**Instituto Politécnico Nacional  
 Escuela Superior de Cómputo  
Ingeniería en   
Sistemas Computacionales**

**Unidad de Aprendizaje:**

**VISIÓN POR COMPUTADORA**

**Grupo: 7CV6**

**Práctica 2:**

**Transformaciones**

**Alumno:**

**Bautista Ríos Alfredo**

**Docente: Saul De la O Torres**

**Fecha de entrega:**

**15/09/2025**

## 

## 

[**Introducción 3**](#_xdm2z8qgplq5)

[Objetivos 4](#_ecstldi6c2vp)

[Objetivo general 4](#_x8kecqvvy6u0)

[Objetivos específicos 4](#_r0nhih89kryn)

[Marco teórico 5](#_4p12caj33ip0)

[Representación digital del color 5](#_r6iea26ce3ge)

[Modelos de color y fórmulas 5](#_2v0ugez4w9jg)

[Metodología y Desarrollo 12](#_u9vsl75eu8g7)

[Arquitectura (C++/GTK+3) 12](#_o769ki4l05t6)

[Flujo Origen → Intermedios → Destino 13](#_q92sspb9p843)

[Modos de visualización por canal 13](#_g75egqo4qjtg)

[Implementación de transformaciones (puntos clave) 13](#_aiv24xisl6gt)

[Resultados y validación 14](#_t8mdsopdq367)

[Pruebas de consistencia 14](#_revzl8vqxphd)

[Evidencia de salida (descripción de pantallas) 15](#_5t5ivg6yare7)

[Discusión 16](#_w5xkgvompjaz)

[Conclusión 16](#_x5mj9a9avpor)

[Referencias 17](#_use78gmhzxwc)

[Apéndice A — Detalles de implementación (resumen) 18](#_g9jea84pterh)

# 

## **Introducción**

El manejo riguroso del color constituye un eje transversal en visión por computadora y en varias etapas de los flujos de trabajo de aprendizaje automático sobre imágenes. Antes de segmentar, clasificar, medir o entrenar modelos, es indispensable comprender **cómo se representa el color en memoria**, cómo se **descompone en canales** y cómo se **transforma** entre modelos de color adecuados a cada tarea. En este proyecto se desarrolla una aplicación en **C++ con GTK+3** que permite convertir imágenes entre los modelos **RGB, CMY, CMYK, YIQ, HSI y HSV**, mostrando el **pipeline completo** de cada transformación con una lógica clara: **Origen → Intermedios → Destino**. El punto de partida es la representación clásica de 8 bits por canal (**0x00RRGGBB**), desde la cual se extraen los valores **R**, **G** y **B** (0–255), se normalizan a **[0,1]** para aplicar las fórmulas, y finalmente se visualizan otra vez como píxeles de 8 bits para facilitar su interpretación.

La relevancia de cubrir múltiples espacios de color no es meramente teórica: cada modelo resuelve necesidades distintas. **RGB** es natural para dispositivos emisores de luz y para gráficos por computadora, mientras que **CMY/CMYK** corresponde al paradigma sustractivo propio de impresión, donde el canal **K** (negro) mejora contraste y economía de tinta. **YIQ** surge de la televisión analógica NTSC: **Y** (luminancia) separa la información de brillo de la crominancia (**I**, **Q**), lo que permitía compatibilidad con receptores monocromos. **HSI/HSV**, por su parte, acercan la descripción del color a la percepción humana: el **tono (H)** expresa la “familia” de color, la **saturación (S)** su pureza, y la **intensidad/valor (I/V)** su luminosidad. Esta diversidad hace que, según el objetivo (segmentar por color, realzar sombras, preparar archivos para impresión o analizar luminancia), sea más conveniente operar en un espacio u otro.

Para sustentar la comprensión y la trazabilidad, la interfaz se organiza en dos paneles. En el **panel izquierdo**, el usuario carga la imagen, observa una vista previa, selecciona la **transformación** y el **modo de visualización por canal** (**Gris**, **Tintado** o **Pseudocolor**), y consulta el apartado **Acciones / Cálculos**, donde se despliegan las ecuaciones y pasos numéricos fundamentales (por ejemplo, el cálculo de K=min⁡(C,M,Y)K=\min(C,M,Y) en CMYK, o de Cmax⁡,Cmin⁡C\_{\max}, C\_{\min} y Δ\Delta en HSV). En el **panel derecho** se muestran de forma ordenada los **canales** del modelo fuente, los **intermedios** que explican el proceso y los canales del modelo destino, cada uno con su **pie de imagen**. El área de resultados utiliza un **grid adaptativo** que mantiene miniaturas uniformes sin deformación, ajustando el número de columnas al ancho disponible.

Desde el punto de vista de ingeniería, la aplicación adopta una **arquitectura ligera tipo MVC**. El **modelo** implementa las funciones puras de conversión (código sin dependencias de UI, operando sobre arreglos normalizados en [0,1]); la **vista/control** se encarga de la construcción de widgets, el ruteo de señales y el rendering de cada canal; y un módulo de **utilidades** resuelve la conversión **Pixbuf ↔ planos** y el renderizado en distintos modos. Este desacoplamiento simplifica la validación (por ejemplo, ejecutar “round-trips” como RGB→HSV→RGB) y facilita la extensión futura (nuevos modelos, filtros o mapas de pseudocolor).

La **visualización por canal** es deliberadamente flexible. El modo **Gris** cuantifica intensidades; **Tintado** preserva la semántica del canal (R en rojo, G en verde, B en azul; C en cian, M en magenta, Y en amarillo; K en gris), lo que ayuda a “leer” rápidamente el aporte de cada plano; y **Pseudocolor** (azul→cian→verde→amarillo→rojo) resalta gradientes sutiles, útil para analizar variaciones locales en **H**, **S**, **I/V** o **K**. En particular, **Hue (H)** se colorea por defecto usando **HSV con S=V=1**, ya que resulta más informativo que verlo en gris, aunque el usuario puede conmutar a pseudocolor si así lo prefiere.

En el plano numérico, se cuidaron aspectos clave: **normalización** de entrada a [0,1], **clamping** tras inversiones potencialmente fuera de rango (p. ej., YIQ→RGB), manejo de **casos límite** (Δ=0\Delta=0 en HSV, I=0I=0 en HSI) y uso de **épsilon** en denominadores para evitar inestabilidades (p. ej., cos⁡(60∘−H)\cos(60^\circ-H) en HSI). Asimismo, se incluyeron **intermedios** visibles que hacen “auditables” las rutas (p. ej., Cmax⁡,Cmin⁡,ΔC\_{\max}, C\_{\min}, \Delta en HSV o min⁡(R,G,B)\min(R,G,B) en HSI), fortaleciendo el vínculo entre la ecuación y su efecto perceptual.

En suma, el proyecto busca ser un **puente** entre la matemática de los modelos de color y su **impacto práctico** en análisis de imágenes y ML. La división **Origen → Intermedios → Destino**, los **modos de visualización**, y el acento en **trazabilidad** y **estabilidad numérica** permiten que el estudiante comprenda no solo **qué** hace una conversión, sino **por qué** se comporta así y **cuándo** conviene usarla en problemas reales.

## 

## **Objetivos**

### **Objetivo general**

Desarrollar una aplicación en **C++/GTK+3** que **cargue** una imagen en RGB (8 bits por canal) y **ejecute transformaciones** entre modelos de color **(RGB↔CMY↔CMYK↔YIQ↔HSI↔HSV)**, **desplegando** de forma ordenada **todos los canales** y **pasos intermedios** (Origen → Intermedios → Destino), con **modos de visualización** por canal (**Gris**, **Tintado**, **Pseudocolor**) y **panel de acciones/cálculos**.

### **Objetivos específicos**

* Implementar **extracción de canales** bajo la convención **0x00RRGGBB** (0–255) y normalización a **[0,1]**.
* Programar las **fórmulas** de conversión **RGB↔CMY**, **CMY↔CMYK**, **RGB↔YIQ**, **RGB↔HSI** y **RGB↔HSV**, cuidando **clamping** y rangos.
* Diseñar UI con **panel izquierdo** (cargar imagen, nombre de archivo, selector de transformación y **Acciones/Cálculos**) y **panel derecho** (secciones **Origen**, **Intermedios**, **Destino**).
* Producir **miniaturas** de tamaño uniforme/adaptativo por canal con **pies de imagen**.
* Validar correctamente con **pruebas de consistencia** (round-trips y casos límite).

## **Marco teórico**

### **Representación digital del color**

* Un pixel color típico en memoria se representa como un **entero de 32 bits**: 0x00RRGGBB, donde cada componente ocupa 8 bits.
* Extracción por bitmask/corrimiento:  
  + R = (pix & 0x00ff0000) >> 16
  + G = (pix & 0x0000ff00) >> 8
  + B = (pix & 0x000000ff)
* Normalización: r = R/255, g = G/255, b = B/255 ∈ **[0,1]**.  
   Este principio se trabajó explícitamente en la práctica previa.

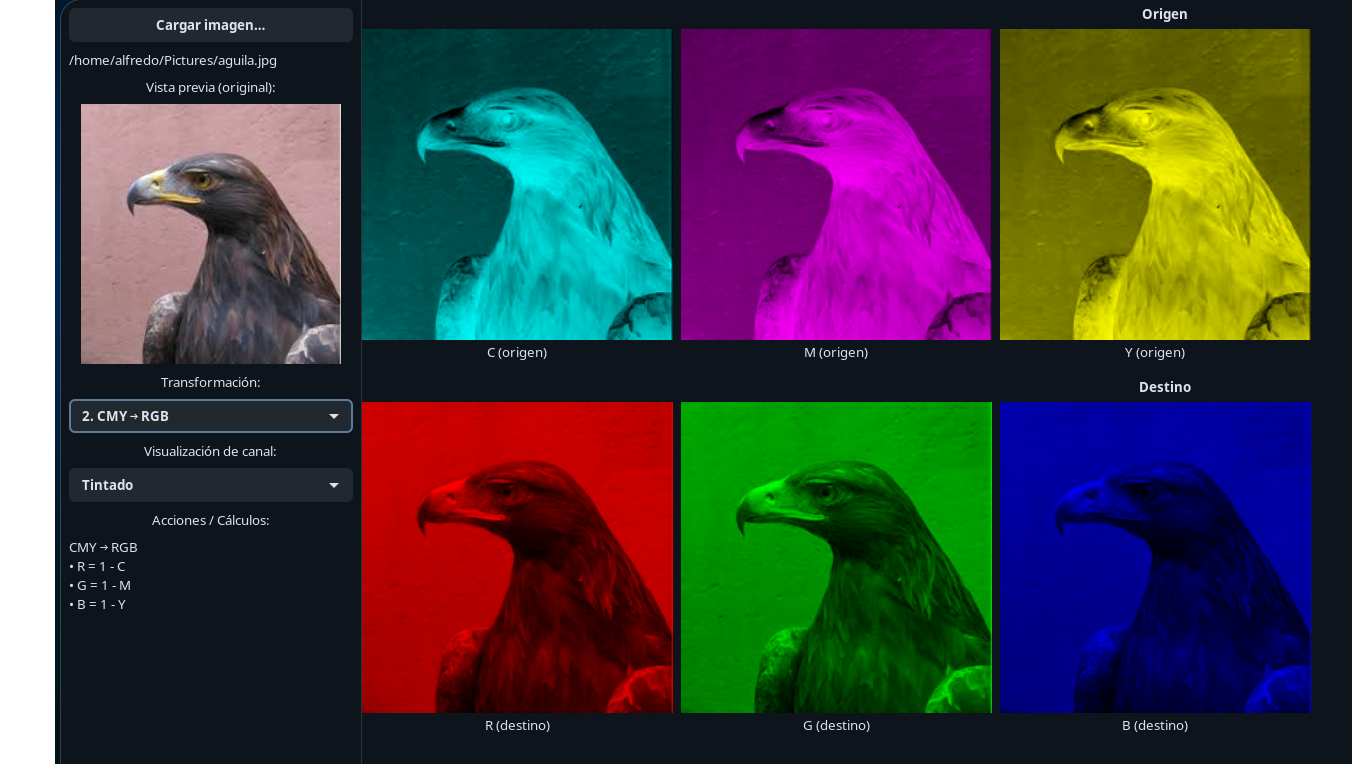
### **Modelos de color y fórmulas**

**RGB ↔ CMY**

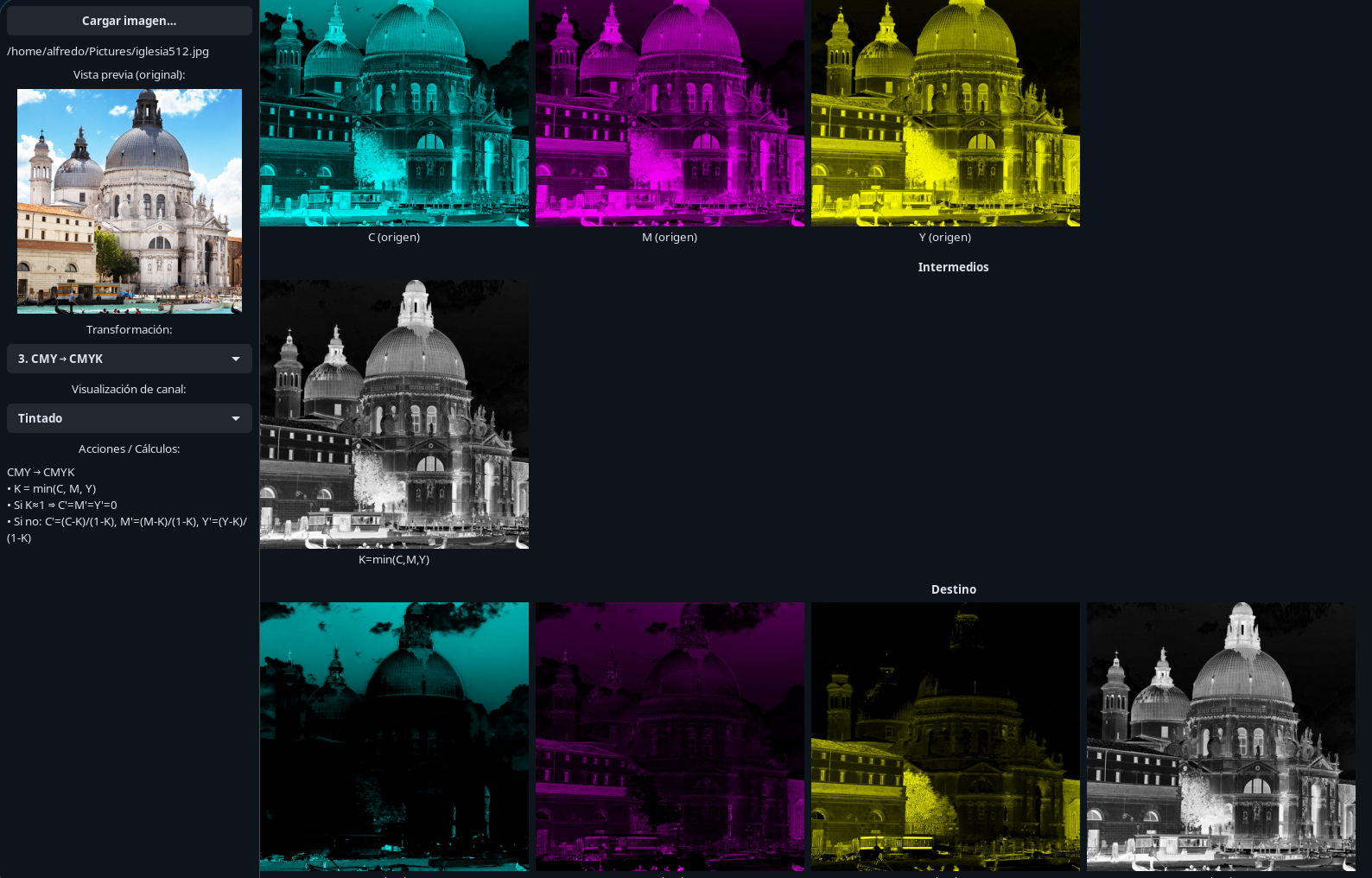
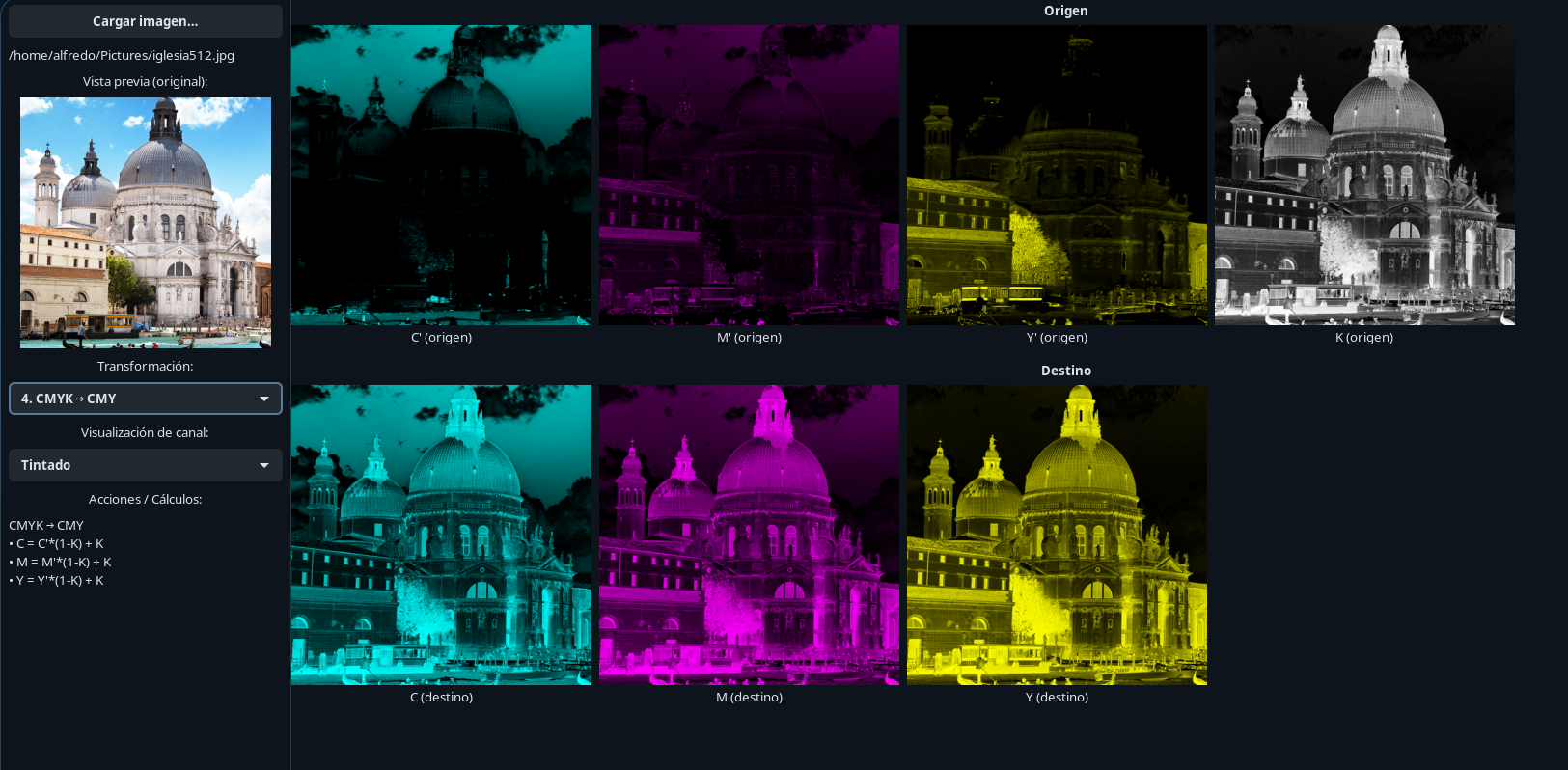
* RGB → CMY: C=1−R, M=1−G, Y=1−B

****

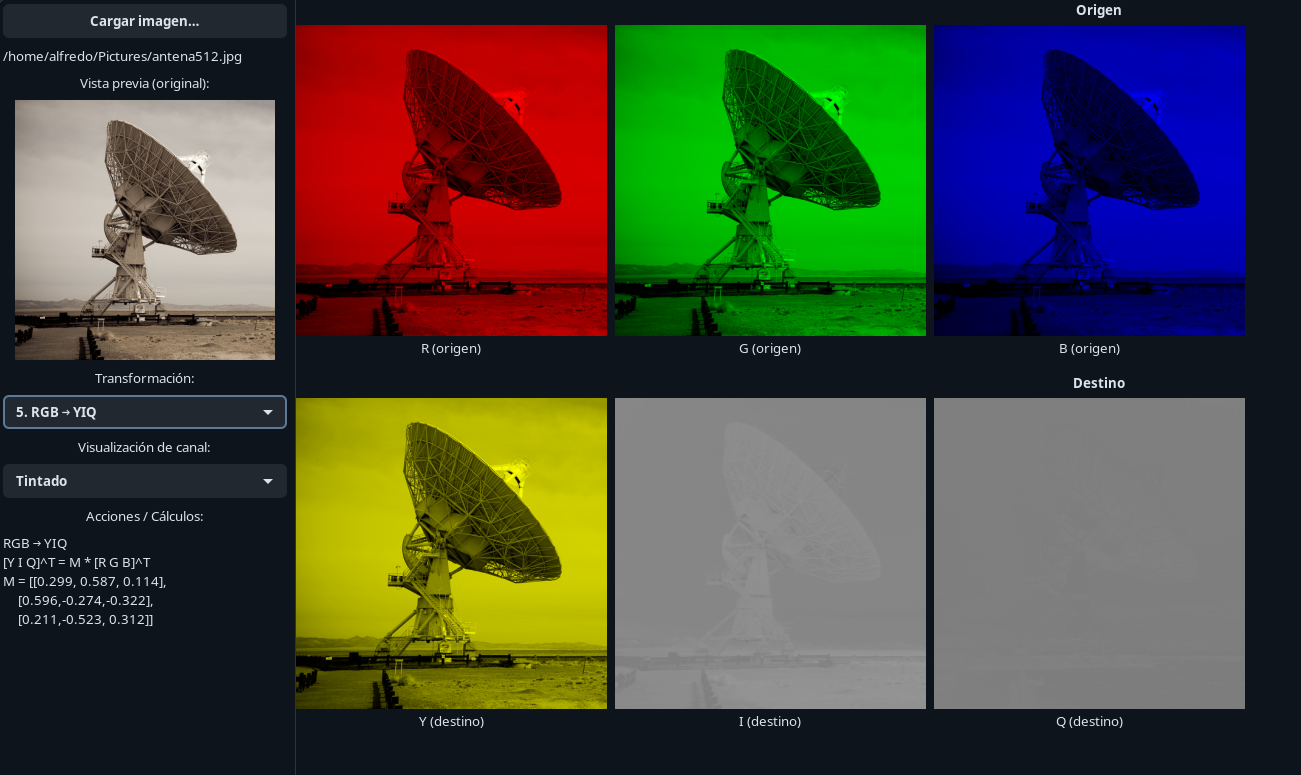
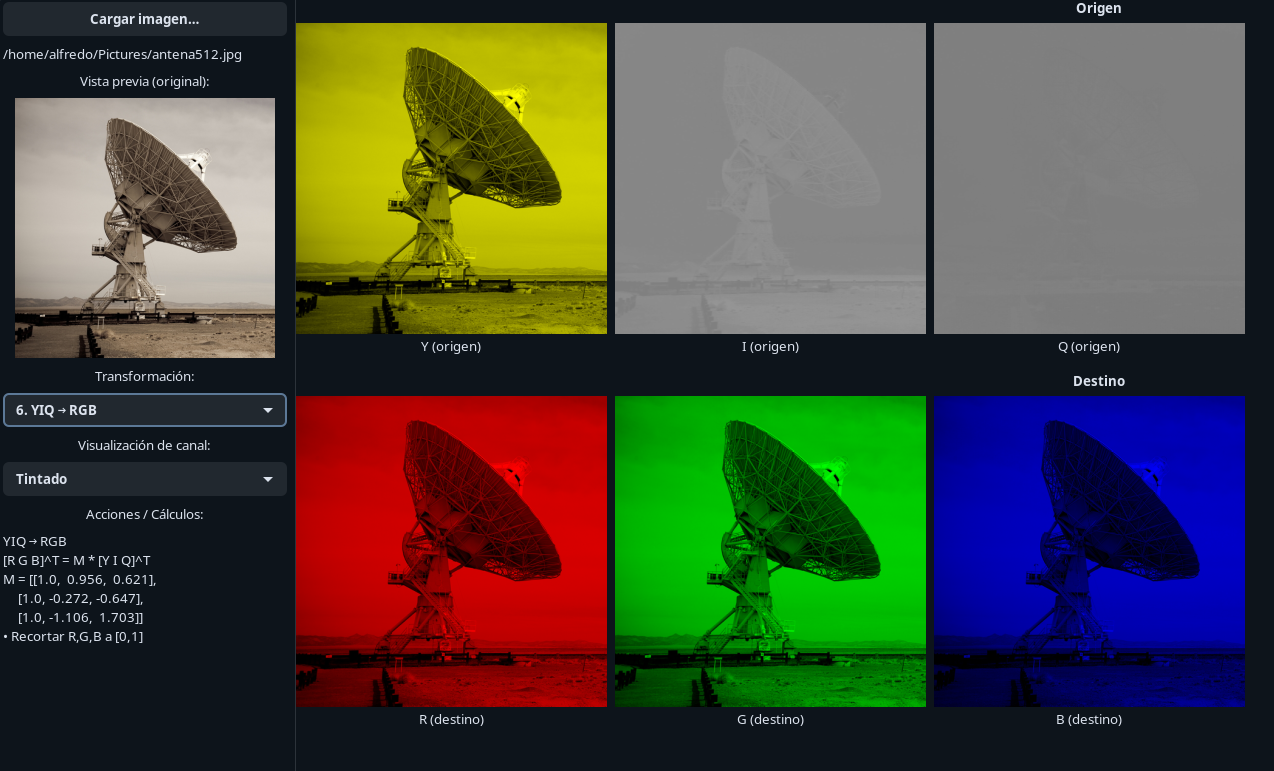
* CMY → RGB: R=1−C, G=1−M, B=1−Y



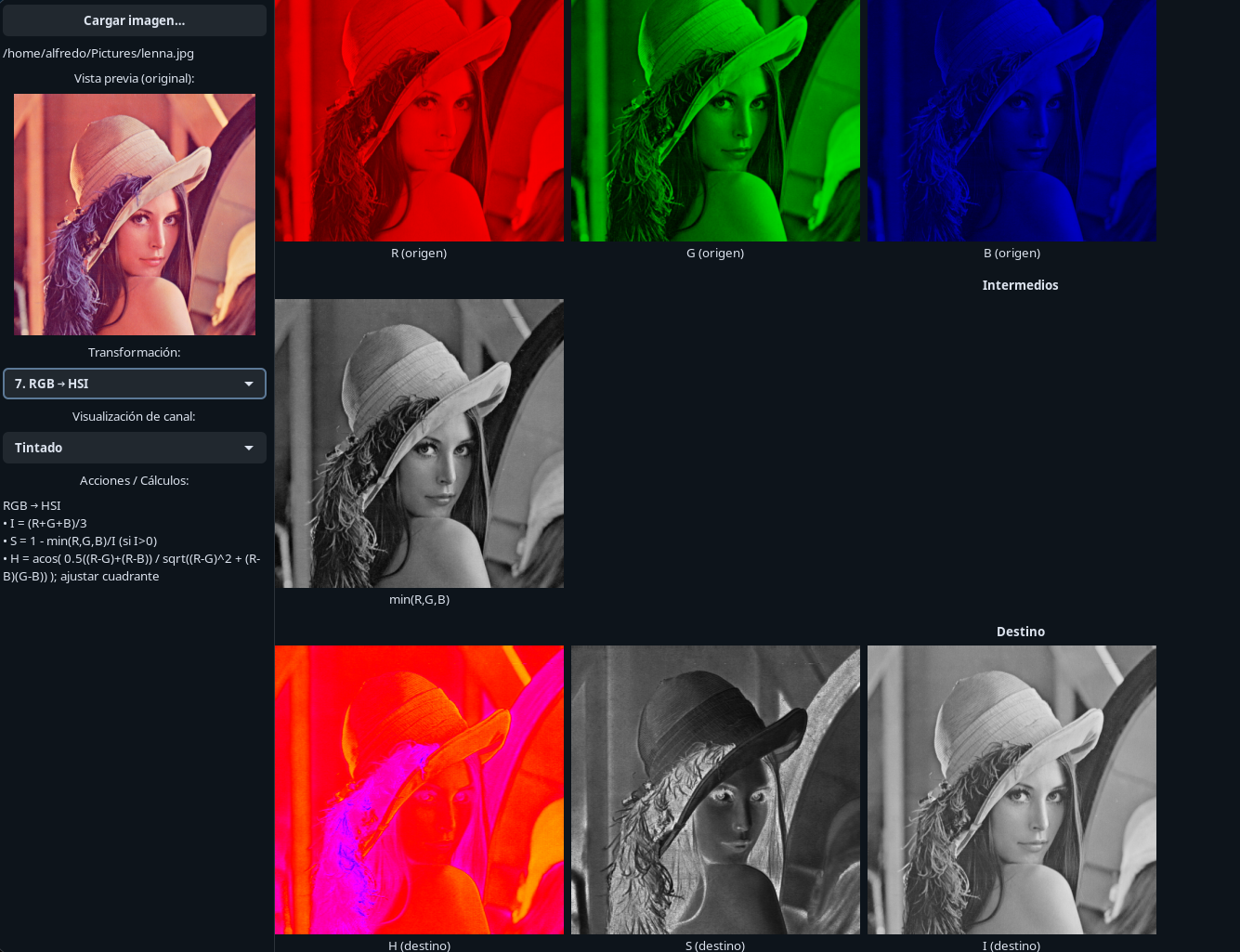
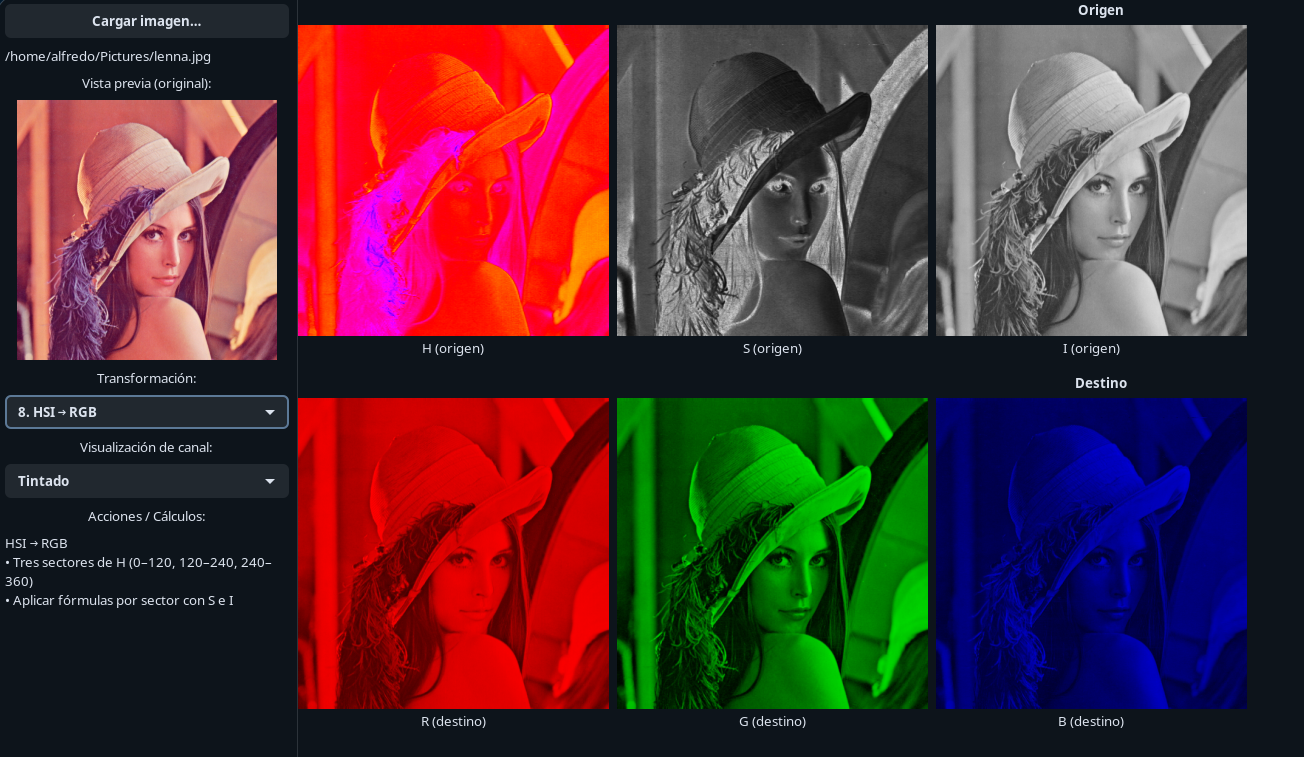
**CMY ↔ CMYK**

* CMY → CMYK:  
  
  + K = min(C,M,Y)
  + Si K=1: C'=M'=Y'=0
  + Si no: C'=(C−K)/(1−K), M'=(M−K)/(1−K), Y'=(Y−K)/(1−K)
* CMYK → CMY:  
  
  + C = C'\*(1−K) + K, M = M'\*(1−K) + K, Y = Y'\*(1−K) + K

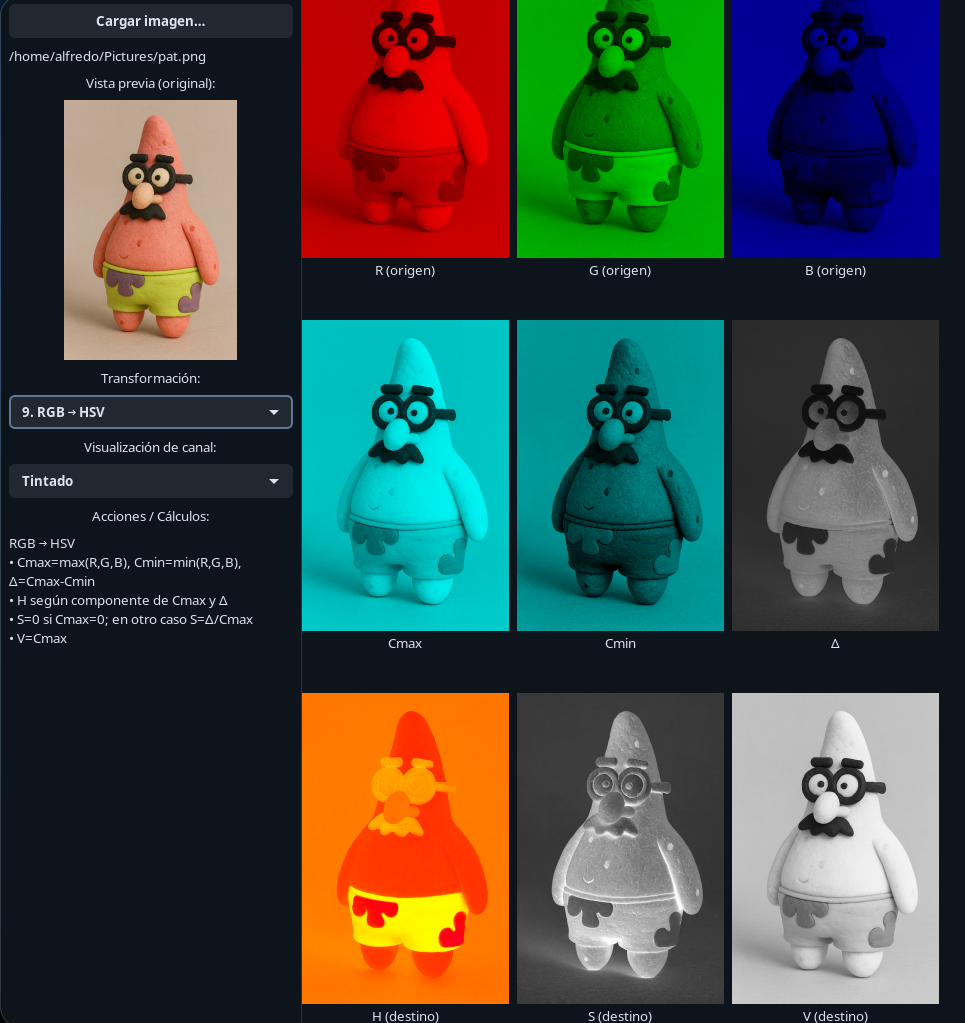
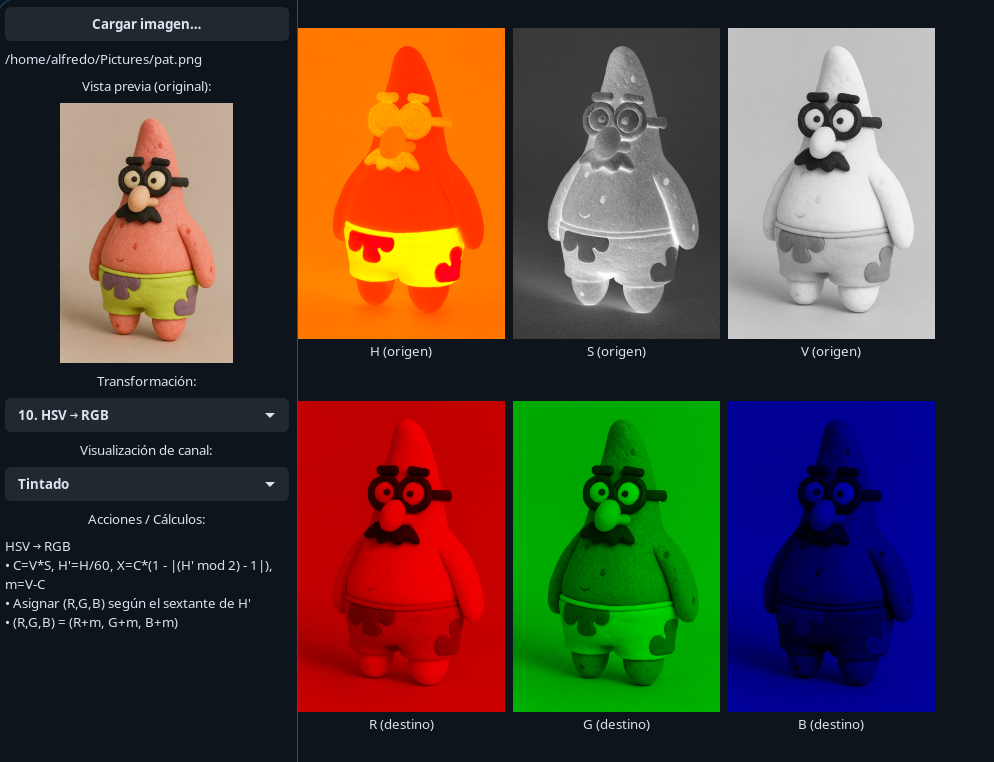
**RGB ↔ YIQ**

* RGB → YIQ:  
   [YIQ]=[0.2990.5870.1140.596−0.274−0.3220.211−0.5230.312][RGB]\begin{bmatrix}Y\\I\\Q\end{bmatrix}= \begin{bmatrix} 0.299 & 0.587 & 0.114\\ 0.596 & -0.274 & -0.322\\ 0.211 & -0.523 & 0.312 \end{bmatrix} \begin{bmatrix}R\\G\\B\end{bmatrix}
* YIQ → RGB:  
   [RGB]=[1.00.9560.6211.0−0.272−0.6471.0−1.1061.703][YIQ]\begin{bmatrix}R\\G\\B\end{bmatrix}= \begin{bmatrix} 1.0 & 0.956 & 0.621\\ 1.0 & -0.272 & -0.647\\ 1.0 & -1.106 & 1.703 \end{bmatrix} \begin{bmatrix}Y\\I\\Q\end{bmatrix}  
   Aplicar **clamping** de R,G,B a **[0,1]**.

**RGB ↔ HSI**

* RGB → HSI:  
  
  + I=(R+G+B)/3
  + S = 1 − min(R,G,B)/I (si I>0)
  + H con:  
     θ=arccos⁡(0.5((R−G)+(R−B))(R−G)2+(R−B)(G−B)),H={θ,G≥B2π−θ,G<B\theta = \arccos\left(\frac{0.5((R-G)+(R-B))}{\sqrt{(R-G)^2+(R-B)(G-B)}}\right),\quad H=\begin{cases} \theta, & G\ge B\\ 2\pi-\theta, & G<B \end{cases}  
     y luego H\_normalizado = H/(2π).
* HSI → RGB:  
  
  + Tres sectores: 0°–120°, 120°–240°, 240°–360°; aplicar fórmulas por sector con S e I.

**RGB ↔ HSV**

* RGB → HSV:  
  ****
  + Cmax=max(R,G,B), Cmin=min(R,G,B), Δ=Cmax−Cmin
  + H según el componente de Cmax (normalizado 0..1)
  + S=0 si Cmax=0; sino S=Δ/Cmax
  + V=Cmax
* HSV → RGB:  
  
  + C=V\*S, H'=H\*6, X=C\*(1−| (H' mod 2) − 1 |), m=V−C
  + Asignar (R,G,B) por sextante de H'; luego sumar m.

## **Metodología y Desarrollo**

### **Arquitectura (C++/GTK+3)**

Se implementó una arquitectura ligera tipo **MVC**:

* **Modelo (transform.cpp/.hpp)**: Funciones puras que reciben estructuras con planos en **[0,1]** y devuelven el modelo de destino (**CMY, CMYK, YIQ, HSI, HSV** o de regreso a **RGB**).
* **Vista/Control (ui.cpp/.hpp)**: Construcción de la interfaz **GTK+3** con **panel izquierdo** (carga de imagen + selector de transformación + **Acciones/Cálculos**) y **panel derecho** (scroller con secciones **Origen**, **Intermedios**, **Destino**).
* **Utilidades (image\_utils.cpp/.hpp)**: Conversión Pixbuf↔planos, generación de miniaturas, y **renderizado de canal** según **modo** (**Gris**, **Tintado**, **Pseudocolor**).

Se retoma la idea (de la práctica previa) de **leer pixeles en GdkPixbuf**, trabajar en memoria con **arreglos** y **tipos concretos**, y actualizar la UI con **pixbufs derivados** de cada plano/canal.

### **Flujo Origen → Intermedios → Destino**

Al seleccionar una **transformación**, el sistema:

1. **Origen** (p. ej., RGB): muestra **R, G, B**.
2. **Intermedios**: calcula y **despliega** las cantidades auxiliares necesarias (p. ej., en **HSV** se muestran **Cmax, Cmin y Δ**; en **HSI**, min(R,G,B) y la normalización de H).
3. **Destino**: muestra los **canales** del modelo objetivo (p. ej., **H, S, V** o **C, M, Y, K**).

Todas las **miniaturas** tienen **tamaño adaptativo** y **pie de imagen**, en un grid que **fluye** según ancho disponible (dos o más columnas). Esta división en **secciones** y su despliegue en la UI de GTK mantiene la claridad del proceso, como el estilo integrador que ya presentaste en la práctica anterior (lectura, separación, ajustes, exportación/visualización).

### **Modos de visualización por canal**

* **Gris**: el canal se muestra con intensidades 0–255.
* **Tintado**:  
  + **R/G/B** tintados en rojo/verde/azul.
  + **C/M/Y** tintados en cian/magenta/amarillo; **K** en gris.
* **Pseudocolor**: mapa simple **azul→cian→verde→amarillo→rojo** (útil para realzar variaciones).
* **Hue (H)**: por defecto se **colorea** (HSV con S=V=1) para interpretar la **tonalidad**; también puede verse en pseudocolor si se escoge ese modo.

### **Implementación de transformaciones (puntos clave)**

* **Normalización**: antes de operar, convertir a **[0,1]**.
* **Clamping**: tras YIQ→RGB (y otras), recortar **R,G,B** a **[0,1]**.
* **Hue**: almacenar **H** normalizado **0..1** (grados/360).
* **Intermedios**:  
  + **HSV**: Cmax, Cmin y Δ (se muestran como planos escala de grises).
  + **HSI**: min(R,G,B) (plano en gris), I y S como planos.
  + **CMYK**: se hace evidente **K = min(C,M,Y)** y la renormalización de **C',M',Y'**.
* **Tamaño de miniaturas**: cálculo responsivo para que todas **se vean uniformes** en el grid.
* **Semántica de pies de imagen**: cada tile indica **canal** y **rol** (p. ej., “H (destino)”, “Cmax (intermedio)”, “R (origen)”).

## **Resultados y validación**

### **Pruebas de consistencia**

**A. Round-trip RGB → CMY → RGB**

* Dado rgb\_in, convertir a cmy = 1−rgb\_in y luego rgb\_out = 1−cmy.
* **Esperado:** rgb\_out ≈ rgb\_in (error solo por redondeos).

**B. Round-trip RGB → YIQ → RGB**

* Verificar **clamping** (YIQ→RGB puede generar salidas fuera de [0,1]).
* **Esperado:** diferencias mínimas en bordes saturados.

**C. Round-trip RGB → HSV → RGB**

* Revisión de **sextantes** y casos Δ=0.
* **Esperado:** identidad si no hay saturación extrema y sin pérdidas de precisión notables.

**D. Round-trip RGB → HSI → RGB**

* Cuidado con **cos(60°−H)** en denominadores y la estabilidad numérica.
* **Esperado:** reconstrucción razonable con desviaciones acotadas por precisión flotante.

**E. CMY ↔ CMYK**

* Verificar que al **subir K**, los **C', M', Y'** se ajustan correctamente y que la **reversión** produce niveles de CMY coherentes.

**F. Casos límite**

* **Δ=0** (grises puros) en **HSV** → H=0, S=0, V=R=G=B.
* **I=0** en **HSI** → S=0, H irrelevante.
* **K≈1** en **CMYK** → C'=M'=Y'=0.
* **Clipping**: imágenes con **altas luces** o **sombras** al pasar por **YIQ**.

### **Evidencia de salida (descripción de pantallas)**

* **Panel izquierdo**:  
  + Botón **Cargar imagen**, **nombre de archivo**, selector de **transformación**, selector de **modo de visualización** y **Acciones/Cálculos** (texto con fórmulas y pasos).
* **Panel derecho (scroller)**:  
  + **Sección Origen**: canales del modelo fuente (p. ej., R, G, B).
  + **Sección Intermedios**: magnitudes auxiliares (p. ej., Cmax, Cmin, Δ, min(R,G,B), K).
  + **Sección Destino**: canales del modelo de llegada (p. ej., H, S, V o C, M, Y, K).
  + **Pies de imagen** y **miniaturas uniformes**; el grid **se adapta** al ancho (2–5 columnas).  
     Esta forma de **desplegar representaciones** y **controles** es consistente con la práctica anterior, donde se mostraba la interfaz y su desglose de áreas funcionales.

## **Discusión**

* **Interpretabilidad**: separar **canales** y mostrar **intermedios** clarifica la lógica de cada modelo; por ejemplo, en **HSV** el usuario ve de dónde sale la **S** (relación con Δ y Cmax) y por qué H es **indefinido** si Δ=0.
* **Visualización**:  
  + **Gris** permite inspeccionar intensidades puras.
  + **Tintado** preserva la **semántica** de cada canal (R/G/B, C/M/Y).
  + **Pseudocolor** realza variaciones sutiles (útil en **H**, **S**, **I/V**, **K**).
* **Estabilidad numérica**:  
  + En **HSI**, la aritmética de acos() y los **denominadores** (cos(60°−H)) requieren **epsilon**.
  + En **YIQ**, la **inversa** puede exceder [0,1] por **mezcla lineal** → aplicar **clamping**.

## **Conclusión**

El desarrollo de esta aplicación cumple y amplifica los objetivos conceptuales y técnicos de la práctica: **comprender la representación digital del color**, **programar conversiones** entre modelos diversos y **explicar visualmente** el recorrido entre el **origen** y el **destino**. La organización en **tres secciones** (Origen → Intermedios → Destino) y la exposición explícita de magnitudes clave (por ejemplo, KK en CMYK, Cmax⁡,Cmin⁡,ΔC\_{\max}, C\_{\min}, \Delta en HSV, min⁡(R,G,B)\min(R,G,B) en HSI) vuelven **auditables** las transformaciones y fomentan una lectura causal: se observa cómo un cambio en un plano **propaga** efectos al resultado final, y se distinguen nítidamente los **papeles** de luminancia, crominancia, tono y saturación.

En términos de **visualización**, disponer de **Gris**, **Tintado** y **Pseudocolor** ofrece perspectivas complementarias. El gris facilita la evaluación cuantitativa (intensidad relativa y contraste), el tintado mantiene una **semántica inmediata** (identificar R, G, B y C, M, Y “a simple vista”), y el pseudocolor realza **gradientes** y **patrones** que podrían pasar desapercibidos. La decisión de renderizar **Hue (H)** a color (S=V=1) por defecto resulta pedagógicamente acertada: acerca la abstracción “ángulo de tono” a una señal visual intuitiva; no obstante, la posibilidad de conmutar a pseudocolor añade una herramienta analítica útil para detectar discontinuidades o envolventes.

Desde la **ingeniería de software**, la arquitectura tipo **MVC** demostró ser valiosa: separa la lógica matemática de la interfaz, facilita pruebas unitarias (round-trips como RGB→Modelo→RGB), y prepara el terreno para extensiones futuras. El cuidado por la **estabilidad numérica** —normalización a [0,1], **clamping** tras inversiones, manejo de **casos límite** y **épsilon** en denominadores— reduce ambigüedades y errores al explorar imágenes reales, donde abundan regiones grises (Δ=0\Delta=0) o intensidades bajas (I≈0I\approx 0). En modelos lineales (RGB↔CMY, YIQ↔RGB) se observan reconstrucciones coherentes, mientras que en los no lineales (RGB↔HSI/HSV) las discrepancias se explican por la definición por tramos y la precisión flotante, aprendizajes que fortalecen el criterio técnico del estudiante.

Pedagógicamente, la herramienta **explica** y **demuestra**. Permite razonar por qué **HSV/HSI** son preferibles para segmentar por **tono**, cómo la **saturación** responde al contraste de canal y por qué la **luminancia** separada (Y en YIQ) simplifica ciertos análisis de brillo. También deja claros los **límites**: YIQ puede producir componentes fuera de rango al invertir; HSI/HSV demandan atención en la definición de HH cuando no hay contraste; y CMYK, al introducir KK, evidencia decisiones de normalización que afectan la interpretación visual y el control de tinta.

De cara al **uso práctico** en visión por computadora y ML, la aplicación aporta un banco de pruebas para **preprocesamiento**: elegir espacio de color según el objetivo (p. ej., HSV para detección por H, YIQ para análisis de luminancia, CMYK para criterios de impresión), verificar **consistencias** antes de alimentar modelos y entender mejor cómo la **distribución** de intensidades por canal impacta la segmentación, la extracción de características o la robustez ante cambios de iluminación.

Finalmente, se abren líneas de **trabajo futuro**: un **inspector por píxel** que reporte (R,G,B), (H,S,V)/(H,S,I) y **0xRRGGBB**; **mapas de pseudocolor** más informativos (viridis/turbo/jet) con barras de color; **exportación** de “contact sheets” y guardado por canal/intermedio; soporte para **16 bits/canal**, **gestión de color** (perfiles ICC) y tratamiento de **gamma** para acercarse al comportamiento de **sRGB** real; y mejoras de **rendimiento** (SIMD, multihilo) para lotes de imágenes. En conjunto, el proyecto no solo satisface la práctica solicitada, sino que deja una base **extensible y didáctica** que conecta la teoría de los modelos de color con aplicaciones concretas en **procesamiento de imágenes** y **aprendizaje automático**, promoviendo una comprensión profunda y operativa del color en sistemas digitales.

## **Referencias**

* Material del curso y práctica previa (lectura, separación de componentes RGB, UI en C++/GTK), utilizados como **base de estilo y estructura** para este reporte.
* Documentación de **GTK+ / GdkPixbuf** (manejo de pixbufs, carga desde archivo).
* **Gonzalez, R. C., & Woods, R. E. (2018). Digital Image Processing. Pearson.** (capítulos de modelos de color y transformaciones).

### **Apéndice A — Detalles de implementación (resumen)**

* **Tipos de imagen**: ImageGray (1 plano), ImageRGB, ImageCMY, ImageCMYK, ImageYIQ, ImageHSV, ImageHSI.
* **Entrada/salida**: GdkPixbuf ↔ estructuras de planos (floats [0,1]).
* **Renderizado**:  
  + Gray → Pixbuf;
  + Tint (factor por canal);
  + Pseudocolor (mapa azul→rojo).
* **Cálculo de miniaturas**: escala “fit” para **no upscaling**; grid con **espaciados** y **columnas** dependientes del ancho del scroller.
* **Señales** GTK: changed en combos (transformación/modo), clicked para **Cargar imagen**, y size-allocate para recálculo del tamaño de tile.
* **Numérico**: clamping pos-conversión; epsilon en denominadores (HSI); H normalizado [0,1].