucles *for* independientes, uno para la cuadratura y otro para la fase, se recorren los *array list* de cuadratura y fase para conseguir cada valor, y guardarlo en un *array* [40].

4.2.4. Draw.java

Con lo explicado en el apartado anterior, se tienen todos los valores calculados que son necesarios para poder representar la relación matemática de la cuadratura y la fase en el vectorscopio. Para conseguir dibujar está relación y que se muestre por pantalla se crea una nueva clase, en la misma carpeta dónde se encuentra el archivo *MainActivity.java.*

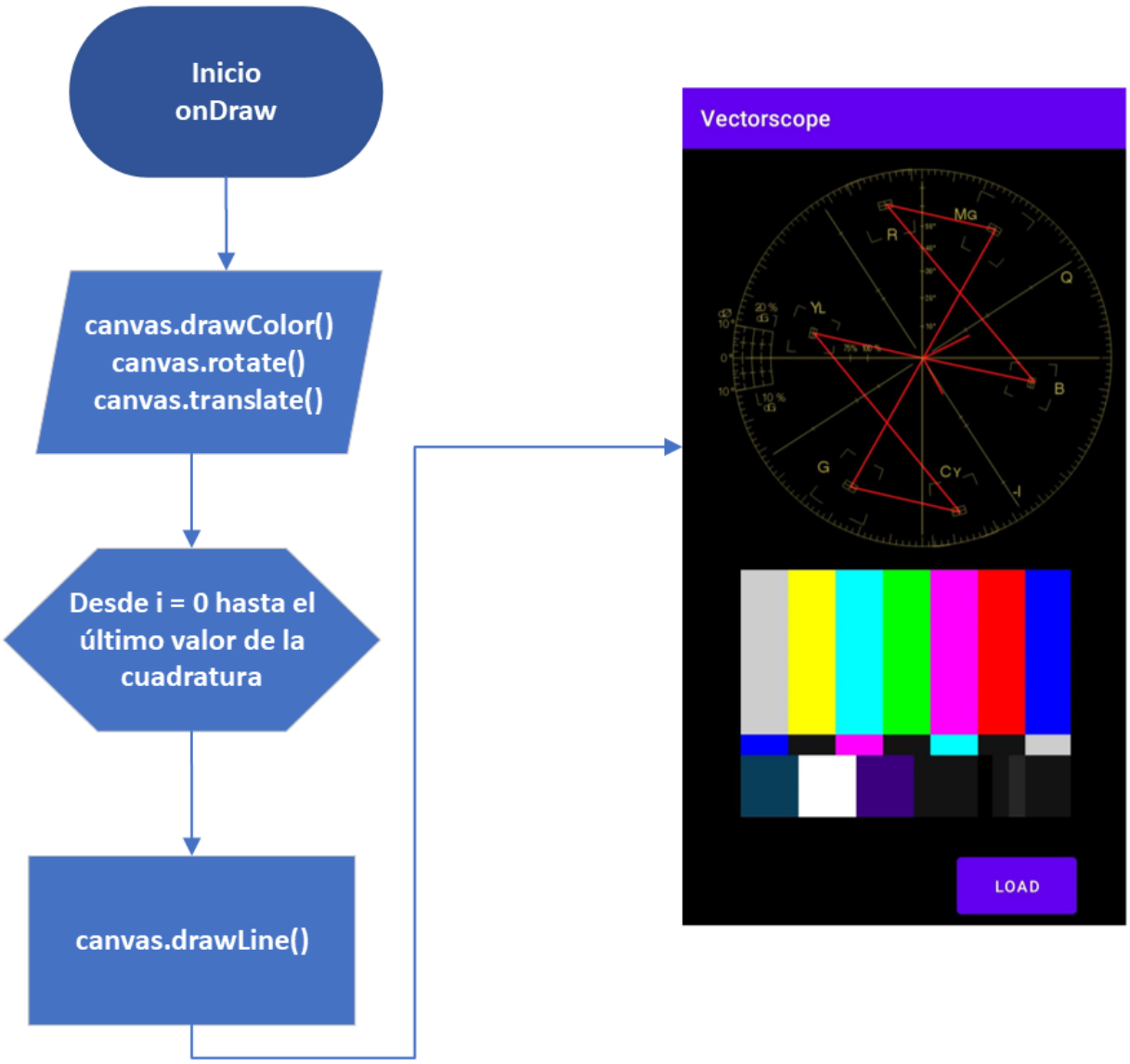

Figura 35. Flujograma a alto nivel de la función *onDraw.*

Esta clase extiende directamente del *image view* correspondiente a la gráfica del vectorscopio. Dentro se define una función llamada *setupDrawing*, dónde se indican los parámetros con los que se va a pintar, es decir, el grosor del trazo, el color de la línea, la forma de la pincelada, etc.

La otra función importante es la llamada *onDraw***,** en esta se define que el *canvas* sea transparente para que se no tape la gráfica del vectorscopio. A continuación, se rota la pintura y se traslada para indicar que el cero (origen de coordenadas) está en el centro del vectorscopio. Se define que en el caso de que la aplicación acaba de iniciar y esté la imagen de prueba, se dibuje la relación matemática asociada a la imagen de prueba.

En caso de que lo que se quiera representar sea una imagen que ha escogido el usuario, se recorre un bucle desde cero hasta el número máximo de elementos relativos a la cuadratura que presente dicha imagen. Para cada valor se dibuja una línea desde el valor de cuadratura hasta el de fase [41].

4.2.5. Otros.

Icono de la aplicación

En cuanto a la parte estética de la aplicación, es importante comentar que el icono de la aplicación es de diseño propio. Se trata de la rejilla del vectorscopio, encima de la rueda de color, además se pueden observar las letras Vs, de *Vectorscope*.

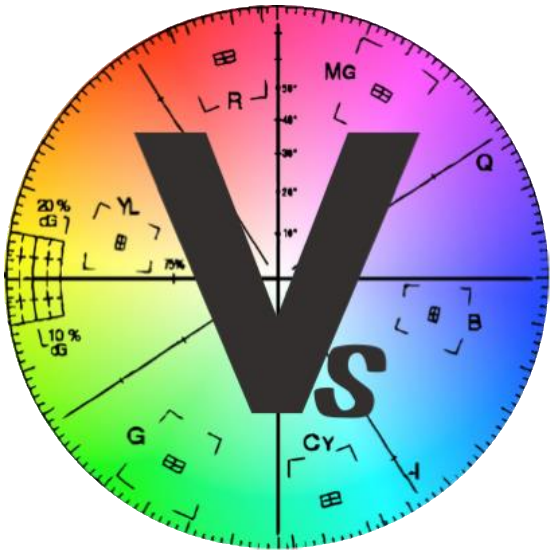

Figura 36. Icono de la aplicación.

La imagen se añade una vez que ya se encuentra alojada en la carpeta *drawable*, con el resto de las imágenes que se usan en el proyecto, seleccionando la carpeta de la aplicación, clic derecho, *New Image Asset*. Desde esa pantalla de configuración se carga la imagen y se ajusta el color de fondo, el recorte, la posición, etc. En esta pantalla se crea el icono que aparecerá en *Google play*, el icono *Legacy*… etc.

En la figura inmediatamente inferior se puede ver junto con otras aplicaciones como se muestra en el teléfono.

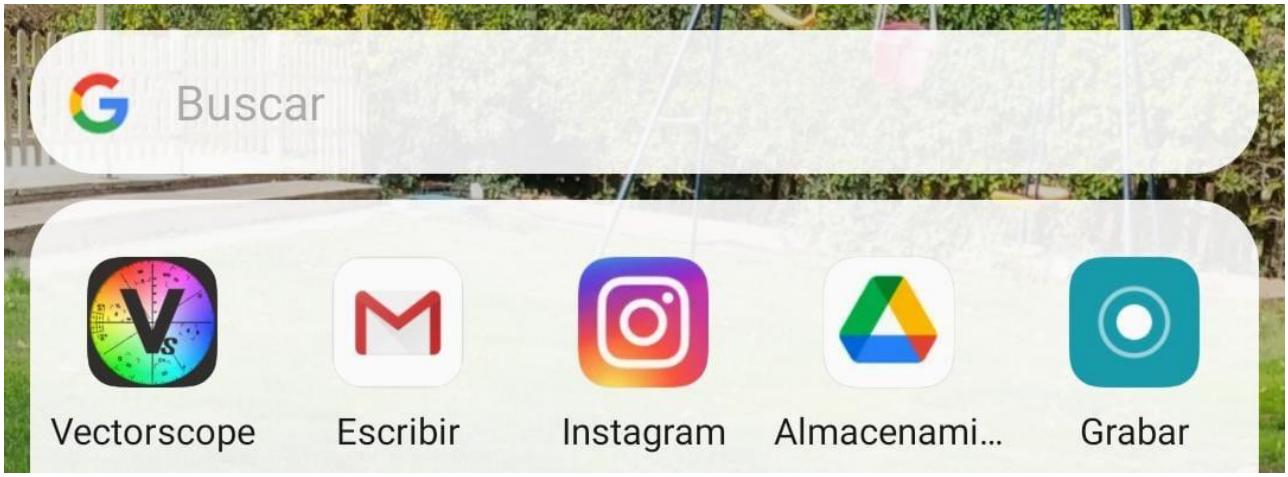

Figura 37. Icono de la aplicación junto con otras aplicaciones en el menú del *smartphone*.

Strings.xml

Una buena práctica de programación es usar este archivo, en el cual se almacenan los nombres que se usan en el archivo *layout.xml.* En la aplicación es el caso del texto del botón *load*, que se carga desde este archivo.

Este archivo se ubica en */res/values.*

Colors.xml

Al igual que el archivo *strinsg.xml*, se almacena en */res/values*. También es una buena práctica, disponer detodos los colores que se usan en la aplicación, definidos en este archivo de la manera:

<color name=”purple_200”>#FFBB86FC</color>

Es decir, el nombre del color, y el hexadecimal asociado al color, todo ello escrito en el lenguaje de marcado XML.

Drawable

En la carpeta *drawable* que es una carpeta dentro de */res*, se almacenan las imágenes usadas en la aplicación.

En este caso, la imagen de barras, la rejilla del vectorscopio, y el icono de la imagen.

Visualización final

Por último, en este apartado se pueden observar dos ejemplos con diferentes imágenes de los resultados del vectorscopio. Ambas imágenes son tomadas por un usuario desde el mismo dispositivo móvil desde el que se está probando la aplicación.

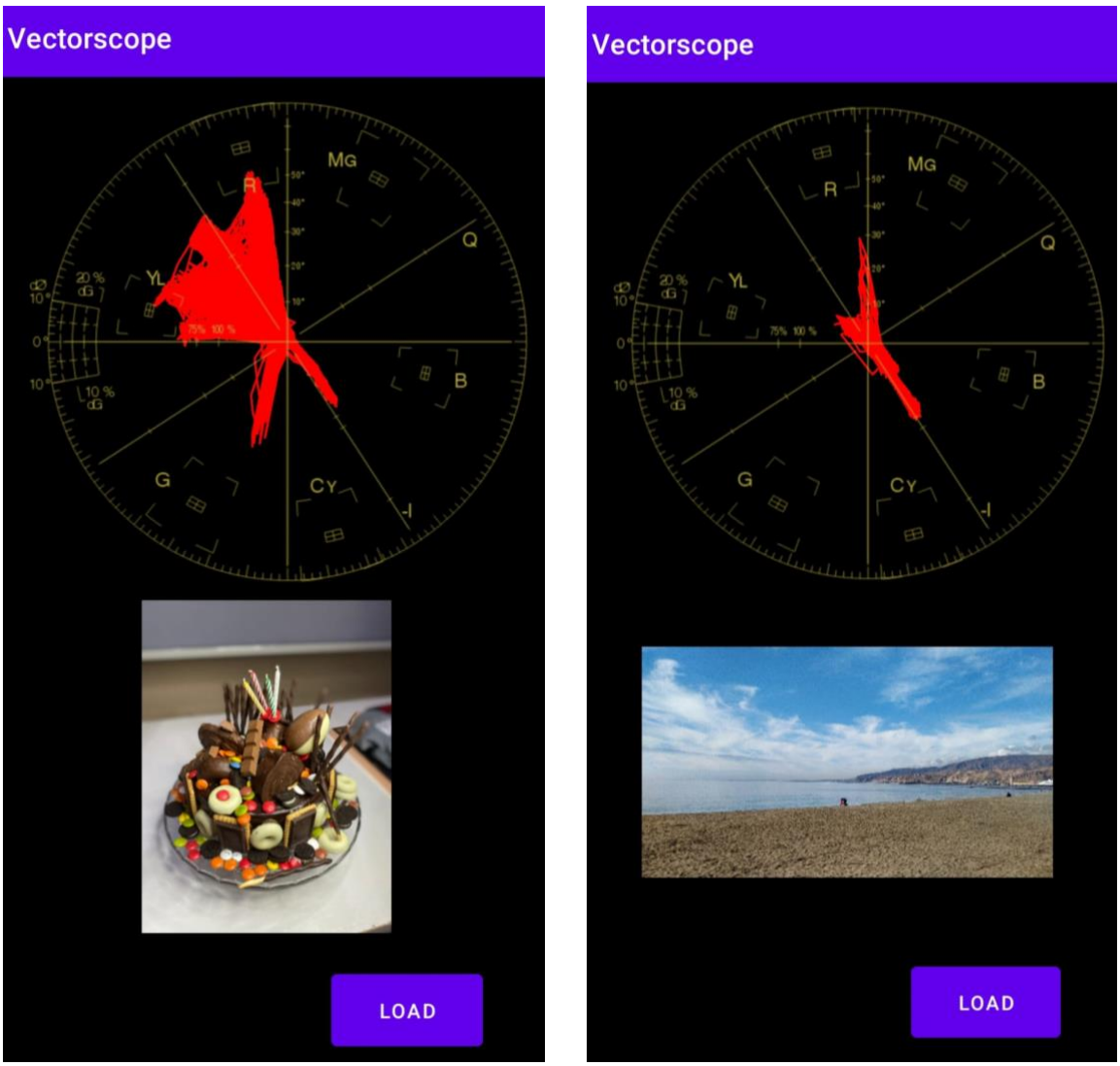

Figura 38. Ejemplos de visualización de imágenes en la aplicación *My Vectorscope*. (a) Tarta en vertical. (b) Cielo arena y mar en horizontal.

5. Resultados

En este capítulo, se van a mostrar y describir los resultados obtenidos al ejecutar el código tanto en Matlab como en *My vectorscope*. Además, se compararán ambos resultados con el vectorscopio que incorpora una herramienta profesional. Es importante recordar que un vectorscopio es un tipo especial de osciloscopio que muestra una representación gráfica de los colores presentes en una imagen y sus intensidades.

Cuando se alimenta una señal de video en un vectorscopio, los puntos que representan cada color que está presente en la señal se colocarán en la forma aproximada de un hexágono, dentro de la pantalla circular. Factores importantes para tener en cuenta en el estudio de las imágenes siguientes:

- la distancia desde el centro de la pantalla hasta cada uno de los puntos de color representa la intensidad o saturación de cada color. Una línea corta indica un color menos saturado, una línea larga equivale a un color más saturado.
- Blanco, negro y gris generan un punto en el centro de la pantalla porque esas señales no contienen información de color.

En la siguiente figura se puede observar el hexágono que forman los colores dentro de la rueda del vectorscopio en el programa *After Effects*, que se usará para comparar los resultados obtenidos con los de un *software* profesional [42].

Figura 39. Ejemplo del hexágono que forman los colores en el vectorscopio de *After Effects.*

En la Figura 40, se puede ver en detalle un cuarto del vectorscopio. En la parte curva se distinguen diferentes líneas que se repiten a cada poco, las líneas representan los grados. En la

Figura 9, se pueden consultar los grados en los que se encuentra cada tono puro que representa el vectorscopio. Cada color primario y complementario se enmarca en dos cajas, o *"boxes"*. Una contenida en la otra como se puede ver en la figura inmediatamente inferior.

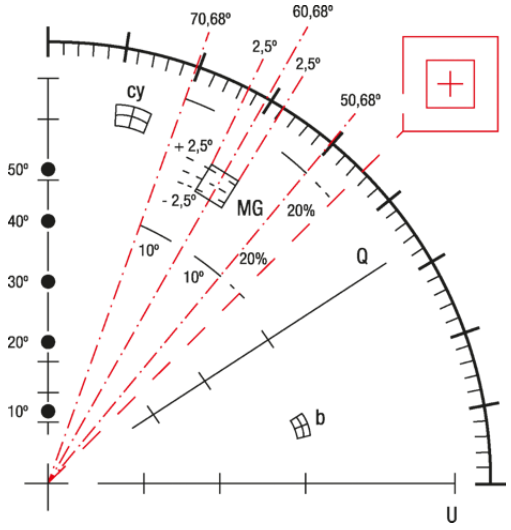

Figura 40. Detalle ampliado de un cuarto del vectorscopio [43].

La caja exterior, que es la de mayor tamaño ocupa ±20 grados, en el caso de ejemplo, el magenta (cuyo tono puro se ubica en los 60'68º), la caja grande se extiende desde el 70'68 º hasta el 50'68º, si se hace la resta, se obtienen los 20º. El tono puro respectivo al magenta se encuentra justo en la mitad de la caja grande, con ±10º a cada lado. Respecto a la caja de menor tamaño, ocupa un total de ±5º. Desde el centro absoluto ±2'5º a cada lado de la caja pequeña.

5.1. Imagen de barras

La primera imagen de estudio es la imagen de prueba, que tradicionalmente se utiliza para comprobar el funcionamiento del vectorscopio.

En este caso se utiliza una imagen de barras de color SMPTE. Estas barras se utilizan para alimentar a televisores y monitores y así poder verificar sus pantallas de color. Las barras de color representan los tonos puros de cada uno de los colores primarios y complementarios.

Cuando se prueba la imagen de prueba en un vectorscopio, un punto caerá (o debería) colocarse en su lugar dentro de cada cuadro que representa los colores amarillo, verde, cian, azul y magenta presentes en la señal. Los colores blancos, grises y negros, al no tener representación de color, solo generan un punto en el centro. Las líneas que parten del centro hasta el cuadro de cada color representan la saturación de cada color.

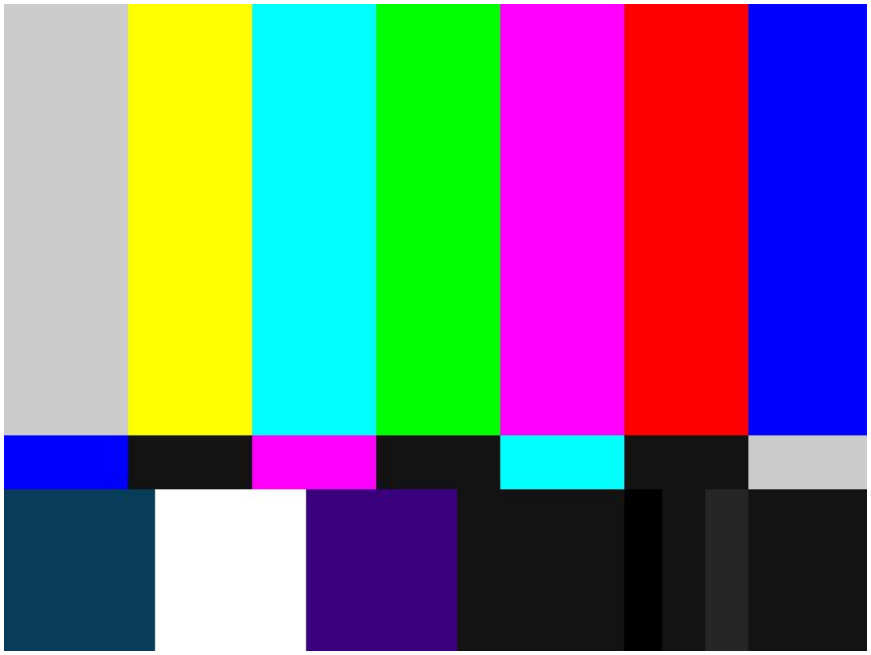

Figura 41. Imagen de barras o de prueba [7].

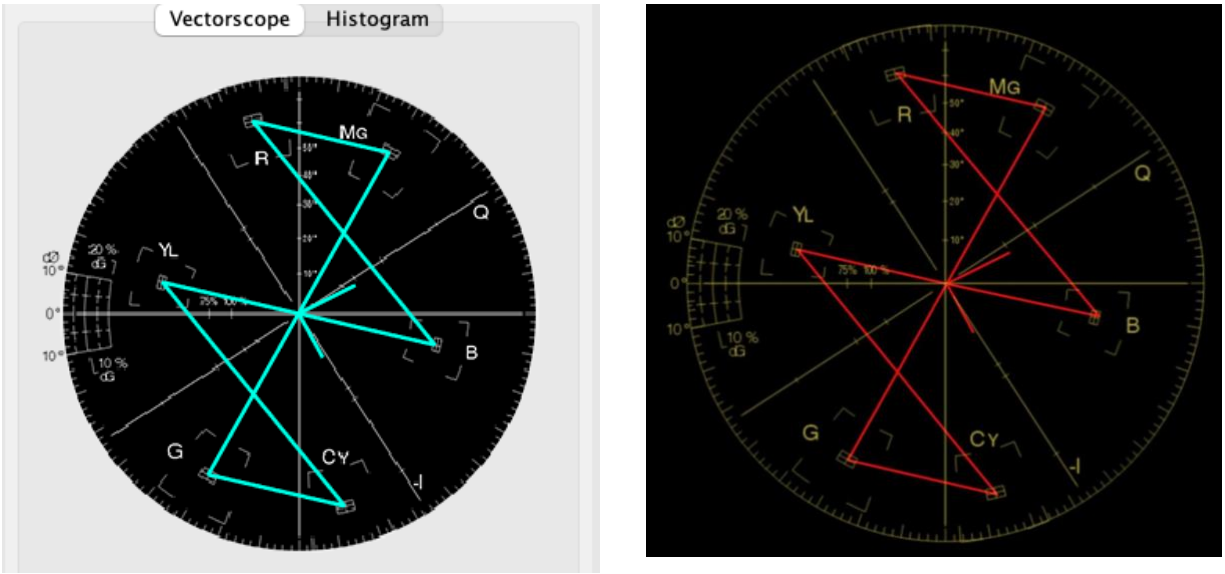

Figura 42. (a) Vectorscopio en *Matlab* para Imagen de barras. (b) Vectorscopio en *Android* para imagen de barras.

En el caso de la representación tanto en la aplicación de *Matlab* como la de *Android*, el resultado es el esperado. Se ha comprobado la imagen en el vectorscopio del programa de uso profesional *After Effects*, aunque este programa no dibuja las líneas entre los puntos, sí se puede observar cómo se señalan los puntos de cada tono puro, y si se unen, la figura que mostraría este vectorscopio de ámbito profesional sería la misma.

Si los puntos no estuvieran en el centro de cada cuadro de color, la señal de video no estaría representando con precisión los colores que se supone que debiera representar. Eso podría significar que la iluminación está apagada, que los ajustes de la cámara no se realizaron correctamente o que la cámara no funciona de forma adecuada. En cualquier caso, ajustando la cámara correctamente antes de la toma, los puntos generados por la cámara deberían situarse en los lugares adecuados dentro de la visualización del vectorscopio, tal y como están en la Figura 42.

Figura 43. Imagen de barras en vectorscopio profesional de *After Effects.*

Respecto a las líneas I y Q, son un remanente de los días del vídeo analógico. En concreto, la línea I se usa para ayudar a detectar si las señales de luminancia y crominancia (color) están sincronizadas en video analógico. Si las señales están sincronizadas, todo se alineará, pero si no lo están, no lo harán. Esto permite realizar ajustes en la señal para corregir la sincronización, de modo que los colores sean precisos.

En ambos ejemplos se observa que se encuentra alineada. Algunas aplicaciones han abandonado las líneas I y Q y simplemente muestran una línea a la que llaman línea de tono de piel (línea entre amarillo y rojo),

En las aplicaciones del proyecto, se han mantenido ambas líneas [44].

5.2. Imagen de fresas

En cuanto a la siguiente imagen de prueba, es una imagen con predominancia de tonos rojos, y algún verde, además se encuentra bastante iluminada.

La salida del vectorscopio en ambas aplicaciones, como se observa en la Figura 45 muestra prácticamente el mismo resultado..

Figura 44. Imagen *fresas.jpeg* [45].

La relación matemática que representan las líneas se distribuye en la zona de colores rojo, amarillo y verde. Las líneas en la zona del color rojo van más lejos, porque este color en la imagen presenta mayor saturación. Sin embargo, como el verde está menos saturado, las líneas en la zona de la G, verde, son más cortas.

Debido a que en la imagen también hay zonas oscuras, en el vectorscopio, se observa un punto en el centro.

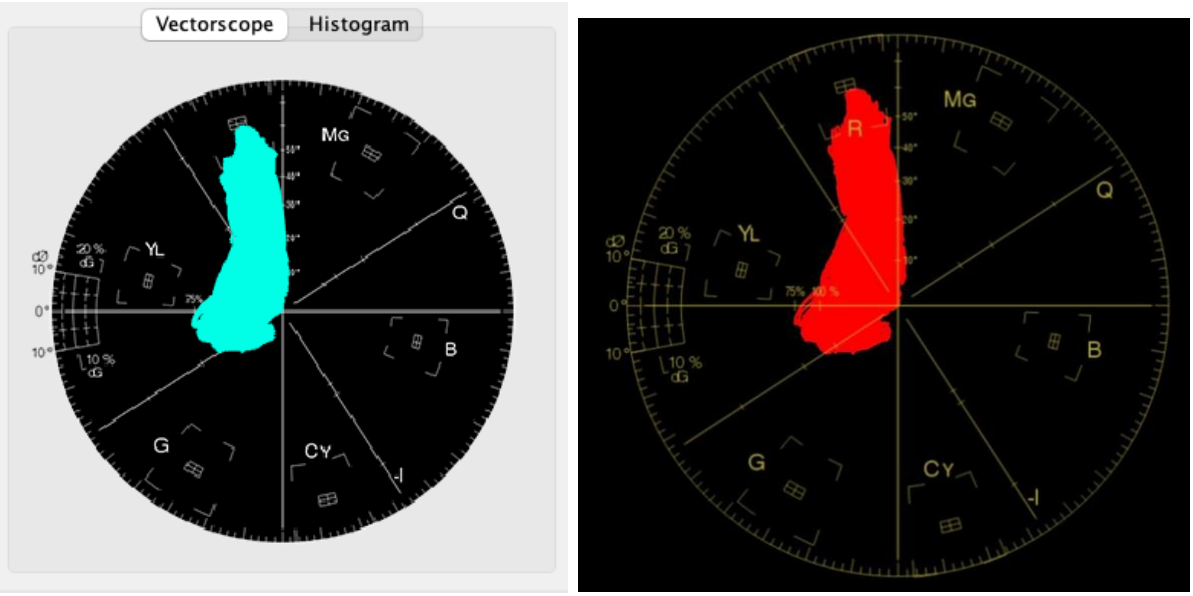

Figura 45. (a)Vectorscopio en *Matlab* para Imagen de fresas. (b) Vectorscopio en *Android* para imagen de fresas.

De nuevo se ha comprobado la salida de un vectorscopio profesional en el programa *After Effects*, como en el caso anterior se puede observar que la salida es la misma. La diferencia es que el vectorscopio profesional dibuja los puntos, mientras que los vectorscopios desarrollados en el proyecto, dibujan las líneas entre los puntos, pero se puede apreciar cómo se distribuyen los trazos exactamente por las mismas zonas.

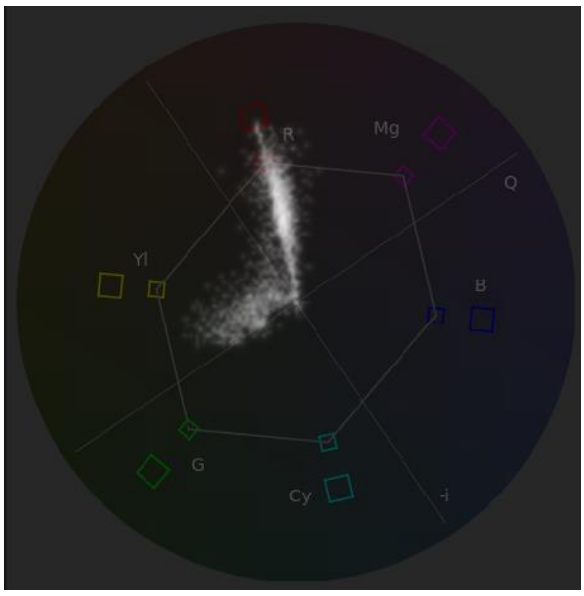

Figura 46. Vectorscopio para la imagen de fresas en el programa *After Effects*.

5.3. Imagen de tono azul

La tercera imagen de pruebas corresponde a un color puro, en este caso azul. Cada píxel de la imagen representa el color R = 0, G = 0, B = 255.

Figura 47. Imagen *azul.jpg* [46].

La visualización de la imagen en el vectorscopio es casi imperceptible. Sin embargo, si se observa con atención, en ambas ruedas se puede ver un pequeño punto en el centro del cuadrado respectivo al azul (dentro del círculo rojo).

En este caso, si la imagen no representara un tono puro, al introducir la imagen en la aplicación no saldría un solo punto, si no puntos distribuidos alrededor del cuadrado azul.

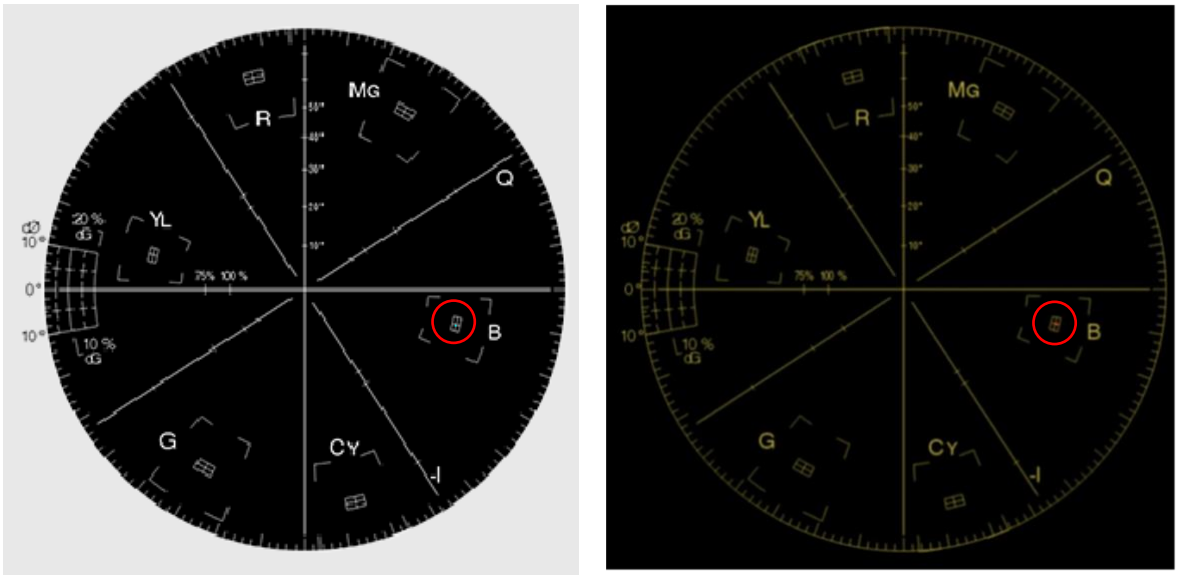

Figura 48. (a) Vectorscopio en *Matlab* para Imagen de tono azul. (b) Vectorscopio en *Android* para imagen de tono azul.

Se ha comprobado la salida de un vectorscopio profesional al probar la imagen de tono azul puro, en la siguiente imagen, en el interior del círculo rojo, se puede ver como se observa solo un punto en el cuadrado de la B, blue.

Figura 49. Vectorscopio para la imagen azul en el programa *After effects.*

5.4. Imagen en escala de grises

La siguiente imagen de prueba es una imagen en escala de grises. En este caso, los únicos colores que se ven son el blanco, el negro y el gris.

El vectorscopio es un osciloscopio para calcular la información relativa al color. Los colores blancos, negro y gris carecen de información de color, por eso tras procesar la imagen, en la rejilla, solo debería verse un punto en el centro.

Figura 50. Imagen en tono de grises, *landscape.jpg* [47].

En la Figura 51 se puede observar la salida del vectorscopio tras calcular la información de la imagen *landscape.jpg*, lo único que se ve, es un punto en el centro, con lo que el cálculo es correcto.

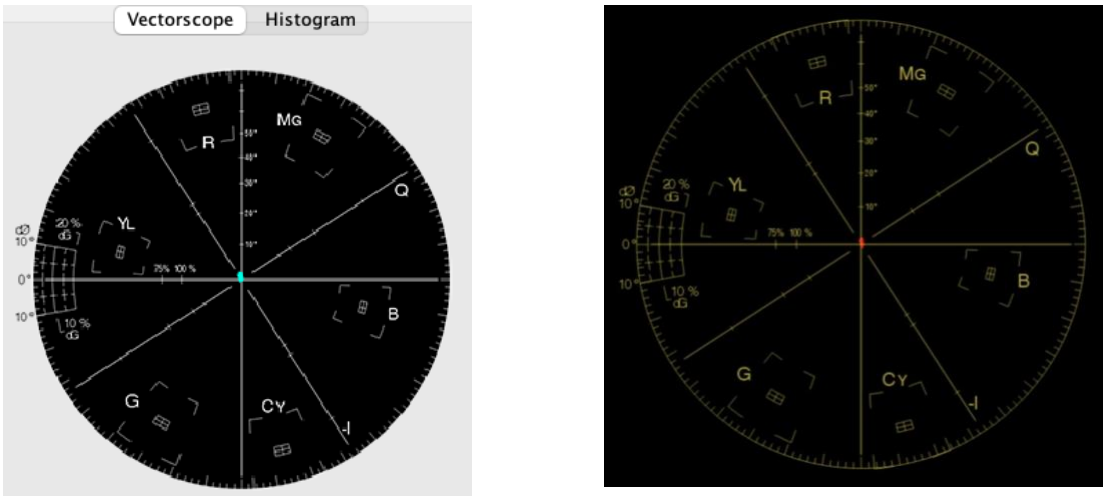

Figura 51.(a)Vectorscopio en *Matlab* para Imagen en escala de grises. (b) Vectorscopio en *Android* para imagen en escala de grises.

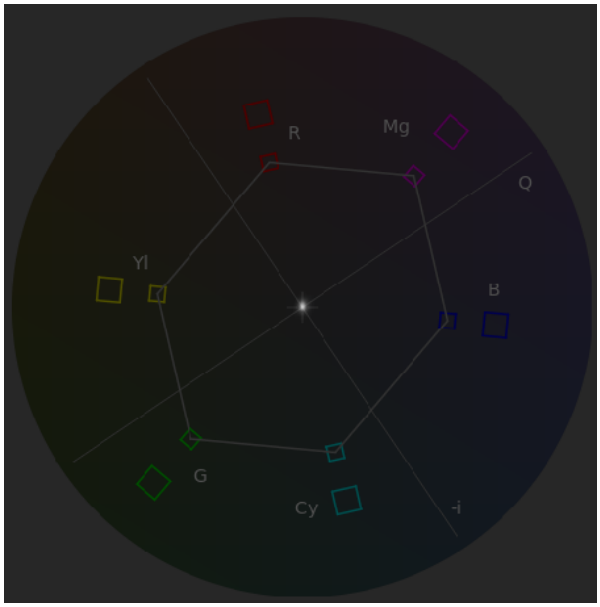

Figura 52. Vectorscopio de *After Effects* con la imagen en escala de grises.

En la Figura 52, se puede observar que el resultado en el programa *After Effects* coincide con los resultados observados en las aplicaciones de *Matlab y Android* desarrolladas.

5.5. Imagen de colores

La quinta imagen de prueba es una imagen de un festival, dónde se observan múltiples colores y mucha luz.

Figura 53. Imagen de colores. *Colors.jpg* [48].

De nuevo, como en los casos anteriores, la salida de la imagen en *Matlab* y *Android* es prácticamente idéntica. Se puede observar con detalle en la imagen de la Figura 54. En cuanto

a la distribución de colores, se puede apreciar una clara predominancia de rojos y amarillos, con una gran saturación. Respecto al resto de colores, el que menor representación presenta y con menor saturación es el Magenta. El verde, el azul y el cian tienen una gran presencia en la imagen, aunque no con tanta saturación como el amarillo y el rojo.

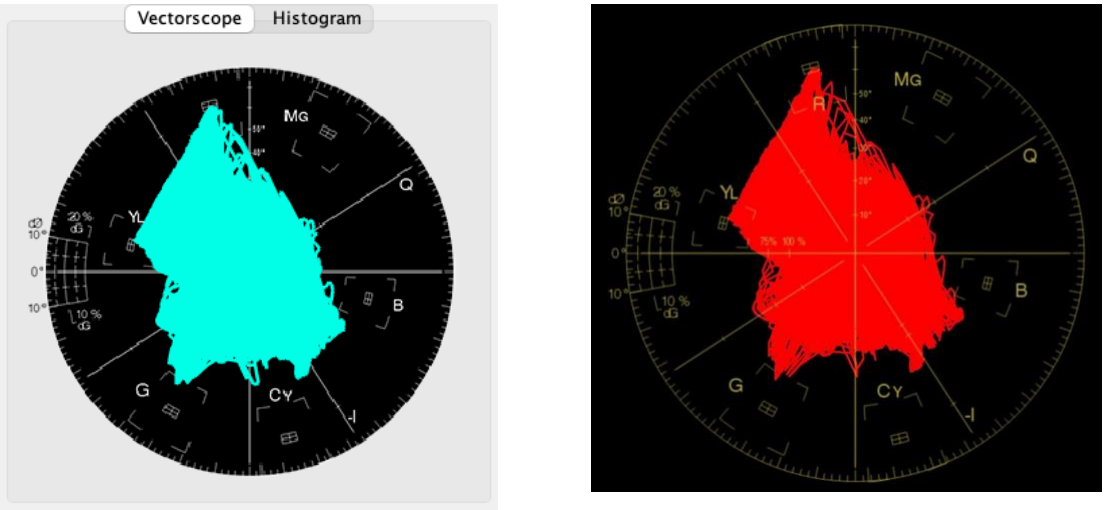

Figura 54. (a) Vectorscopio en *Matlab* para Imagen de múltiples colores. (b) Vectorscopio en *Android* para imagen de múltiples colores.

En la Figura 55, la representación en un programa profesional, el resultado de la distribución coincide con los programas creados, la diferencia radica en que *After Effects*, realiza la representación con puntos en vez de unir las líneas.

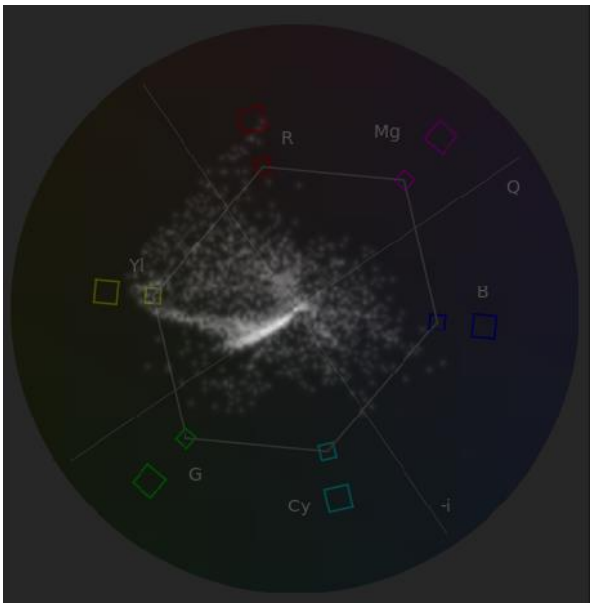

Figura 55. Vectorscopio en *After Effects* con imagen de colores.

5.6. Imagen de persona

La sexta imagen de prueba es una fotografía de un rostro de hombre con barba.

Figura 56. Imagen de hombre con barba. *Beard.jpg* [49].

La línea entre el rojo y el amarillo en el Vectorscopio es la línea de tono de piel, que te permite comprobar de forma rápida y eficaz la precisión de todos los tonos de piel de tus secuencias. Todos los tipos de piel deben pertenecer a lo largo de esta línea para que parezcan realistas.

En el caso de la imagen de estudio, se observa en ambas implementaciones como una parte de la imagen correspondiente al tono de piel se distribuye en esta línea. También se observa una parte entre el Cian y el azul debido a que estos tonos están presentes en la imagen en el jersey del hombre, sus ojos o el fondo. El resto de representación se encuentra entre el magenta y el azul, correspondiéndose con los tonos marrones oscuros de la barba y el resto del fondo [50].

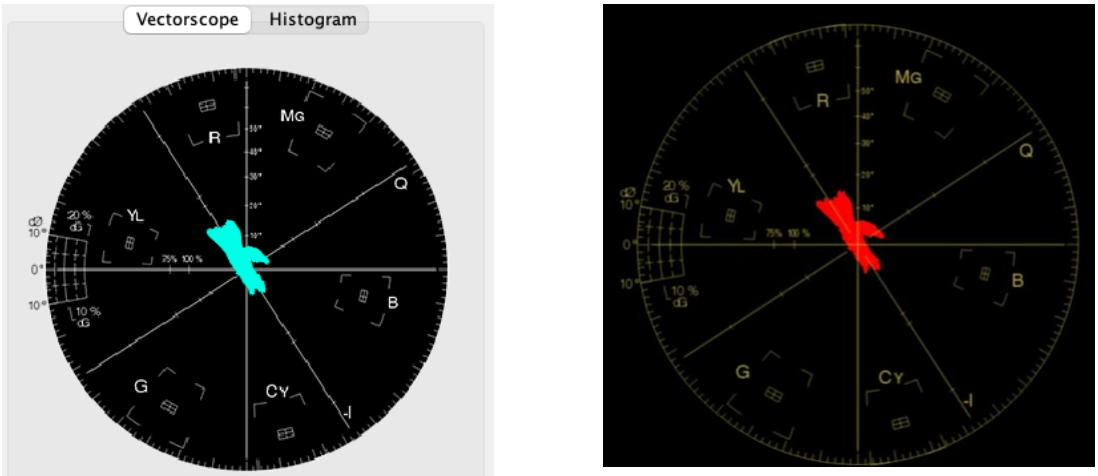

Figura 57.(a) Vectorscopio en *Matlab* para Imagen de hombre. (b) Vectorscopio en *Android* para imagen de hombre.

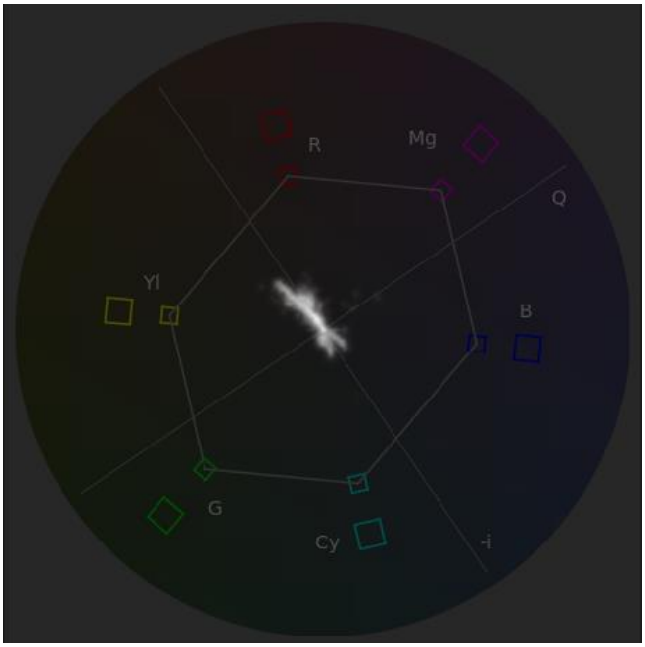

Figura 58. Vectorscopio en *After Effects* con imagen de hombre con barba.

Con esta imagen (Figura 58) se comprueba la salida en el programa *After Effects*. De nuevo el resultado coincide con pequeñas variaciones con el obtenido en los programas de creación propia. Estas variaciones se deben a como lleva a cabo la representación el programa profesional y seguramente al espacio de color, que en el caso del programa de *Adobe* es YUV, y en el de las implementaciones en *Matlab* y *Android* es YIQ.

6. Costes

Para el cálculo del coste por la elaboración de la aplicación *My Vectorscope,* es necesario tener en cuenta los medios físicos y digitales que han sido necesarios.

Para poder realizar la parte del trabajo respectivo a *Matlab* sería necesario tener la respectiva licencia que tiene un coste de 800 €.

En cuanto a medios físicos sería necesario estar en posesión de un ordenador que soporte *Matlab* y *Android Studio*. Un ordenador de gama media tiene un coste de unos 800€ en el mercado. Además, es necesario tener un smartphone *Android*, en el que lanzar las pruebas de la aplicación y poder testear que todo funcione correctamente, tanto la parte visual como la técnica. El precio de un dispositivo móvil de gama media está actualmente entre 200 y 500 euros, como no es necesario que sea el más potente del mercado, escogeremos uno de 200€.

En cuanto a la mano de obra, se necesita al menos un ingeniero/a, trabajando unas 360 horas. Teniendo en cuenta que en España la hora de ingeniería se cobra en torno a 15,38€ brutos por hora [51], el coste del trabajador se podría establecer en 30€/h. (360 horas x 30 euros/hora = 10800€) En la siguiente tabla se puede consultar el coste estimado total:

Tipo de Coste	Coste total [€]
Mano de obra	10800 €
Material	1000 €
Licencias	800 €
TOTAL	12600 €

Tabla 4. Tabla de costes.

7. Conclusiones

En este último capítulo, se describen las conclusiones que se han obtenido de este proyecto, las competencias que se han empleado y las nuevas competencias adquiridas. Además, se describen algunas posibles líneas de mejora para implementar en el futuro.

7.1. Conclusiones finales

En este TFG, se ha profundizado en los conceptos relacionados con el vectorscopio, el cuál se utiliza para medir la colorimetría de imágenes o secuencias de video, con el objetivo principal de desarrollar una aplicación *Android* que emule el comportamiento de este dispositivo.

El gran reto de este proyecto es que es novedoso en la aplicación que se ha desarrollado, ya que no se han encontrado aplicaciones que emulen un vectorscopio en el *Marketplace* de *Android,* lo cual aporta valor.

Para abordar el desarrollo final en *Android,* se realizó en primer lugar una versión de vectorscopio utilizando la herramienta *Matlab*, ya que es muy adecuado en el tratamiento de imágenes con el buen comportamiento de esta herramienta en cálculos matriciales.

Por último, se ha comprobado el funcionamiento de la herramienta desarrollada en *Android* realizando diversas pruebas con distintas imágenes que presentan diferentes distribuciones de colores, comparando los resultados con la herramienta desarrollada previamente en *Matlab*, así como con una herramienta profesional *(Adobe After Effects*), obteniendo unos resultados similares, y por lo tanto satisfactorios.

7.2. Competencias empleadas

Esta memoria recoge toda la información relevante recabada a la hora de realizar el Trabajo de fin de grado, es por ello por lo que muchos conocimientos que se habían adquirido durante los años de estudio han sido empleados.

Uno de los conocimientos empleados más importantes, sin el cual este proyecto no se podría haberse llevado a cabo es la programación. Durante los años de carrera, son varias las asignaturas en las que se aprende a programar: Informática I, Informática II, Protocolos para la Transmisión de Audio y Vídeo en Internet, Construcción de Servicios y Aplicaciones Audiovisuales en Internet, Laboratorio de Tecnologías Audiovisuales en la Web, Arquitectura de Sistemas Audiovisuales II, Gráficos y Visualización en 3D etc.

En el desarrollo de este proyecto, el conocimiento de saber programar, saber cómo enfrentarse a los problemas, buscar otros caminos para realizar el trabajo, etc., ha sido de vital importancia.

Así mismo, tras el estudio de otras asignaturas, como Estadística para Sistemas Audiovisuales, Sistemas Lineales, Circuitos y Sistemas, Fundamentos de las Comunicaciones, Campos y Ondas, Gestión y Optimización de Recursos, Estándares de Comunicación de Audio y Vídeo, Equipos y Sistemas de Audio y Vídeo, Tratamiento Digital de la Imagen, Difusión de Audio y Video … Se tenía conocimiento programación en MATLAB.

Otras competencias relevantes que han sido aplicadas en el proyecto son las adquiridas del uso de las funciones y manejo del vectorscopio, en la asignatura de Equipos y Sistemas de Audio y Vídeo.

También se ha aplicado el conocimiento de tratamiento de imágenes, a nivel técnico, así como su estudio, gracias a las asignaturas de Tratamiento Digital de la Imagen, Estándares de Comunicación de Audio y Vídeo, Difusión de Audio y Video y Equipos y Sistemas de Audio y Vídeo.

Por último, otro conocimiento aplicado ha sido el de la asignatura de Idioma Moderno, es decir, comprender textos en inglés y saber interpretar. Casi toda la información encontrada y consultada, sobre usos del vectorscopio, así como la información relativa a programación se encuentra en inglés, por lo que sin esta competencia habría sido imposible tener éxito en la realización del trabajo.

7.3. Competencias adquiridas

Tras la realización del proyecto se han adquirido algunas competencias, como consecuencia del desarrollo del mismo.

Lo primero de todo, el aprendizaje de programar en *Java*, que es un lenguaje muy importante y que presenta cierta complejidad en la tarea que se ha desempeñado.

Ligado a esta competencia, se encuentra la capacidad de poder realizar aplicaciones en *Android*, antes de la fase de investigación de este proyecto, no se tenía ningún conocimiento de la forma en que se programaban las aplicaciones en Android.

Otra competencia adquirida, ha sido la de aumentar la capacidad en buscar información seleccionando lo más relevante, así como poder transformar y acoplar esa información en el proyecto. Esto, sobre todo, se aplica a la búsqueda de soluciones en programación.

Hay otras competencias que no se han adquirido desde cero, pero sí se han mejorado tras el trabajo realizado. Se ha profundizado muchísimo en el conocimiento del mundo audiovisual, de las herramientas que se utilizan, y cómo se utilizan.

7.4. Trabajos futuros

En líneas de mejora para el futuro de la aplicación se van a comentar algunas ideas.

- **Mejorar la visualización para diferentes dispositivos**. Hay que tener en cuenta que la actual aplicación se ha probado en un dispositivo, pero para profesionalizar la aplicación sería interesante probar en múltiples tamaños de pantalla para que fuera extensiva a diferentes dispositivos.
- **Cargar fotos tomadas en tiempo real,** añadir al botón *Load* la posibilidad de tomar una foto en el momento, y que se calcule la relación de vectorscopio de esa foto.
- **Modificar la fuente de datos** que alimenta al vectorscopio para que sea un **vídeo** en vez de una foto fija. Aunque en términos de estudio, el vectorscopio analiza fotograma a fotograma, podría ser interesante, tener la capacidad de estudiar un vídeo e ir moviéndose por los diferentes *frames* en vez de tener que cargar cada fotograma por separado.
- Seguir implementando en la misma aplicación **otros mecanismos de estudio de las imágenes y vídeos**, como por ejemplo tener otras pantallas en la aplicación para ver la forma de onda de la imagen, la distribución de los canales de color, o el histograma.
- **Integrar el vectorscopio en una aplicación de edición de vídeo**. Esta línea de mejora quizás es algo ambiciosa, pero si existiera una aplicación *Android* en el mercado, añadir esta herramienta sería un valor añadido.

8. Bibliografía

[1] Pixeles, «Ionos Website», 2023 [En línea]. Available: https://www.ionos.es/digitalguide/paginas-web/diseno-web/que-es-un-pixel/

[2] Uniqtech. (2019). Distribución RGB en un píxel. [Figura]. Recuperado de: https://miro.medium.com/v2/resize:fit:640/format:webp/1*KzSu3ytpLj8Dl0v2qVC1zw.png

[3] Fernández, F. (2023). Separación de valores de luma y Croma de un píxel. [Figura]. Recuperado de: https://quecamarareflex.com/wp-content/uploads/2016/11/submuestreo_croma_chroma_subsampling.png

[4] Clipstudio. (2018). Espacio de color aditivo, RGB. [Figura]. Recuperado de: https://www.clipstudio.net/wp-content/uploads/2020/01/0045_001_es-es.jpg

[5] Hmong. (2019). El espacio de color YIQ en Y = 0.5. [Figura]. Recuperado de: https://hmong.es/wiki/YIQ

[6] Color spaces, «Es-academic Website,» 2023 [En línea]. Available: https://es-academic.com/dic.nsf/eswiki/1233244

[7] Pixabay. (2020). Carta de barras de color SMPTE. [Figura]. Recuperado de: https://cdn.pixabay.com/photo/2020/11/30/18/14/smpte-color-bars-5791787_1280.png

[8] Tilanov. (2015). Tabla mira UER. [Tabla]. Recuperado de: https://tilanotv.es/todo-lo-que-deberias-saber-sobre-las-barras-de-color-y-sus-usos/

[9] Tilanov. (2021). Carta de barras UER. [Figura]. Recuperado de: https://tilanotv.es/wp-content/uploads/2021/02/figura-4.3.png

[10] Andreas Ulrich Maute, «Professional Content Management Systems - Handling Digital Media Assets», 2004, John Wiley & Sons Inc.

[11] Rio, J. (2014). Carta de barras ARIB. [Figura]. Recuperado de: https://tecnicaparaaudiovisuales.files.wordpress.com/2014/12/arib-calibrated.jpg

[12] Robert L Hartwig, «Basic TV Technology: Digital and Analog», 2012, Routledge

[13] Jack, Keith, «NTSC, PAL, and SECAM Overview», 10.1016/B978-075067822-3/50009-9, 2005

[14] FotoNostra. (2019). Mapa mundial de distribución de sistemas de codificación. [Figura]. Recuperado de: https://www.fotonostra.com/digital/fotos/sistemascod.jpg

[15] Mavis. (2018). Rueda cromática. [Figura]. Recuperado de: https://support.shootmavis.com/hc/en-us/articles/212451065-Vectorscope-

[16] Blain Brown, «Cinematography Theory and Practice. Image making for cinematographers and directors, » 2016, Routledge.

[17] James. (2017). Histograma. [Figura]. Recuperado de: https://i0.wp.com/thevideoproguys.com/wpcontent/uploads/2017/07/LumaWaveform.jpg?resize=1024%2C279&ssl=1

[18] Borys Golik. (2010). «Software Interface for Video Image Quality Analysis» Recuperado de: https://www.image-engineering.de/content/library/diploma_thesis/borys_golik_software_interface.pdf

[19] Steve Hullfish y Jamie Fowler, « Color Correction for video», 2008, Focal Press

[20] Steve Hullfish, «The Art and Technique of Digital Color Correction», 2008, Focal Press

[21] Fabara, S. (2018). Carta de colores de tonos de piel en Davinci Resolve. [Figura]. Recuperado de: https://i.blogs.es/be182d/pieles002/1366_2000.webp

[22] Scott, David. (2008). Histograms: Theory and Practice. 10.1002/9780470316849.ch3.

[23] Charlotte & Peter Fiell, «A Guide to MATLAB for Beginners and Experienced Users MATLAB», 2001, Cambridge University Press

[24] Penalva, J. (2008). Dispositivo con sistema Android antiguo. [Figura]. Recuperado de: https://i.blogs.es/7bfa9b/google-g1/1366_2000.jpg

[25] Android. (2021). Dispositivo con sistema Android actual. [Figura]. Recuperado de: https://www.movilzona.es/app/uploads-movilzona.es/2021/11/Fondos-de-pantalla-Android-12.jpg

[26] Android, «Android Website», 2023 [En línea]. Available: https://www.android.com/what-is-android/

[27] Android. (2023). Tabla de versiones Android. [Tabla]. Recuperado de: https://andro4all.com/android/versiones-android-historia

[28] Kotlin, «Kotlin Website», 2023 [En línea]. Available: https://kotlinlang.org/

[29] John Horton, «Android preparing for beginners», 2015, Pack Publishing

[30] Java, «Blog Centro de Elearning Website», 2023 [En línea]. Available: https://blog.centrodeelearning.com/2021/08/13/java-todo-lo-que-hay-que-saber-sobre-uno-de-los-lenguajes-de-programacion-mas-utilizados/

[31] Java, «Java Website», 2023 [En Línea]. Available: https://www.java.com/es/download/help/whatis_java.html

[32] Extensible Markup Language, «W3 Website», 2023 [En Línea]. Available: https://www.w3.org/standards/xml/core

[33] Android. (2020). Diseño de restricciones. [Figura]. Recuperado de: https://developer.android.com/static/codelabs/basic-android-kotlin-training-xml-layouts/img/8dea708333aebabe_856.png?hl=es-419

[34] Adobe After effects, «Adobe Website», 2023 [En Línea]. Available: https://www.adobe.com/es/products/aftereffects.html

[35] MathWorks, «MATLAB Website», 2022. [En línea]. Available: https://es.mathworks.com/help/matlab/.

[36] Rafael C. Gonzalez, Richard E. Woods y Steven L. Eddins, «Digital Image Processing Using MATLAB», 2003, Pearson Prentice Hall.

[37] Jerome DiMarzio, «Beginning Android Programming with Android Studio», 2016, Wrox

[38] Organización de XML en Android, «Academia Android Website», 2022. [En Línea]. Available: https://academiaandroid.com/tratamiento-de-xml-en-android-introduccion/

[39] Imgur. (2022). Diseño de restricciones. [Figura]. Recuperado de: https://i.stack.imgur.com/R22jh.png

[40] Jef Friesen y Peter Späth, «Learn Java for Android development», 2020, Apress.

[41] Programming with Android Java «Developer Android Website», 2023. [En Línea]. Available: https://developer.android.com/docs

[42] Iván Guerrero Vaquerizo, «Sistemas de producción audiovisual», 2017, Paraninfo

[43] Detalle del vectorscopio https://tilanotv.es/wp-content/uploads/2021/02/Figura-12.16-983x1024.png

[44] El vectorscopio y las barras de color «Jpereira Website», 2023, [En Línea]. Available: http://www.jpereira.net/gestion-de-color-articulos/cartas-de-color-en-video-digital

[45] iStock. (2022). Imagen de fresas. [Figura]. Recuperado de: https://www.finedininglovers.com/es/sites/g/files/xknfdk1706/files/styles/article_1200_800_fallback/public/2022-04/fresas%C2%A9iStock.jpg?itok=iBcd_HLd

[46] Pngtree. (2021). Imagen de tono azul puro. [Figura]. Recuperado de: https://png.pngtree.com/background/20210715/original/pngtree-blue-pure-color-simple-background-picture-image_1323911.jpg

[47] Hudson, M. (2022). Imagen de paisaje en escala de grises. [Figura]. Recuperado de: https://michael-hudson.com/wp-content/uploads/2022/08/landscape-gaaf71654b_640.jpg

[48] Adventure Volunteer. (2018). Imagen de fiesta de colores. [Figura]. Recuperado de: https://www.adventurevolunteer.org/wp-content/uploads/2017/07/la-india-holi.jpg

[49] Aprendum. (2020). Imagen de hombre con barba. [Figura]. Recuperado de: https://d2qc4bb64nav1a.cloudfront.net/cdn/13/images/curso-online-de-como-leer-el-rostro-en-las-personas:-los-microgestos_l_primaria_1_1561117635.jpg

[50] Jorge Carrasco González, «Cine y televisión digital. Manual técnico», 2010, Publicacions i Edicions de la Universitat de Barcelona.

[51] Sueldo Ingeniero en España «Talent Website», 2023. [En Línea] Available: https://es.talent.com/salary?job=ingeniero

Anexos

A.1. Instalación y Uso

Para **la instalación de la aplicación *Matlab***, es necesario tener el programa *Matlab* en el ordenador, se puede clonar la carpeta desde el siguiente repositorio: https://github.com/Elenadr/Matlab_Vectorscope

A continuación, solo sería necesario ejecutar desde *Matlab* el archivo *my_vectorscope.m*. En caso de querer probar con imágenes distintas, habría que editar la línea 45 del código: *RGB = imread('bars.tif'*), sustituyendo bars.tif, con el nombre y la extensión de la imagen a probar, por ejemplo *fresas.jpeg, blue.jpg, landscape.jpg*…

La **instalación de la aplicación *Android*** se puede realizar descargando la apk (*Android Application Package*), desde su ubicación:

https://github.com/Elenadr/MyVectorscope/blob/master/my_vectorscope.apk

En el dispositivo móvil será necesario tener activada la opción de permitir Instalar aplicaciones de orígenes desconocidos. Después se ejecuta la instalación que debería ser relativamente rápida y ya se podría usar la aplicación. Hay que tener en cuenta que la aplicación no es *responsive*, por lo que puede que no se vea correctamente en todos los dispositivos.

A.2. Publicación

El código fuente de la aplicación desarrollada en *Matlab* se encuentra en: https://github.com/Elenadr/Matlab_Vectorscope

El código fuente de la aplicación desarrollada en *Android* se encuentra en: https://github.com/Elenadr/MyVectorscope Se puede clonar el repositorio y abrir el proyecto en *Android Studio* para consultar todas las carpetas y hacer modificaciones si se desean.

La presente memoria se encuentra disponible en: https://github.com/Elenadr/Memoria_TFG