Tecnológico de Costa Rica

Escuela de Ingeniería Electrónica

Diseño e implementación de una aplicación para generar imágenes HDR mediante la técnica de múltiples exposiciones con mapeo de tonos sobre la plataforma BeagleBoard-xM

Informe de Proyecto de Graduación para optar por el título de

Ingeniero en Electrónica con el grado académico de Licenciatura

Bayron Pérez Vega

Cartago, 26 de Noviembre, 2011

Declaro que el presente Proyecto de Graduación ha sido realizado enteramente por mi persona, utilizando y aplicando literatura referente al tema e introduciendo conocimientos propios.

En los casos en que he utilizado bibliografía he procedido a indicar las fuentes mediante las respectivas citas bibliográficas. En consecuencia, asumo la responsabilidad total por el trabajo de graduación realizado y por el contenido del correspondiente informe final.

Bayron Pérez Vega

Bayron Pérez Vega

Cartago, 31 de enero de 2012

Céd: 7-0163-0166

Instituto Tecnológico de Costa Rica
Escuela de Ingeniería Electrónica
Proyecto de Graduación
Tribunal Evaluador

Proyecto de Graduación defendido ante el presente Tribunal Evaluador como requisito para optar por el título de Ingeniero en Electrónica con el grado académico de Licenciatura, del Instituto Tecnológico de Costa Rica.

Miembros del Tribunal

Ing. Johan Carvajal Godínez
Profesor Lector

Dr. Pablo Alvarado Moya
Profesor Asesor

Los miembros de este Tribunal dan fe de que el presente trabajo de graduación ha sido aprobado y cumple con las normas establecidas por la Escuela de Ingeniería Electrónica.

Cartago, 29 de noviembre de 2011

Resumen

El presente proyecto tiene como objetivo diseñar e implementar una aplicación para generar imágenes de alto rango dinámico mediante la técnica de múltiples exposiciones con mapeo de tonos sobre la plataforma BeagleBoard-xM.

Con este fin, se desarrolla un sistema que permite capturar imágenes a diferentes exposiciones a partir de una señal de activación. Estas imágenes en conjunto con la información de tiempos de exposición se utilizan en el algoritmo de calibración fotométrica para obtener la respuesta con que el sensor mapea las intensidades de luz a valores discretos y el algoritmo de generación de imágenes de alto rango dinámico.

En la visualización se debe aplicar a la imagen de alto rango dinámico un algoritmo de mapeo de tonos con la finalidad de reducir el contraste, esto debido a limitaciones de hardware en los dispositivos de despliegue.

Para mejorar la calidad de la imagen se utiliza el algoritmo de mundo gris y la correción gamma para compensar la no linealidad del receptor del dispositivo de visualización.

Una vez obtenida la imagen de alto rango dinámico con mapeo de tonos, se utiliza el sistema de almacenamiento y visualización. La imagen se almacena en el formato *jpeg* con el máximo factor de calidad.

El proceso anterior se implementa para el procesador ARM y posteriormente en el procesador DSP de la plataforma BeagleBoard-xM. La evaluación de los algoritmos se da por medio de los tiempos de ejecución e histogramas de frecuencias de las imágenes con diferentes exposiciones e imagen de alto rango dinámico de una escena estática.

Palabras clave: HDR, Mapeo de tonos, DSP, codec, servidor, GStreamer, Codec Engine, iUniversal, CMEM, XDM, XDAIS, modelo de mundo gris, memoria contigua, espacio de color.

Abstract

This project aims the design and deployment of an application to generate high dynamic range images using a technique of multiple exposures with tone mapping on the BeagleBoard-xM platform.

For this purpose, a system to capture images with different exposures on a trigger signal is developed. These images, along with information on exposure times, are used in the photometric calibration algorithm to obtain the sensor response used to map light intensities to discrete values, and the high dynamic range images generation algorithm.

To display the high dynamic range image, a tone mapping algorithm has to be applied in order to reduce the contrast, due to hardware limitations of display devices.

Once the high dynamic range image with tone mapping is obtained, it is passed to the storage and display system. The image is stored in jpeg format with the highest quality factor.

The above process is implemented in both ARM and DSP processors in the BeagleBoard-xM platform. The evaluation of algorithms is given by execution times and image frequency histograms of different exposures and the high dynamic range image of a static scene.

Keywords: High Dynamic Range, Tone Mapping, Codec, Server, DSP, iUniversal, Gray-World Assumption, Color Space, CMEM.

a mi familia y a las personas que han estado presentes en alguna etapa de mi vida y me han apoyado de forma incondicional

Agradecimientos

Deseo agradecer primeramente a Dios por la fuerza de voluntad y sabiduría que me ha dado para enfrentar los diferentes retos de la vida. Al profesor y amigo el Dr. Pablo Alvarado por su apoyo y consejos en la etapa final de mis estudios.

A la empresa RidgeRun y compañeros de trabajo por su apoyo, escucharme y darme consejos en diferentes etapas del desarrollo del proyecto.

Agradezco a mi familia por siempre escucharme, apoyarme y darme inspiración para salir adelante con mis metas. A mi hermana Katherine Pérez que me ayudó en la etapa final de mis estudios.

Al grupo Softmer conformado por mis mejores amigos de la universidad, quienes estuvieron presentes en todo momento apoyándome.

Sin ser menos importante un agradecimiento al profesor Johan Carvajal quien me dio consejos en diferentes etapas de mis estudios.

Agradezco en general a todas aquellas personas que en alguna etapa de mi vida me dieron de su apoyo y consejos, lo que contribuyó en mi desarrollo personal y profesional.

Bayron Pérez Vega

Cartago, 31 de enero de 2012

Índice general

Índice de figuras

Índice de tablas

Lista de símbolos y abreviaciones

Abreviaciones

AE	Auto Exposición (*Auto Exposure*)
API	Interfaz de Programación de Aplicaciones (*Applications Programming Interfaz*)
AWB	Auto Balance de Blancos (*Auto White Balance*)
CE	Codec Engine
CRT	Tubo de Rayos Catódicos (*Cathode Ray Tube*)
DSP	Procesador de Señales Digitales (*Digital Signals Processor*)
EV	Valor de Exposición (*Exposure Value*)
GPP	Porcesador de Próposito General (*General Purpose Processor*)
HDR	Alto Rango Dinámico (*High Dynamic Range*)
IPC	Comunicación Interprocesador (*Interprocessor Communication*)
LCD	Pantalla de Cristal Líquido (*Liquid Crystal Display*)
LDR	Bajo Rango Dinámico (*Low Dynamic Range*)
RTSC	Componentes de Software de Tiempo Real (*Real Time Software Components*)
SDK	Paquete de Desarrollo de Software (*Software Development Kit*)
SoC	Sistema sobre un chip (*System on Chip*)
VISA	Vídeo Imágenes Habla Audio (*Video Imaging Speech Audio*)
XDAIS	eXpress DSP Algorithm Interoperability Standard
XDM	eXtremeDSP Digital Media
xM	Memoria Extendida (*Extended Memory*)

Notación general

$\mathbf{A}$	Matriz. $\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{bmatrix}$
$\mathcal{T}[\cdot]$	Transformación realizada por un sistema
$\underline{\mathbf{x}}$	Vector. $\underline{\mathbf{x}} = [x_1\ x_2\ \ldots\ x_n]^T = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$
y	Escalar.

Capítulo 1

Introducción

Actualmente existe un incremento en el desarrollo de sistemas embebidos y por ende aplicaciones que se ejecutan sobre estos para diferentes áreas como medicina, control automático, redes, entre otras.

1.1. Sistemas embebidos

Un sistema embebido (o sistema empotrado) es un sistema computacional constituido por elementos de hardware y software, el cual se optimiza con funciones preestablecidas y es utilizado en tareas específicas.

En el mercado de tecnologías se encuentra variedad de sistemas embebidos como OMAP-L138-EVM, DM365-Leopard, DM8148-EVM, OMAP3530, DM3730 (BeagleBoard-xM), entre otras.

1.1.1. BeagleBoard-xM

La plataforma BeagleBoard-xM es una tarjeta desarrollada por la organización BeagleBoard. Este sistema posee memoria extra (xM, extra memory) con 512MB de RAM DDR de baja potencia y se diseñó para el uso de aplicaciones multimedia.

El sistema BeagleBoard-xM está compuesto por un núcleo heterogéneo (dos procesadores distintos en un mismo chip, SoC), un procesador ARM cortex A8 como procesador de próposito general (GPP) y un procesador DSP C64P como un procesador de próposito específico. Además posee controladores y hardware para DVI-D, S-vídeo, USB (anfitrión y OTG), Ethernet, puerto para cámara, entrada y salida de audio estéreo, JTAG, RS232, puerto micro-SD, entre otros [4].

Los módulos de cámara adapatables al sistema son producidos por Leopard Imaging, quien se dedica al diseño y manufactura de cámaras embebidas de alta definición. Los sensores de captura son de la familia LI-MOD, sensores CMOS con resoluciones desde VGA, 1,3Mp,

2Mp, 3Mp hasta 5Mp.

Además la BeagleBoard-xM tiene soporte de software. La empresa RidgeRun ofrece un paquete de desarrollo de software (SDK), basado en GNU/Linux e integra paquetes como GStreamer, DBus, glib, openCV, libusb, jpeg, libpng, entre otros, con la finalidad de simplificar el desarrollo de aplicaciones multimedia como la generación de imágenes de alto rango dinámico.

1.2. Imágenes de alto rango dinámico

Las imágenes de alto rango dinámico (o imágenes HDR por sus siglas en inglés High Dynamic Range) son un conjunto de técnicas que permiten un mejor rango dinámico de luminancia (intensidad de luz máxima menos intensidad de luz mínima) entre regiones saturadas y zonas oscuras de una imagen.

El rango dinámico de las imágenes HDR está comprendido por el rango de la combinación de las imágenes digitales estándar, dando así un rango dinámico mayor que una imagen digital estándar, la información que contiene debe ser representada con más bits que una imagen estándar, por lo que existen formatos que soportan valores de punto flotante para imágenes HDR como:

- RadianceHDR (RGBE), 8 bits por canal R, G y B, y 8 bits de exponente que poseen en común los canales. 32 bits por pixel.
- OpenEXR (EXR), 16 bits por canal, 48 bits por pixel.
- HD Photo (JPEG-XR), 96 bits por pixel.

La técnica de imágenes HDR surge debido a la limitación de rango dinámico que se presenta en los sensores de captura en comparación al rango que percibe el ojo humano, el cual se adapta a diferentes luminosidades a diferencia de los sensores de captura que no poseen esta capacidad.

Con la finalidad de poder desplegar las imágenes HDR en dispositivos que poseen un menor rango dinámico como monitores CRT, LCD, impresoras, etc, se utiliza un algoritmo de mapeo de tonos en la imagen HDR.

1.3. Mapeo de tonos

La técnica de mapeo de tonos consiste en la reducción del contraste o rango dinámico de luminancia de la imagen HDR. Existen dos categorías de mapeo de tonos en el dominio espacial: locales y globales. Los primeros reducen el contraste por regiones, zonas o vecindades de píxeles (picture elements, elementos de imagen) y los globales reducen el contraste global de la imagen considerando todos los píxeles de la imagen.

1.4. Imágenes HDR en la BeagleBoard-xM

La aplicación de generación de imágenes HDR con mapeo de tonos en el sistema BeagleBoard-xM es un proceso que se ve limitado por factores como: recursos de hardware, selección de entorno de desarrollo, paquetes y herramientas necesarias para el desarrollo de la aplicación, integración con dispositivos externos, etc.

La empresa RidgeRun ofrece un paquete de desarrollo de software (SDK) que permite desarrollar aplicaciones de multimedia haciendo uso de los recursos de la plataforma BeagleBoard-xM e integración de dispositivos externos, paquetes y herramientas de software, lo anterior con la finalidad facilitar y optimizar el tiempo de desarrollo de aplicaciones multimedia [39].

La creación de aplicaciones multimedia o aplicaciones demostrativas es la parte de la gama de productos que ofrece la empresa RidgeRun, por lo cual una aplicación que integre un módulo de sensor de captura y haga uso de los dos procesadores, permite a las empresas que contratan a RidgeRun promover sus productos mediante las aplicaciones demostrativas.

Un sistema de generación de imágenes HDR con mapeo de tonos es una aplicación del aŕea de gráficos por computador y de procesamiento y análisis de imágenes digitales, que permite hacer uso de los recursos ofrecidos por el SDK.

El presente trabajo es el desarrollo e implementación del sistema de generación de imágenes HDR con mapeo de tono en la BeagleBoard-xM, el cual conlleva varias etapas y consideraciones de recursos tanto a nivel de hardware como software.

La figura 1.1 muestra un diagrama de bloques general del sistema de generación de imágenes HDR con mapeo de tonos en una BeagleBoard-xM, donde se aprecian cinco bloques: sensor de captura, sistema de captura, sistema de generación de HDR, sistema de mapeo de tonos y por último el sistema de almacenamiento y despliegue de imágenes.

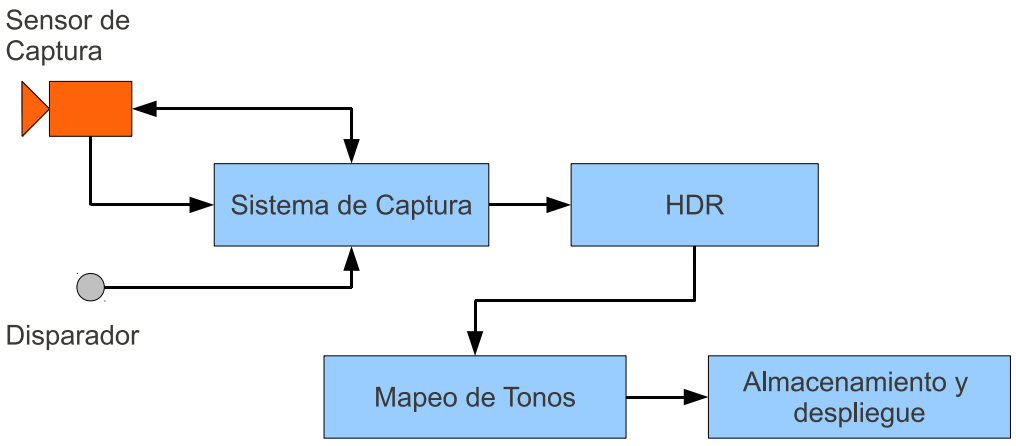

Figura 1.1: Diagrama general de un sistema de generación de imágenes HDR con mapeo de tonos

El módulo de sensor de captura corresponde a un sensor del tipo LI-MOD conectado a la

BeagleBoard-xM, que permite adquirir imágenes del medio con variaciones en el tiempo de exposición, diferentes ganancias de los colores primarios, ajuste de balance de blancos, entre otros.

El módulo del sistema de captura, permite obtener mediciones de tiempo de exposición y ajustar el sensor para obtener imágenes a diferentes exposiciones, las cuales son enviadas al sistema de generación de HDR para la obtención de la imagen, en donde el valor de intensidad de luz de cada pixel en la imagen es un valor de tipo flotante. Luego la imagen es transformada por medio del sistema de mapeo de tonos a un formato como (jpeg o png), para que esta sea escrita en una unidad de almacenamiento y finalmente visualizar la imagen en un dispositivo de despliegue.

1.5. Objetivos y estructura del documento

El objetivo general del presente proyecto consiste en desarrollar un sistema de generación de imágenes HDR con mapeo de tono utilizando un sensor de captura de la familia LI-MOD en un sistema embebido BeagleBoard-xM. Con este fin, se debe crear un módulo de adquisición que permita controlar los parámetros del sensor de captura, donde se puedan obtener imágenes con diferentes tiempos de exposición. Además se debe diseñar y programar un módulo de procesamiento de imágenes para la generación de imágenes HDR. También es requerido un módulo de mapeo de tonos con la finalidad de mapear la imagen HDR a un formato soportado por los dispositivos de despliegue y almacenamiento de datos. Finalmente se debe comparar la imagen resultante con las imágenes obtenidas por el sensor de captura a diferentes exposiciones, en cuanto al rango dinámico de luminancia presente en cada imagen y calidad.

El presente trabajo se enfoca en explicar el desarrollo del sistema de generación de imágenes con mapeo de tono en el sistema empotrado BeagleBoard-xM y los resultados obtenidos en la implementación del proyecto. El capítulo 2 presenta los fundamentos teóricos en los que se basa la solución. El capítulo 3 muestra detalladamente los módulos que conforman la solución, esbozados en la figura 1.1. El capítulo 4 presenta los resultados obtenidos en la generación de imágenes HDR con mapeo de tono en el sistema embebido BeagleBoard-xM y finalmente el capítulo 5 contiene las conclusiones del presente trabajo y las recomendaciones para futuros desarrollos.

Capítulo 2

Marco Teórico

2.1. Fundamentos de óptica

En la percepción de la luz intervienen tres factores: fuentes de luz, objetos y el sistema visual humano. A continuación se detalla el sistema visual humano, estructura y percepción de los haces de luz, puesto que las imágenes HDR se generan para satisfacer las demandas perceptivas de los usuarios.

2.1.1. Sistema visual humano

En la figura 2.1, se aprecia el sistema de proyección del ojo humano comprendido por el iris, cristalino y la pupila. Además está presente un sistema de conversión de patrones de luz a señales neuronales, principalmente comprendido por la retina que contiene aproximadamente 130 millones de células fotorreceptoras (115 a 125 millones de bastones y cerca de 7 millones de conos), sensibles a la luz. La distribución de las células fotorreceptoras no es uniforme, la mayor densidad de conos se encuentra en la fóvea [30].

Las principales diferencias entre conos y bastones citadas en [1] son:

- Físicamente, los bastones son más grandes que los conos.
- Los conos se encargan de la visión en detalle.
- La respuesta temporal de los bastones es mayor que la de los conos, por lo que los bastones se usan para detectar movimiento.
- En lo referente a la respuesta luminosa, los bastones son más sensibles, encargados de la visión nocturna (visión escotópica) y los conos son encargados de la visión en color (visión fotópica).

Existen tres tipos de conos a determinadas logitudes de onda cercanas al:

- Azul, S (short, pequeñas).
- Verde, M (middle, medias).
- Rojo, L (long, largas).

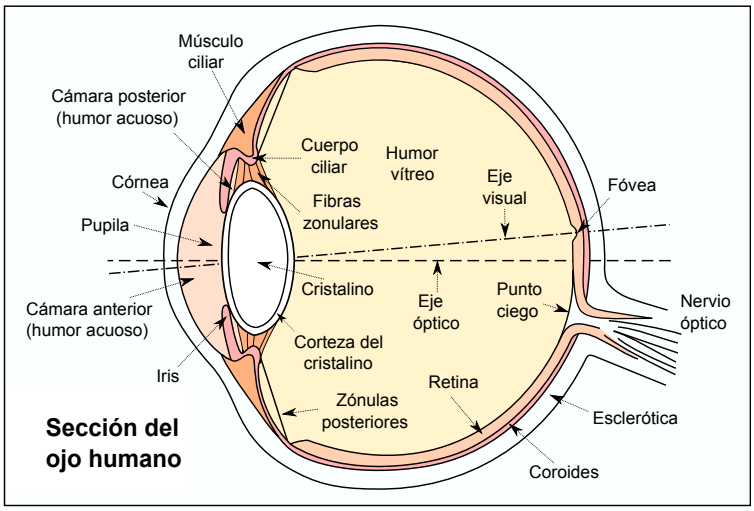

Figura 2.1: Anatomía del ojo humano [38]

La proporción de conos L:M:S es de 40:20:1, por lo que existe una mayor cantidad de células fotorreceptoras de rojo, luego verde y por último azul. En lo referente a la sensibilidad los conos verdes son los más sensibles y los azules los menos sensibles [30].

2.1.2. Conceptos básicos

En [9] y [32] se definen las siguientes magnitudes físicas, requeridas para plantear el funcionamiento de las imágenes HDR:

Brillo: es el atributo de una sensación visual según la cual un área parece emitir más o menos luz.

Fotometría: conjunto de técnicas para medir la luz que es detectable por el ojo humano en términos de brillo percibido.

Radiometría: conjunto de técnicas para medir radiación electromagnética o energía radiante en función de potencia absoluta.

Luminancia: cantidad de energía que percibe un observador de una fuente luminosa; se mide en candelas por metro cuadrado (cd/m^2).

Radiancia: cantidad de energía que suministra una fuente luminosa; se mide en watts (W).

Por ejemplo, si una fuente luminosa emite a longitudes de onda que están fuera del rango visible para el ser humano se tiene que la luminancia es nula, mientras que la radiancia posee un valor elevado con respecto a la luminancia.

Irradiancia: es la potencia recibida por unidad de área, se mide en watts por metro cuadrado (W/m^2).

Intensidad: es una medición lineal de luz sobre algún intervalo del espectro electromagnético del flujo de potencia.

Contraste: es el rango dinámico de intensidades que existe entre un punto de la imagen y sus alrededores.

Saturación: es la intensidad del color de una área en proporción a su brillo. La saturación va desde gris neutro a los colores saturados (colores puros del espectro).

Crominancia: es la componente de la señal que contiene información del color.

2.1.3. Rango dinámico y percepción humana

El ojo humano puede percibir un rango de contraste cercano a 24 EV. Un EV es la reducción a la mitad (-1 EV) o duplicación (+1 EV) de la cantidad de luz en una escena. Esto se debe a la apertura variable de la pupila y a mecanismos de adaptación en la retina.

La pupila se adapta a la cantidad de luz recibida para la visión óptima. Si la cantidad de luz es intensa la pupila se cierra para para limitar la cantidad de radiación y si es escasa se abre, permitiendo la visión en condiciones desfavorables de iluminanción.

Además, la exposición prolongada a una luz intensa produce que los bastones y conos reduzcan su sensibilidad a la luz adaptándose al entorno. En el caso de adaptación a la oscuridad, los bastones y conos aumentan su sensibilidad a la intensidad de luz.

El rango dinámico de las cámaras dígitales estándar es reducido con respecto al del sistema visual humano. Las cámaras dígitales estándar con convertidores ADC de 8 bits poseen un rago dinámico de 8 EV [13].

La técnica de imágenes de alto rango dinámico amplía el rango dinámico de una imagen a partir de imágenes con diferentes exposiciones, lo que permite al ojo humano percibir información que en una imagen estándar no se aprecia. El rango dinámico reducido los dispositivos de captura y visualización se debe a limitaciones de hardware.

2.2. Generalidades de imágenes digitales

Las imágenes digitales son representaciones de funciones bidimensionales que han sido muestreadas o discretizadas en el dominio espacial como en su codominio de intensidad de luz. Cada par ordenado (x,y) representa la posición de un píxel [9].

2.2.1. Espacios de color

Los colores se representan mediante puntos en un espacio de color. Existen diferentes espacios de color desarrollados para aplicaciones específicas, por ejemplo: los espacios de color RGB, YCbCr y XYZ, los cuales se basan en la teoría tricrómatica (el ojo humano tiene tres tipos de conos, por lo que bastan tres componentes para describir un color percibido) [32].

Los espacios de color YCbCr y RGB son espacios de color dependientes del dispositivo

de hardware utilizado para la visualización, por lo que la percepción de un color en un dispositivo de despliegue como un monitor CRT es diferente a la percepción del mismo color en un monitor LCD, esto debido a la dependencia del sistema de color al hardware y la forma en que se mapean los colores al dispositivo de despliegue.

Existen espacios de color independientes del dispositivo de visualización, como el caso del espacio de color XYZ, donde la percepción del color no depende de la estructura del dispositivo de despliegue.

Espacio de color RGB

El espacio RGB es engendrado por tres ejes independientes que pertenecen a los colores primarios aditivos R, G y B (figura 2.2). Cada componente cromática puede ser procesada independientemente y tiene valores entre 0 y 255 en imágenes estándar.

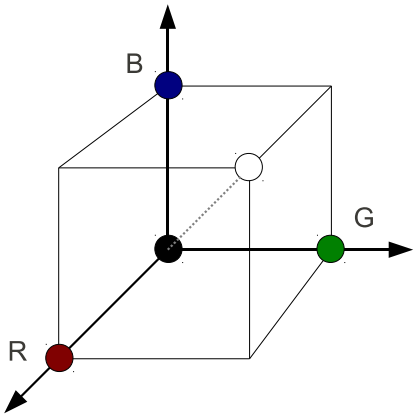

Figura 2.2: Representación del espacio de color RGB

La escala de grises esta representada mediante una línea que va desde el color negro al color blanco mostrado en la figura 2.2.

Espacio de color YCbCr

Es un estándar digital utilizado frecuentemente en la transmisión de señales de televisión. Y es luminancia, Cb la crominancia del azul y Cr la crominancia del rojo.

Este espacio tiene su origen en una característica del sistema visual humano: la sensibilidad a las intensidades debido a las células fotorreceptoras del tipo bastones presentes en el ojo humano.

Espacio de color XYZ

Se obtiene por transformación lineal del sistema RGB. Las magnitudes de las componentes XYZ son proporcionales a la energía física, pero su composición espectral corresponde a las características de color coincidentes con la visión humana [32]. La diferencia con el espacio de color RGB es que XYZ es un espacio perceptivo, que se obtiene de curvas de un observador humano promedio y RGB es un formato asociado a tecnologías de captura y reproducción.

2.2.2. Conversión entre espacios de color

Conversión entre YCbCr y RGB

El sistema de ecuaciones lineales de transformación entre YCbCr y RGB, se basan en el estándar CCIR 601, actualmente ITU-R 601 [37]:

$$\begin{aligned} R &= Y + 1,402Cr - 179,456 \\ G &= Y - 0,34414Cb - 0,71414Cr + 135,45984 \\ B &= Y + 1,772Cb - 226,816 \end{aligned} \tag{2.1}$$

De estas ecuaciones se pueden despejar las componentes de YCbCr para resultar en:

$$\begin{aligned} Y &= 0,299R + 0,587G + 0,114B \\ Cb &= 0,500R - 0,4187G - 0,0813B + 128 \\ Cr &= -0,1687R - 0,3313G + 0,500B + 128 \end{aligned} \tag{2.2}$$

Conversión entre XYZ y RGB

Para un observador 2i (observador estándar con un ángulo de visión de 2 grados) y el iluminante D65 (luz del día), el rango de valores de X es entre 0 y $95,047$, Y entre 0 y 100 y Z entre 0 y $108,883$ [25]. Los valores del espacio de color XYZ son normalizados con el valor de 100, para posteriormente calcular los valores de RGB con el sistema de ecuaciones lineales de transformación:

$$\begin{aligned} R' &= 3,240708X - 1,537259Y - 0,498570Z \\ G' &= -0,969257X + 1,875995Y + 0,041555Z \\ B' &= 0,055636X - 0,203996Y + 1,057069Z \end{aligned} \tag{2.3}$$

En [31] después de la transformación lineal entre XYZ y RGB, se convierte de un espacio de color RGB lineal a uno no lineal (sRGB), con la finalidad de mapear valores negativos del espacio RGB a un valor positivo en el espacio sRGB, evitando recortar los valores negativos a cero y valores máximos a 255 en una representación de 8 bits por canal.

$$C = \begin{cases} 1,055C'^{\left(\frac{1}{2,4}\right)} - 0,055; & \text{si } C' > 0,0031308 \\ 12,92C'; & \text{caso contrario} \end{cases} \tag{2.4}$$

Con
C : valores de un canal del espacio sRGB.
C' : valores de un canal del espacio RGB.

Por último, los valores de cada canal R, G y B se encuentran en el rango de 0 a 1, por lo que se deben multiplicar por 255 para representarlos en 8 bits por canal.

Conversión entre RGB y XYZ

En [31] primero se convierte del espacio RGB al espacio sRGB, donde el rango de valores de cada canal es de 0 a 1.

$$C' = \begin{cases} \left(\frac{C+0,055}{1,055}\right)^{2,4}; & \text{si } C > 0,04045 \\ \frac{C}{12,92}; & \text{caso contrario} \end{cases} \tag{2.5}$$

Con
C' : valores de un canal del espacio RGB.
C : valores de un canal del espacio sRGB.

Cada valor de R, G y B se multiplica por 100 y se utiliza el sistema de ecuaciones lineales de transformación de RGB a XYZ.

$$\begin{aligned} X &= 0,412424R + 0,357579G + 0,180464B \\ Y &= 0,212656R + 0,715158G + 0,072186B \\ Z &= 0,019332R + 0,119193G + 0,950444B \end{aligned} \tag{2.6}$$

El rango de valores de los canales X, Y y Z es dado por el estándar del observador 2i e iluminante D65.

2.3. Sensores de captura

Los sensores de captura de imágenes digitales son los encargados de discretizar la señal luminosa (haces de luz en colores primarios R, G y B). Estos se clasifican en CCD (Charge Coupled Device, Dispositivo de Cargas Acopladas) y CMOS (Complementary Metal Oxide Semiconductor, Semiconductor de Óxido de Metal Complementario). Estos fotodetectores se usan para convertir la luz en carga eléctrica. La diferencia radica en la forma en que obtienen datos de intensidades de luz [26].

Los sensores usan un patrón de color Bayer como el que se muestra en la figura 2.3, donde se observa que por cada cuatro valores de verde, solo hay dos valores de rojo y dos de azul.

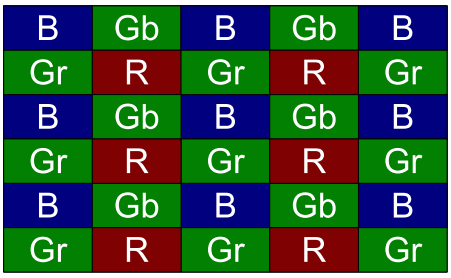

Figura 2.3: Patrón de color Bayer

El rango dinámico de luminancia en los sensores de captura se encuentra limitado por razones de hardware y precisión en las variables. Debido a ello, la captura mediante el patrón de color Bayer puede obtener zonas saturadas y zonas oscuras en una imagen que el ojo humano, a través de sus procesos de adaptación, captura sin saturación y sin zonas oscuras.

2.4. Imágenes de alto rango dinámico

En el área de imágenes con un alto rango dinámico (HDRI, de *High Dynamic Range Imaging*) se han desarrollado varias técnicas para mejorar el rango dinámico efectivo de cámaras usando imágenes o fotografías de la misma escena (escenas estáticas) de bajo rango dinámico (LDR, *Low Dynamic Range*) a diferentes tiempos de exposición del sensor a la intensidad de luz.

Este enfoque requiere de una calibración inicial del sensor de captura para determinar la función de respuesta del sensor ante intensidades de luz y generar la imagen HDR.

La función de respuesta del sensor corresponde a la función de mapeo de la intensidad de luz que llega a la cámara en valores de R, G y B. En [24] y [28] la función de respuesta del sensor es considerada lineal y utilizan un píxel de cada imagen para determinar la respuesta. Las cámaras en general no poseen una respuesta lineal por lo que estos métodos no son aplicables [22].

En [6] se calcula la respuesta del sensor a partir de un conjunto de píxeles de las imágenes y realiza una optimización de la curva de respuesta con restricción de suavidad.

Mitsunaga y Nayar en [27] utilizan, para describir la curva del sensor o función de respuesta radiométrica, una función paramétrica (polinomio de orden n). Por otro lado en [22] la respuesta del sensor se calcula mediante una calibración fotométrica, donde la respuesta del sensor se optimiza con todos los píxeles de todas las imágenes asignando mayor peso a valores con exposiciones altas, lo que permite estabilidad en la respuesta de sensor ante ruido de datos de entrada.

Las imágenes HDR con mapeo de tono pueden presentar problemas como aparación de halos (debido al contraste local de la escena), artefactos fantasma (debido a movimientos en la escena), antinaturalidad, post-proceso, recuperaración del contraste global y zonas irreales sueltas [34].

De forma similar a la aproximación polinomial de Mitsunaga y Nayar [27], en [10] la función de respuesta se calcula con derivación desde el histograma, lo que permite movimientos moderados en la escena.

La cantidad de imágenes necesarias a diferentes exposiciones para generar HDRI depende del rango dinámico de la escena y limitaciones como tiempo de ejecución o procesamiento, donde como mínimo se utiliza una imagen sub-expuesta a -2 EV, una a exposición normal y sobre-expuesta a $+2$ EV.

En la figura 2.4 se muestra un ejemplo de HDRI con mapeo de tonos y las imágenes a diferentes EV utilizadas en la generación de la imagen HDR.

Figura 2.4: Ejemplo de HDRI [33]

2.4.1. Algoritmo de Robertson, Borman y Stevenson

En [22] se realiza una calibración fotométrica para obtener la respuesta del sensor ante intensidades de luz, contemplando la no linealidad de la función de respuesta de la cámara, la cual asigna pesos altos a valores de píxel tomados a exposiciones altas. Dicha función de respuesta describe la relación entre los valores de píxeles y la intensidad de luz que alcanza el sensor.

En lo referente a la generación de imágenes HDR, una vez realizada la calibración, se crea la imagen HDR con un promedio ponderado de las imágenes de entrada. La calibración del sensor sólo se lleva a cabo una vez.

Modelo de observación

Para N imágenes de una escena estática, el j-ésimo píxel de la imagen expuesta con tiempo de exposición t_i se denota y_{ij} y el conjunto $\{y_{ij}\}$ representa las observaciones conocidas,

donde $i = 1, 2, ..., N$. La finalidad del algoritmo de HDR es determinar los valores de luz subyacente o irradiancias (x_j) de las observaciones y_{ij}. Si solamente se varía el tiempo de exposición, la cantidad de luz en el valor de salida y_{ij} será $t_i x_j$.

Para contemplar el ruido de captura se introduce un término de ruido aditivo N_{ij}^c, el cual es mapeado por la función de respuesta fotométrica de la cámara $f(\cdot)$ para dar los valores de salida

$$y_{ij} = f(t_i x_j + N_{ij}^c) \tag{2.7}$$

En el contexto de imágenes digitales, los valores mapeados son reales y positivos $\{R^+\}$, con un rango de espacio (O) entre 0 y 255 para 8 bits de datos por canal. La función de respuesta del sensor se obtiene con

$$f(z) = \begin{cases} 0; & \text{si } z \in [0, I_0] \\ m; & \text{si } z \in]I_{m-1}, I_m]; \quad m = 1, ..., 254 \\ 255; & \text{si } z \in]I_{254}, \infty[\end{cases} \tag{2.8}$$

donde I_m es el valor de respuesta del sensor en la posición m. La función $f(\cdot)$ está definida en términos de 255 valores, donde para una respuesta lineal los valores de I_m deben estar uniformemente distribuidos.

Desarrollo matemático del algoritmo con función de respuesta conocida

Con la función de respuesta $f(\cdot)$ conocida se puede definir un mapeo desde el espacio O a R^+ como

$$f^{-1}(y_{ij}) = t_i x_j + N_{ij}^c + N_{ij}^q = I_{y_{ij}} \tag{2.9}$$

donde la función $f^{-1}(\cdot)$ no es la función inversa, sino un mapeo no inyectivo de R^+ al rango de valores entre 0 y 255 en el espacio O, y N_{ij}^q es el error de cuantificación debido a la asignación incierta de $f^{-1}(m) = I_m$.

También debido a la naturaleza aditiva del ruido, (2.9) se simplifica como

$$I_{y_{ij}} = t_i x_j + N_{ij} \tag{2.10}$$

El ruido N_{ij} se modela con una distribución gaussiana independiente de media cero con variables aleatorias y varianza σ_{ij}^2. Las varianzas son escogidas heurísticamente y reemplazadas por pesos

$$w_{ij} = \frac{1}{\sigma_{ij}^2} \tag{2.11}$$

Los pesos se seleccionan con base en la confianza y en la precisión de los datos observados. En los extremos la sensibilidad del sensor decrece, por lo que se utiliza una función de ponderación cerca del valor 128 del tipo gaussiana:

$$w_{ij} = w_{y_{ij}} = e^{\frac{-4(y_{ij}-127,5)^2}{(127,5)^2}} \tag{2.12}$$

donde w_0 y w_{255} tienden a cero y $w_{127,5} = 1$.

La función objetivo es

$$O(x) = \sum_{i,j} w_{ij}(I_{y_{ij}} - t_i x_j)^2 \tag{2.13}$$

que se optimiza haciendo el gradiente $\nabla O(x) = 0$ con respecto al tiempo de exposición t_i y utilizando la relación:

$$x_j = \frac{\sum_i w_{ij} t_i I_{y_{ij}}}{\sum_i w_{ij} t_i^2} \tag{2.14}$$

Calibración Fotométrica

Existe una dependencia de los valores de x_j con respecto a los valores de I_m, mostrada en (2.14). A partir de la función objetivo mostrada en (2.13) y una forma de relajación de Gauss-Seidel se determina la función de respuesta del sensor.

Primero se minimiza la función con respecto a I_m, lo que significa derivar (2.13) con respecto a I_m e igualar a cero.

$$\tilde{I}_m = \frac{1}{|E_m|} \sum_{i,j \in E_m} t_i x_j \tag{2.15}$$

donde $E_m = \{(i,j) : y_{ij} = m\}$ y $|E_m|$ es la cardinalidad de E_m (cantidad de veces que m fue observado).

Luego se minimiza con respecto a x_j utilizando el valor de I_m; esto constituye una iteración del algoritmo. Para esta primera iteración se escoge una función lineal con $I = 128$. El proceso se repite hasta converger y el criterio de convergencia es que la función objetivo decrezca hasta caer bajo un umbral preestablecido.

2.5. Modelo de mundo gris y teoría de Retinex

En el modelo de mundo gris la intensidad media de los canales rojo, verde y azul para una escena deben ser iguales. Primero se calculan los valores promedio de cada canal. En el caso

que sean iguales, la imagen ya cumple el modelo de mundo gris; caso contrario se mantiene el canal verde sin cambios y se obtiene una ganancia para los canales rojo (α) y azul (β).

$$\begin{aligned} \alpha &= \frac{G_{avg}}{R_{avg}} \\ \beta &= \frac{G_{avg}}{B_{avg}} \end{aligned} \tag{2.16}$$

Luego se ajustan los píxeles de los canales rojo y azul, multiplicándolos por las ganancias α y β, respectivamente. Esto permite mapear el color gris al valor promedio dado por los canales de la imagen.

La teoría de Retinex, indica que el máximo de cada canal debe ser ajustado a los valores máximos representables en los canales, que indican el color blanco [21].

2.6. Mapeo de Tonos

En [7] se define el mapeo de tonos como una función que prepara una imagen HDR para ser visualizada en dispositivos LDR. No existe un algoritmo universal de mapeo de tonos. Los diferentes algoritmos fueron desarrollados para satisfacer requisitos específicos de aplicaciones.

El mapeo de tonos tiene la finalidad de disminuir la brecha presente entre la percepción entre una escena del mundo real y una escena desplegada en un dispositivo de visualización.

Reinhard et ál. clasifican en [7] los algoritmos de mapeo de tonos en dos grupos: aquellos basados en la reflexión de la luz en superficies difusas (dominio de la frecuencia y de gradiente) y los que trabajan directamente sobre los píxeles (dominio espacial).

En la reducción del rango dinámico se presentan cuatro posibles enfoques:

- *Operadores globales*: Las imágenes son comprimidas utilizando una curva no lineal idéntica para todos los píxeles.
- *Operadores locales*: Reducción del rango dinámico mediante la modulación de la curva no lineal por un nivel de adaptación derivado para cada píxel, considerando sus vecinos.
- *Operadores en el dominio de la frecuencia*: Reduce el rango dinámico de las componentes de la imagen selectivamente, basado en su frecuencia espacial.
- *Operadores en el dominio del gradiente*: Modifica la derivada de una imagen para lograr la reducción del rango dinámico.

2.6.1. Algoritmo de mapeo logarítmico adaptativo

El algoritmo de mapeo logarítmico adaptativo es del tipo de operadores globales y fue desarrollado por Drago et ál. en [8]. El algoritmo realiza una compresión logarítmica de los valores de luminancia imitando la respuesta humana a la luz.

El algoritmo cumple con requisitos como evitar que el alto contraste se pierda y evitar que

la compresión sea excesiva, debido a que estos factores producen pérdida de información en la imagen resultante.

El mapeo de los píxeles se realiza con

$$L_d = \frac{0,01 L_{dmax}}{\log_{10}(L_{wmax}+1)} \frac{\ln(L_w+1)}{\ln\left(2+8\left(\frac{L_w}{L_{wmax}}\right)^{\left(\frac{\ln b}{\ln 0,5}\right)}\right)} \tag{2.17}$$

con
L_w: luminancia de cada píxel en la escena o imagen HDR.
L_{wmax}: máxima luminancia de la escena.
L_d: luminancia de cada píxel de la escena mapeada al dispositivo de visualización.
L_{dmax}: capacidad de luminancia máxima en el medio de despliegue, en monitores CRT es de $100 cd/m^2$.
b : valor de bias; da una proporción de balance de luz y oscuridad en la imagen. Para este algoritmo se utiliza $b \in [0,7; 0,9]$.

Después de ser aplicado el mapeo de tonos a los píxeles, se debe hacer una corrección gamma.

2.7. Corrección Gamma

La corrección gamma se realiza por medio de la función no lineal que se aplica a los píxeles de una imagen para compensar la no linealidad de los dispositivos de despliegue, y está dada por:

$$L_d = L_w^{\left(\frac{1}{\gamma}\right)} \tag{2.18}$$

con
L_d: valor de píxel de la imagen después de corrección.
L_w: valor de píxel de la imagen antes de la corrección.
γ: factor gamma.

2.8. Paquete de software pfstools

El paquete de software pfstools es un conjunto de programas de línea de comando para leer, escribir y manipular imágenes HDR, desarrollados en el lenguaje de programación C++. En el conjunto de herramientas básicas se encuentran pfstmo (algoritmos de mapeo de tonos, tanto globales como locales) y pfscalibration (recuperar la respuesta del sensor de captura y generar imágenes HDR a partir de imágenes LDR con múltipes exposiciones) [35].

En el presente trabajo el código y los resultados de pfstools se usan como referencia.

2.9. GStreamer: API para el manejo de multimedia

GStreamer es un marco de trabajo para aplicaciones de flujo de datos, principalmente para multimedia. Está escrito en lenguaje de programación C y se basa en la biblioteca GObject y GLib, permitiendo la programación orientada a objetos y herencia [29].

GStreamer cuenta con las siguientes caracteristicas [36]:

- Interfaz de Programación de Aplicaciones (API, del inglés Applications Programming Interfaz) para multimedia.
- Arquitectura de complementos (plug-ins).
- Arquitectura de tuberías (pipelines).
- Mecanismo de manipulación de tipos de medios y negociación.
- Más de 150 complementos.
- Conjunto de herramientas.

2.9.1. Conceptos Básicos

Elementos (Elements)

Un elemento es la unidad básica que posee una función específica como lectura/escritura de datos en un archivo, decodificación de estos datos, entre otras. Los elementos son una clase de objeto de GStreamer y pueden ser enlazados entre sí (siempre y cuando sean compatibles, dependiendo de las características que soporten), formando una tubería y logrando un flujo de datos entre estos, como se muestra en la figura 2.5.

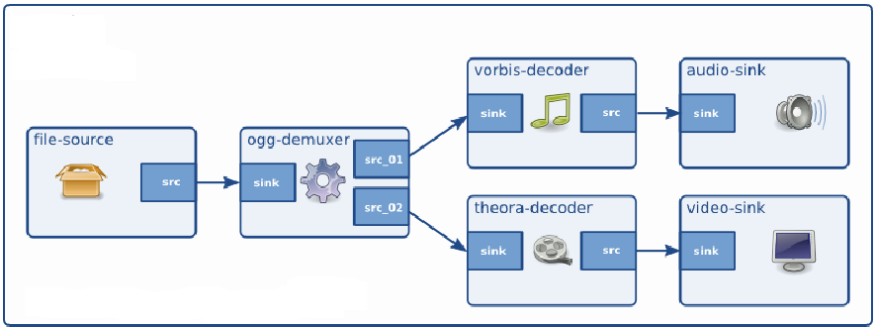

Figura 2.5: Ejemplo de una tubería de GStreamer de audio y video [36]

Los elementos pueden tener diferentes propiedades las cuales son implementadas usando el estándar de propiedades de GObject. GStreamer posee un conjunto básico de elementos. Adicionalmente a estos se pueden crear y agregar elementos con nuevas funcionalidades. Los elementos se clasifican en:

- *Elemento fuente (source)*: solamente generan datos, por ejemplo: leer un archivo o desde un controlador de sensor de captura.
- *Elemento filtro (filter)*: reciben datos y los envían una vez modificados a la salida.

No existe limitación en la cantidad de entradas y salidas. Ejemplos: decodificadores/codificadores de video y audio.
- *Elemento sumidero (sink)*: solamente reciben datos; por ejemplo, reproducir audio y video.

Contactos (Pads)

Son entradas y salidas de los elementos. Debido al tipo de elementos, existen contactos fuente y sumidero. Estos se encargan de negociar el flujo de datos entre elementos (negociación de capacidades de elementos), permitiendo el flujo de datos.

Contenedor (Bin)

Es un contenedor de una colección de elementos. La funcionalidad de un contenedor se controla mediante los elementos que lo componen.

Tuberías (pipelines)

Es un subclase especial de contenedor de alto nivel que permite enlazar varios elementos y la ejecución de estos como se muestra en la figura 2.5. Los elementos dentro de una tubería pueden cambiar su estado para lograr el flujo de datos. Cuando los elementos son creados y enlazados en una tubería requieren un cambio de estado para realizar sus funcionalidades. En GStreamer se cuenta con cuatro estados:

- *Nulo*: estado por defecto y se encarga de localizar las capacidades de los elementos.
- *Listo*: asignación de recursos globales.
- *Pausa*: abre el flujo de datos.
- *Corriendo*: inicia el procesamiento de datos.

GStreamer posee dos formas de ejecutar una tubería:

- Herramienta gst-launch-version para pruebas de tuberías y elementos.
- Implementación de una aplicación en código C, donde sean creados y enlazados los elementos en una tubería.

2.10. DBus

DBus se define como un sistema de comunicación que permite la comunicación local entre procesos [11]. Se compone de tres capas:

- *Biblioteca libdbus*: permite a dos aplicaciones conectarse entre sí e intercambiar mensajes.

- *Bus de mensajes sobre un demonio* [1] *ejecutable*: construido sobre libdbus y múltiples aplicaciones se pueden conectar. El demonio puede enrutar mensajes de una aplicación a otras.
- *Bibliotecas adaptadas o de enlace*: utilizadas en marcos de trabajo específicos como Python.

Existen dos tipos de buses: el de sesión (comunicación entre aplicaciones en la misma sesión) y el del sistema (comunicación entre sistema operativo y la sesión).

En DBus se utilizan dos tipos de aplicaciones: los servidores (escuchan solicitudes de un cliente y ponen a disposición métodos) y los clientes (se conectan a un servidor y hacen sus propios métodos). Cada servidor puede tener varios clientes, estos son direccionados por medio del nombre de cada cliente.

2.11. Marco de trabajo del procesador DSP

La plataforma BeagleBoard-xM posee dos procesadores en un SoC DM373x: un procesador ARM cortex A8 y un procesador DSP C64x+ de Texas Instruments. Esto se conoce como un núcleo heterogéneo.

Las aplicaciones de multimedia requieren de recursos de procesamiento, por lo que se debe repartir la carga entre los dos procesadores de acuerdo a las capacidades de cada uno. Adicionalmente se debe analizar la estructura y la comunicación interprocesador (IPC, del inglés *Interprocessor Communication*) para lograr optimizar el tiempo de procesamiento de aplicaciones multimedia.

2.11.1. Comunicación Interprocesador

Texas Instruments ha desarrollado paquetes de software para el uso eficiente de recursos (memoria) y comunicación entre procesadores en un SoC. Estos paquetes se describen a continuación.

DSPLink

En [19] definen DSP/BIOS Link o DSPLINK como un esquema para pasar mensajes y datos en sistemas multiprocesadores, con el procesador ARM o GPP como cliente y el procesador DSP como servidor. Entre las características o capacidades que ofrece están:

- Provee una interfaz genérica para aplicaciones.
- Esconde detalles específicos de hardware o la plataforma desde las aplicaciones.
- Oculta detalles específicos del sistema operativo del GPP desde la aplicación.

[1] Un demonio es un proceso lanzado desde otro que se ejecuta en forma continua en un segundo plano. Cuando termina el proceso que los lanzó, los demonios se interrumpen [5].

- Aplicaciones escritas en el DSPLink para una plataforma pueden correr directamente en otras plataformas o sistemas operativos, requiriendo cambios mínimos en el código de la aplicación.
- Da portabilidad a las aplicaciones.
- Permite flexibilidad a las aplicaciones de escoger y usar el protocolo más apropiado de nivel alto/bajo.
- Provee escalabilidad a la aplicación en escoger solo los módulos requeridos desde el DSPLink.

CMEM, Memoria Contigua

El CMEM es una interfaz de programación de aplicaciones o biblioteca para el manejo de bloques contiguos de memoria. El bloque de memoria compartida es reservado en memoria RAM y debe ser continuo. Las caracteristicas de CMEM son:

- Servicio de traducción de direcciones, por ejemplo de direcciones virtuales a físicas.
- Evita la fragmentación de memoria.
- El bloque de memoria física compartida es usado como búferes de datos que podrán ser compartidos con otro procesador o un acelerador de hardware.
- Asegura que grandes bloques de memoria físicamente contigua estén disponibles incluso después de que el sistema ha estado funcionando durante períodos prolongados de tiempo.

El bloque de memoria compartida o contigua es visto por el GPP a través de direcciones virtuales, mientras que el DSP las ve por medio de direcciones físicas. Esta traducción de direcciones es llevada a cabo por la función CMEM_getPhys() [18].

Interfaz XDAIS / XDM y IUNIVERSAL

XDAIS (e**X**press **D**SP **A**lgorithm **I**nteroperability **S**tandard) consiste en un conjunto de reglas y guías que aseguran la correcta convivencia de varios algoritmos dentro de un mismo sistema, independientemente de su procedencia. Para este propósito distintas interfaces de programación de aplicaciones se proporcionan al desarrollador para facilitar el cumplimiento de dichas reglas. Por ejemplo, IALG permite reservar y manejar la memoria utilizada por un algoritmo, IRES traza los lineamientos para solicitar recursos o IDMA3 define el acceso directo a memoria.

XDM (e**X**tremeDSP **D**igital **M**edia), extiende a XDAIS para casos en los que la aplicación es estrictamente de multimedia. Esta API pone a disposición interfaces designadas como VISA ("Video", "Imaging", "Speech" y "Audio") [15].

Similar a XDM, IUNIVERSAL es una API que extiende XDAIS para permitir el desarrollo de algoritmos genéricos (no multimediales) [20]. La figura 2.6 ilustra la secuencia de funciones que deben ser invocadas por un algoritmo implementado con IUNIVERSAL.

Los pasos en la figura 2.6, se describen como [15]:

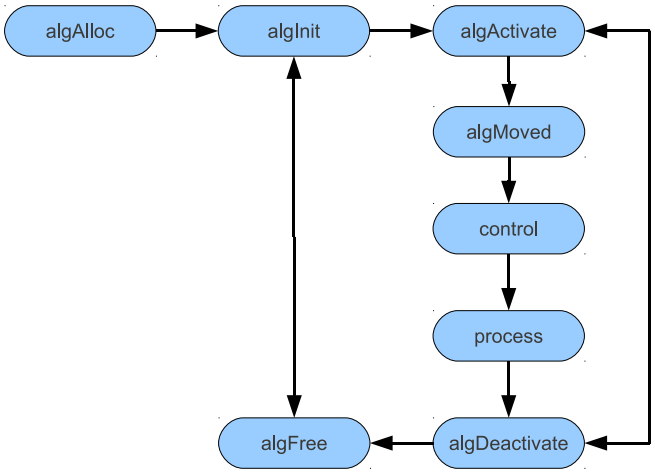

Figura 2.6: Secuencia de pasos ejecutados por un algoritmo IUNIVERSAL [15]

- *algAlloc*: especifica la cantidad de memoria que el algoritmo necesita. El programador elige entre distintos niveles de memoria disponibles en el sistema embebido, de acuerdo a las necesidades que la aplicación demande.
- *algInit*: inicializa los objetos y demás elementos, asignando los segmentos de memoria previamente reservados.
- *algActivate*: es llamada automáticamente antes de iniciar el proceso principal del algoritmo, por lo que se utiliza para copiar los datos a procesar de la memoria persistente a la temporal para aprovechar que el acceso a la segunda es hasta un factor de treinta veces más rápida que la primera.
- *algMoved*: es llamada automáticamente por la interfaz cuando se necesita mover el algoritmo a otra posición de memoria.
- *algControl*: permite modificar parámetros en tiempo real.
- *algProcess*: contiene el proceso principal del algoritmo. Para el paso de argumentos y búferes de datos IUNIVERSAL establece campos particulares en las estructuras de datos.
- *algDeactivate*: complementa a la función algActivate y se usa para trasladar los resultados procesados en la memoria temporal a la memoria persistente, de modo que estén disponibles en la siguiente llamada al proceso.
- *algFree*: permite liberar la memoria reservada por el algoritmo en algAlloc.

Codec Engine y RTSC

Codec Engine es un entorno de desarrollo diseñado para trabajar con algoritmos XDAIS encapsulados a manera de componente RTSC.

El RTSC o componentes de software en tiempo real (Real Time Software Components) es un modelo de componente de software basado en C para el desarrollo, distribución e implementación de software en tiempo real y reutilizable para diversas plataformas embebidas.

Este software es una forma de emulación de programación orientada a objetos [14].

Codec

En el marco de trabajo del DSP de Texas Instruments se incluyen especificaciones de cómo se debe empaquetar la implementación de un algoritmo para asegurar su portabilidad, y la posibilidad de incluir implementaciones para diferentes plataformas. Al paquete se le denomina en este entorno de TI "Codec", independientemente de la tarea concreta que realiza.

Para empaquetar el algoritmo se usa la herramienta RTSC, que a su vez hace uso de XDCtools. Este último es una extensión del lenguaje C y provee un conjunto de instrucciones para la creación, prueba e instalación de los componentes RTSC.

Aprovechando la estructura regular que siguen los algoritmos, se proveen al desarrollador asistentes que generan una implementación genérica (o plantilla) con las funciones XDM, siguiendo las especificaciones XDC y es empaquetado como codec. De esta forma se eliminan las tareas rutinarias. La interpretación del codec se realiza mediante la herramienta "Codec Engine" (CE), más específicamente, se utiliza "CE Algorithm Creator" [16].

Servidor

Si el desarrollo se realiza para una plataforma híbrida como los sistemas en chip (SoC) de la familia OMAP35x y BeagleBoard-xM, que contienen procesadores de propósito general y DSP, se debe crear un servidor para manejar las llamadas remotas del algoritmo. Esta herramienta se llama "Codec Server". El servidor se implementa mediante archivos de configuración XDC y por medio de asistentes se genera un servidor genérico que se modifica según corresponda. Utilizando este mismo servidor múltiples codecs pueden ser ejecutados remotamente.

Para ello CE utiliza "CE engine integrator". Todos los archivos necesarios para crear estos módulos son automáticamente creados y configurados por los asistentes disponibles [16].

Aplicación

Una vez que se cuenta con un codec y un servidor configurados adecuadamente, se procede a programar la aplicación en el lenguaje C que ejecuta el codec, en la que se instancian las funciones IUNIVERSAL y funciones especiales de CE que interacúan con los componentes RTSC.

En la aplicación se inicializan los módulos de CE y se preparan las estructuras que permiten utilizar las funciones IUNIVERSAL. Aquí automáticamente son invocadas las funciones algAlloc y algInit.

Las funciones algControl y algProcess son llamadas manualmente para efectuar los algorit-

mos programados para el DSP, las cuales internamente llaman a algActivate y algDeactivate. En la etapa de liberación de memoria, se eliminan las instancias de IUNIVERSAL y CE.

Programada la aplicación, se utiliza la herramienta "XDC Configuro" para interpretar todos los archivos de configuración escritos anteriormente en el codec y en el sevidor.

Seguidamente, el proceso de compilación cruzada y enlazado es llevado a cabo utilizando el intérprete suministrado en la cadena de herramientas de "Code Sourcery". Por último se instala la aplicación en el sistema empotrado y está lista para ser ejecutada.

2.11.2. Biblioteca TMS320C64x+ IQMath

La biblioteca IQMath contiene una colección de funciones optimizadas, lo que permite portar algoritmos de punto flotante en código de punto fijo que es ejecutado en dispositivos TMS320C64x+. Mediante la utilización de las funciones o rutinas de la biblioteca se pueden alcanzar velocidades de ejecución más rápidas que código equivalente escrito en el lenguaje ANSI C estándar [17].

Capítulo 3

Sistema de generación de HDRI con mapeo de tonos sobre la plataforma BeagleBoard-xM

El sistema de generación de imágenes con alto rango dinámico sobre el sistema embebido BeagleBoard-xM, está comprendido por varias etapas: sensor de captura, sistema de captura, módulo de HDR, módulo de mapeo de tonos y la etapa de almacenamiento y visualización de la imagen HDR con mapeo de tonos (figura 1.1). En las siguientes secciones se explican estas etapas.

3.1. Sensor de captura

El módulo de cámara adaptable al sistema BeagleBoard-xM es de la familia LI-MOD, producido por Leopard Imaging en el cual se utiliza un sensor CMOS MT9V113 con resolución VGA (640×480) creado por la empresa Aptina Imaging [12].

Algunos requisitos que debe cumplir el sensor son: habilitar y deshabilitar el balance automático de blancos (AWB) y la exposición automática (AE), y leer y escribir los registros relacionados con el tiempo de exposición. Lo anterior con la finalidad de poder capturar imágenes a diferentes exposiciones necesarias para la generación de HDR.

El sensor VGA MT9V113 cumple con los requesitos anteriores, solamente que su controlador no posee la funcionalidad para leer/escribir los registros relacionados con el tiempo de exposición y habilitar/deshabilitar el AWB y el AE.

3.1.1. Funcionalidad adicional para el controlador del sensor

La lectura/escritura de los registros del sensor se realiza por medio de la API Video for Linux Two (V4L2) y funciones de control de entrada-salida, (ioctl, *input/output control*).

En [2] se define la API V4L2 como una interfaz del núcleo de GNU/Linux, donde programar un dispositivo de V4L2 consiste de los siguientes pasos:

- Abrir el dispositivo.
- Cambiar las propiedades del dispositivo.
- Negociar el formato de datos.
- Negociar un método de entrada/salida.
- Lazo actual de entrada/salida.
- Cerrar el dispositivo.

La función "ioctl" permite acceder al sistema para solicitar operaciones específicas del dispositvo, las cuales no pueden ser expresadas por una llamada normal al sistema (ejemplo, recepción y transmisión de datos en el puerto serial). Esta función toma como parámetro el código de solicitud específico del dispositivo y utiliza descriptores de archivos.

En el sistema operativo Unix todas las entradas y salidas se realizan mediante lectura/escritura de archivos (los dispositivos periféricos son archivos en el sistema). El proceso de leer o escribir un archivo inicia mediante la operación *abrir* un archivo. El sistema verifica la existencia y los permisos del archivo, y devuelve al programa un número entero no negativo conocido como descriptor de archivo (file descriptor). Un descriptor de archivo es equivalente al apuntador de archivo utilizado en la biblioteca estándar o el manipulador de archivos (file handle) de MS-DOS [3].

Como parte de este trabajo en el controlador se ha agregado la funcionalidad para habilitar o deshabilitar el AWB y AE, escribiendo los registros para dichos propósitos, a partir de la función ioctl(), la cual se le da como parámetro el archivo descriptor resultante de la apertura del dispositivo, una estructura con la identificación de AWB o AE en V4L2, ejemplo: V4L2_CID_AUTO_WHITE_BALANCE o V4L2_CID_AUTOEXPOSURE respectivamente y la variable para la lectura/escritura del valor de registro. Además del macro VIDIOC_S_CTRL que indica escritura de registros o VIDIOC_G_CTRL para lectura.

El tiempo de exposición se determina con la relación lineal que involucra los valores de dos registros en el sensor: el tiempo de integración fino y el grueso.

$$t_{exp} = SW \cdot t_{row} - SO \cdot 2 \cdot t_{pixclk} \tag{3.1}$$

donde t_{exp} es el tiempo de exposición, SW el factor de tiempo de integración grueso, t_{row} el tiempo por fila, SO el factor de tiempo de integración fino y t_{pixclk} el tiempo por píxel.

Para agregar la funcionalidad de modificar el tiempo de exposición, se debe deshabilitar el AWB y AE, para posteriormente escribir mediante la función *ioctl* los registros del tiempo de integración fino y grueso, representados por V4L2_CID_FINE y V4L2_CID_COARSE respectivamente.

En el controlador se utilizan las funciones de lectura/escritura de registros, las cuales utilizan el protocolo I^2C para la comunicación con el sensor. También cabe resaltar que el sensor es hardware propietario.

3.2. Sistema de captura

En la figura 3.1 se muestra la secuencia de pasos diseñada en este trabajo para capturar imágenes con diferentes exposiciones, donde N es la cantidad de imágenes a capturar. El sistema de captura toma las N imágenes en forma consecutiva con una única señal de activación (disparador).

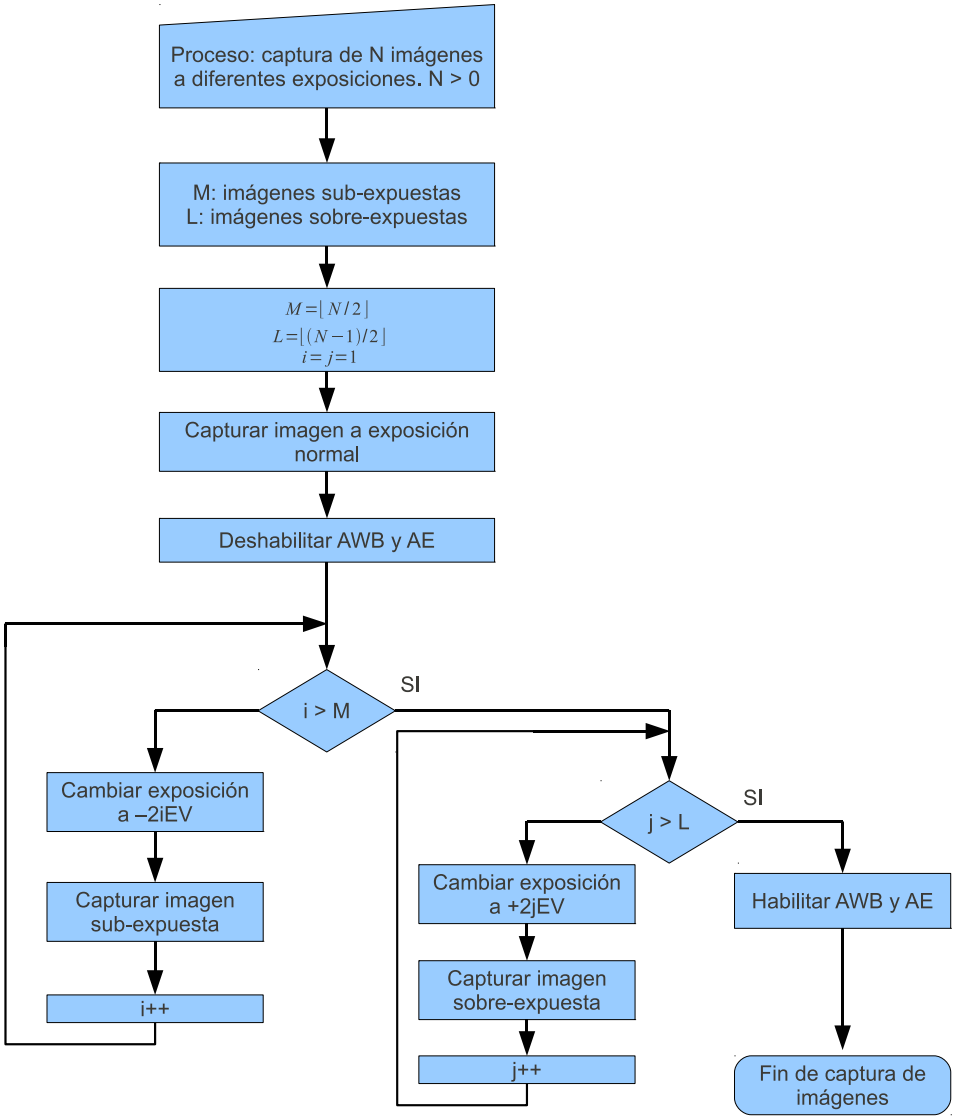

Figura 3.1: Proceso de captura de imágenes

Algunas consideraciones en el sistema de captura son:

- El sensor requiere del tiempo de procesamiento de una imagen o cuadro para aplicar los cambios realizados en los registros de tiempo de integración (relacionado directamente con el tiempo de exposición) y habilitar/deshabilitar el AWB y AE.
- Si $N = 1$, el sistema posee el comportamiento normal de una cámara estándar, donde

captura una imagen a exposición normal.
- Con $N = 2$, el sistema captura una imagen a exposición normal y una imagen sub-expuesta.
- Con $N = 3$, el sistema captura una imagen sub-expuesta, una sobre-expuesta y una a exposición normal.

El sistema de captura se ha encapsulado en este trabajo en el diseño del contenedor de GStreamer *camerabin2*, mostrado en la figura 3.2.

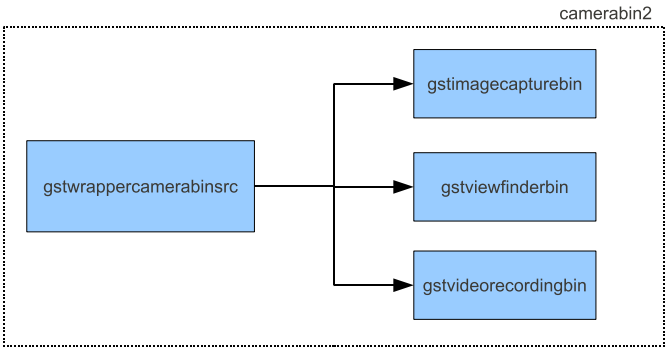

Figura 3.2: Estructura del contenedor *camerabin2*

Elementos de *camerabin2*:

- *gstwrappercamerabinsrc*: establece la comunicación con el sensor de captura, lectura/escritura de registros y pasa los búferes de datos a etapas posteriores.
- *gstimagecapturebin*: encargado de la captura de imágenes con la funcionalidad de una cámara. Las imágenes son almacenadas en diferentes formatos como: video/x-raw-yuv, jpeg, png, etc.
- *gstviewfinderbin*: permite desplegar video en un monitor.
- *gstvideorecordingbin*: encargado de grabar video en archivos con diferentes formatos como: h264, mpeg, etc.

A partir del contenedor *camerabin2* y sus elementos se diseñó la aplicación cameraApp, comprendida por: el uso del paquete de DBus para la comunicación cliente-servidor, una biblioteca desarrollada en el lenguaje de programación Vala con la funcionalidad de servidor y una interfaz para relacionar las funciones desarrolladas en la biblioteca con las que utiliza la aplicación cliente (desarrollada en el lenguaje de programación C).

En la figura 3.3 se aprecia la estructura de cameraApp, cliente-servidor, la comunicación por medio de DBus y la interfaz de interpretación de funciones (aplicación cliente en el lenguaje de programación C hace uso de las funciones de la biblioteca en el servidor programadas en el lenguaje de programación VALA).

El servidor cameraApp posee todas las funcionalidades de *camerabin2* y el cliente cameraEvents selecciona el modo de funcionamiento, sea este captura de imágenes o grabar video. En ambos casos, se tiene una salida de despliegue de video. La señal de activación para capturar imágenes o grabar video se genera a través del botón de usuario de la BeagleBoard-xM.

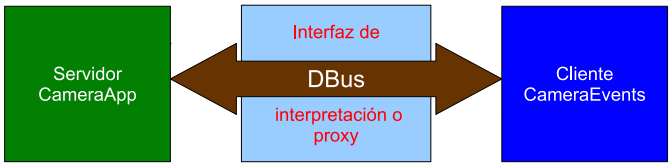

Figura 3.3: Estructura de la aplicación con cameraApp

En las funcionalidades de *camerabin2* no se incluye la posibilidad de tomar varias imágenes con un solo disparo ni almacenar los tiempos de exposición de las imágenes, por lo que estas funcionalidades se agregan modificando el gstwrappercamerabinsrc. Estas funcionalidades se detallan en las siguientes secciones.

3.2.1. Señal de activación del botón de usuario

El cliente cameraEvents posee dos hilos de ejecución, uno es del flujo de datos desde el sensor y el otro para el evento dado por */dev/input/event0* que pertenece al botón de usuario. La secuencia en la figura 3.4 muestra el proceso para generar la señal de activación para capturar imágenes/grabar video.

La señal de activación en cameraEvents permite utilizar la funcionalidad de capturar imágenes o grabar video en el servidor cameraApp, de acuerdo al modo de funcionamiento seleccionado por el usuario.

3.2.2. Funcionalidad adicional del contenedor gstwrappercamerabinsrc

La funcionalidad de capturar múltiples imágenes a partir de solo una señal de activación se logró por medio de la incorporación de una propiedad en el elemento gstwrappercamerabinsrc llamada *img_num* (images number, número de imágenes). Esta propiedad le indica al gstwrappercamerabinsrc la cantidad de imágenes que debe enviar al elemento gstimagecapturebin.

Adicional a la propiedad, se debe crear una función llamada *exposure_changes()*, para la lectura/escritura de los registros del sensor con la finalidad de cambiar las exposiciones de las imágenes y calcular el tiempo de exposición a partir del tiempo de integración fino y grueso. Los valores de tiempo de exposición de cada imagen se almacenan en un archivo de texto con el respectivo nombre de la imagen.

En la figura 3.5 se muestra el proceso para capturar tres imágenes con diferentes exposiciones considerando:

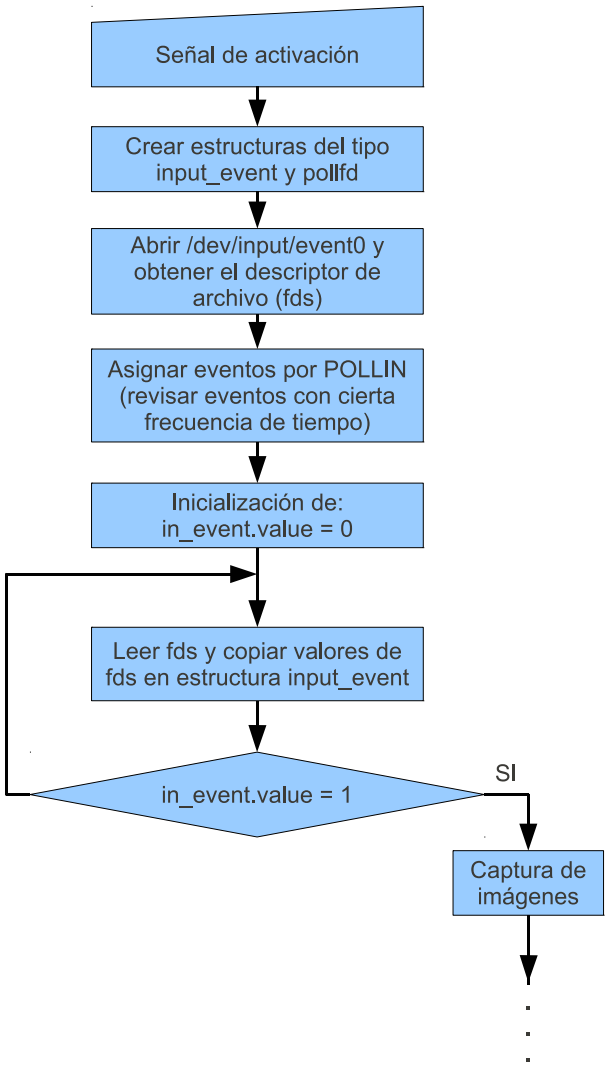

Figura 3.4: Secuencia para leer eventos en */dev/input/event0*

- Lectura/escritura de los registros del sensor.
- Cálculo y almacenamiento de los tiempos de exposición.
- Propiedad de número de imágenes a capturar.
- Señal de activación.
- Cambios de exposición a +/-2EV.
- Formato de imágenes x-raw-yuv.

Las imágenes en formato x-raw-yuv se convierten a jpeg por medio de la siguiente línea de proceso de GStreamer. Se utiliza la herramienta gst-launch en línea de comandos.

gst-launch filesrc location=image_0.yuv ! 'video/x-raw-yuv, format=(fourcc)UYVY, width=(int)640, height=(int)480, framerate=(fraction)1/1' ! dmaienc_jpeg qValue=97 ! filesink location=img0.jpg

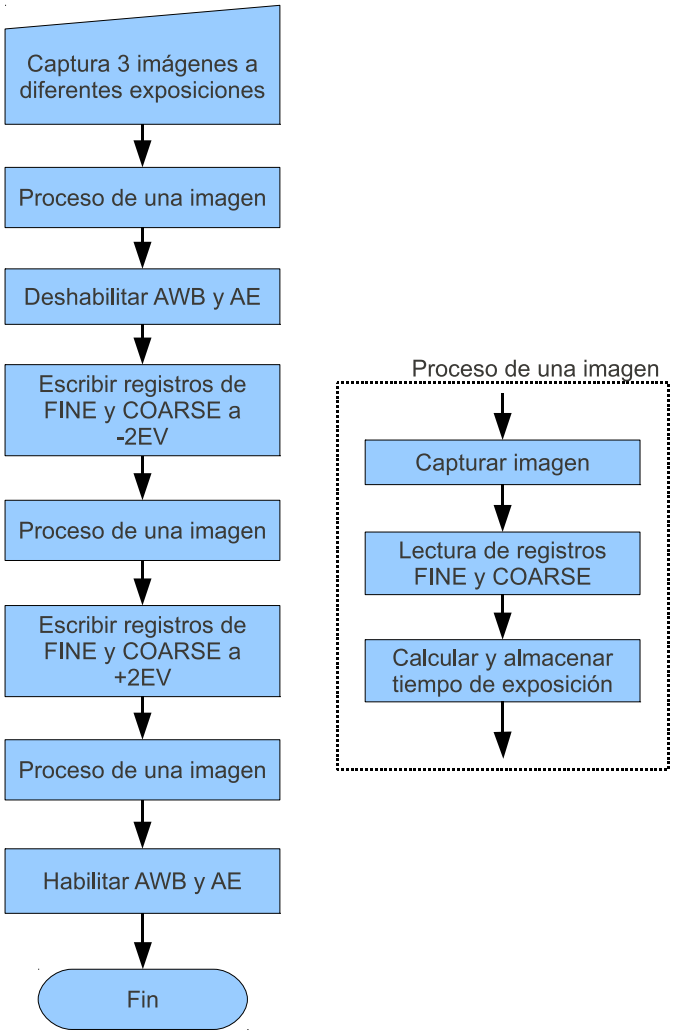

Figura 3.5: Proceso para capturar 3 imágenes a diferentes exposiciones

3.3. Módulo de HDR

La etapa de HDR contempla la generación de imágenes HDR a partir de imágenes con múltiples exposiciones con el algoritmo de Robertson, Borman y Stevenson [22], además de consideraciones como conversión entre espacios de colores y utilización de elementos de GStreamer para leer de archivos las imágenes provenientes del sistema de captura. La figura 3.6 comprende el módulo de HDR.

Este módulo se desarrolla en una aplicación de GStreamer, comprendido por el elemento *multifilesrc* y un complemento (plugin) de GStreamer. El complemento está comprendido por el almacenamiento de datos de imágenes en búferes, lectura de tiempos de exposición de un archivo, un convertidor de espacio de color YCbCr a sRGB y el algoritmo de HDR. En primera instancia el complemento se creó para ser ejecutado en el procesador ARM de la BeagleBoard-xM y posteriormente se migró a código ejecutable en el DSP.

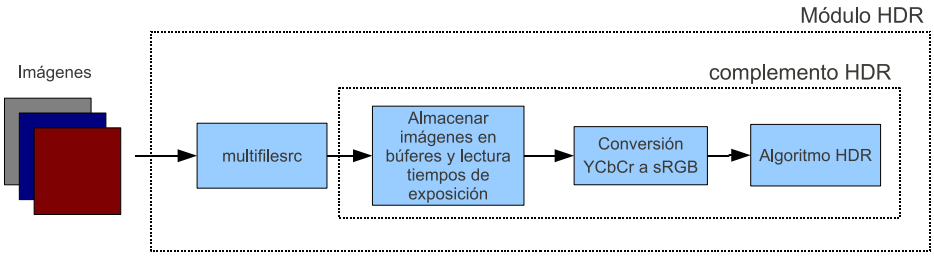

Figura 3.6: Diagrama de bloques del módulo HDR

El elemento *multifilesrc* lee las imágenes de archivos que se encuentran en formato x-raw-yuv (almacenadas por el sistema de captura) y las envia en búferes al módulo de almacenamiento. El elemento *multifilesrc* tiene propiedades como: nombre y localización de imágenes a leer.

El convertidor de espacio de color YCbCr a sRGB, es necesario puesto que el formato en que se almacenan las imágenes leidas por el elemento *multifilesrc* es x-raw-yuv, mientras que el algoritmo de HDR requiere los canales RGB.

3.3.1. Algoritmo de Robertson, Borman y Stevenson

La implementación del algoritmo de Robertson, Borman y Stevenson (módulo Algoritmo HDR en el complemento, figura 3.6) en el procesador ARM y en el DSP se basa en el paquete de software pfstools desarrollado en el Instituto Max Planck.

La calibración fotométrica del sensor de captura se lleva a cabo solo una vez (si se cambia de sensor de captura se debe realizar la calibración fotométrica con el nuevo sensor). Esta calibración requiere del algoritmo generador de HDR debido a que el proceso de aplicar la respuesta al canal se utiliza en el proceso iterativo en la determinación de la respuesta del sensor.

Algoritmo generador de HDR

El algoritmo que permite generar imágenes HDR requiere los tiempos de exposición de las imágenes de la escena, la curva o función de respuesta del sensor y una función de ponderación.

La función de ponderación se muestra en la figura 3.7. La tendencia a cero en los extremos de la curva permite atenuar o suprimir el ruido en la imagen.

La forma de la función de ponderación responde a necesidades como: eliminar el ruido, asignar pesos altos a píxeles con tiempos de exposición altos y evitar la saturación. Esta función se compone de cuatro segmentos de curva.

La secuencia de pasos para generar una HDRI a partir de una respuesta conocida del sensor, se muestra en la figura 3.8 y en la relación (2.14). A esta secuencia se le llama *aplicar función*

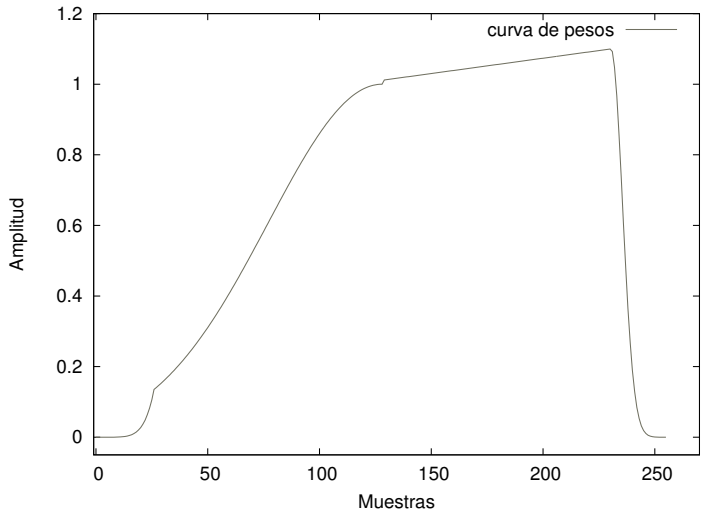

Figura 3.7: Forma de la función de ponderación

y se realiza para cada canal R, G y B por separado.

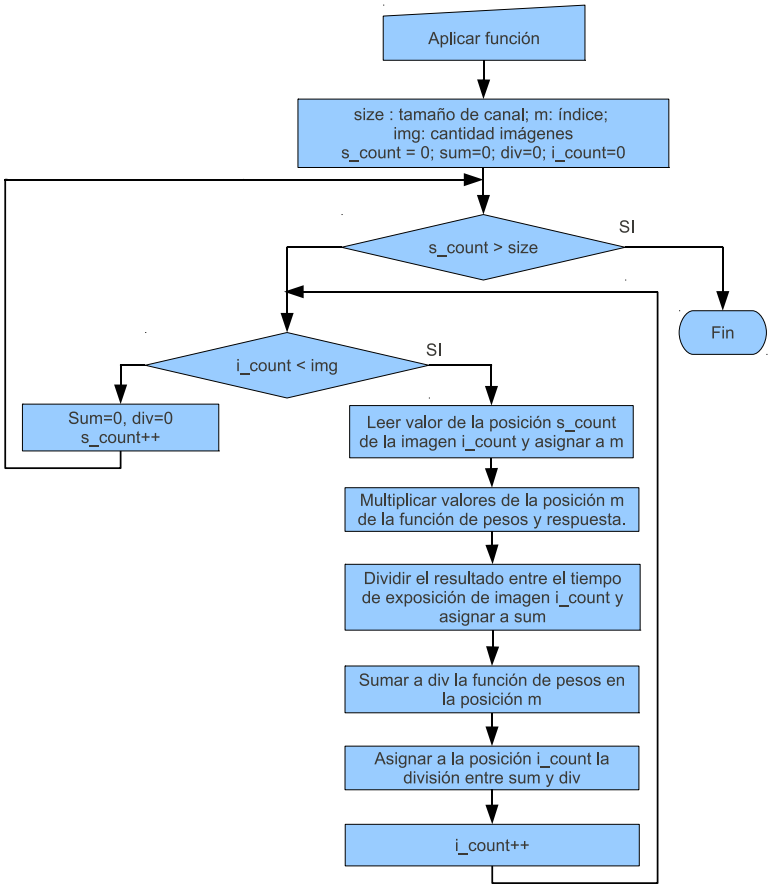

Figura 3.8: Proceso para aplicar la respuesta del sensor a los canales

La imagen HDR es el resultado de aplicar la respuesta a cada píxel de los canales R, G y B de las imágenes de entrada, para obtener las irradiancias. Si la respuesta del sensor no es

conocida, se debe estimar mediante el proceso de calibración fotométrica.

Calibración Fotométrica

La calibración fotométrica del sensor se realiza para recuperar la respuesta con que el sensor mapea las intensidades de luz que inciden sobre él a valores discretos. Se parte de la función objetivo mostrada en (2.13), la cual es una función en dos variables (I_m y x_j, curva del sensor e irradiancias, respectivamente).

En la figura 3.9 se muestra la secuencia de pasos para obtener la respuesta de los canales R, G y B. El algoritmo, primero minimiza con respecto a la curva del sensor. Además utiliza una forma de relajación de Gauss-Seidel para estimar los valores de I_m y x_j.

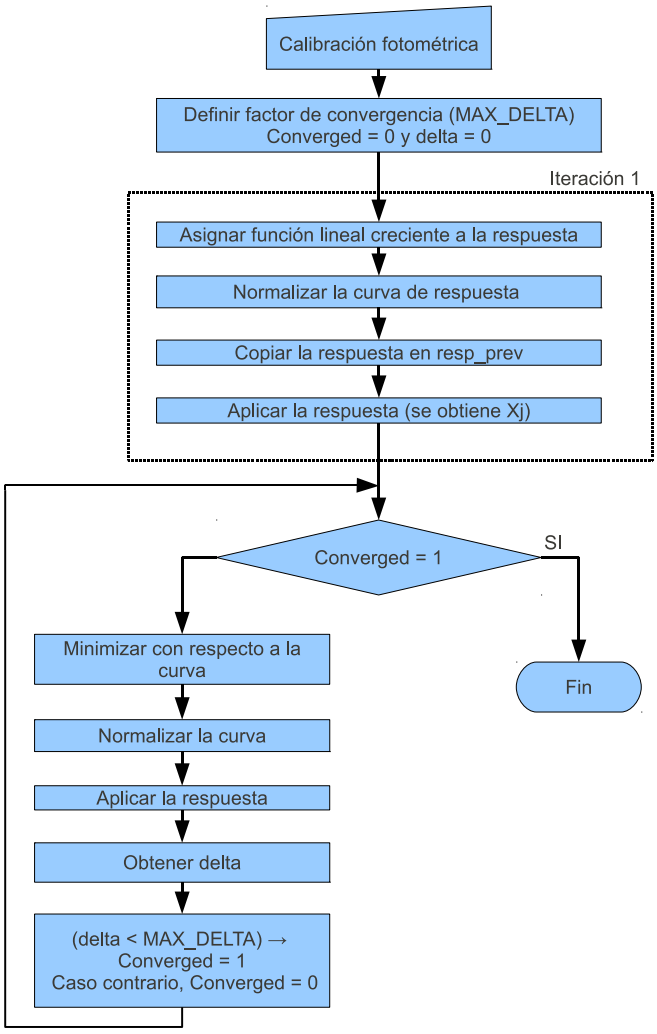

Figura 3.9: Calibración fotométrica

La primera iteración del algoritmo se realiza utilizando una curva lineal creciente, la cual permite estimar el valor de x_j. Luego se itera el algoritmo hasta que cumpla con el valor de

umbral de convergencia para estimar la respuesta del canal. El paso "*aplicar la respuesta*" corresponde al proceso de "*aplicar función*" (figura 3.9) en el algoritmo generador de HDR.

La respuesta del sensor es dada por el promedio de las respuestas de los canales R, G y B, como se muestra en la figura 3.10.

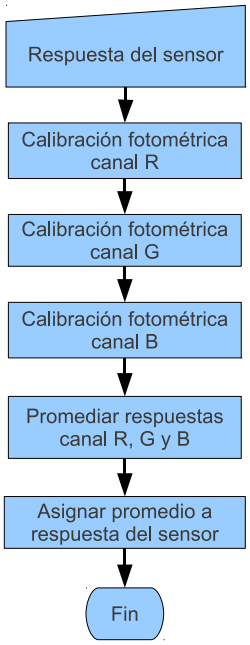

Figura 3.10: Proceso para obtener respuesta del sensor

Para visualizar la imagen HDR en dispositivos de despliegue, es necesario hacer una reducción de contraste mediante un operador de mapeo de tonos.

3.4. Módulo de mapeo de tonos

El módulo de mapeo de tonos esta conformado por las etapas que se muestran en la figura 3.11.

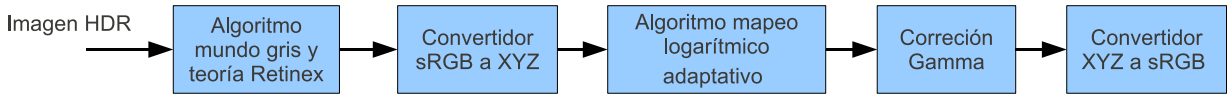

Figura 3.11: Diagrama de bloques del módulo de mapeo de tonos

3.4.1. Algoritmo de mundo gris y teoría de Retinex

La imagen HDR o de irradiancias se encuentra dividida en tres canales R, G y B. Si los valores máximos de cada canal difieren, hay problemas con regiones blancas. Caso similar ocurre con los valores promedio de cada canal, provocando problemas con el color gris en la imagen. Estos problemas son corregidos por medio del algoritmo de mundo gris y la teoría Retinex.

Los algoritmos de mundo gris y de la teoría Retinex son incompatibles debido a las condiciones de las que se parten no se pueden satisfacer de forma conjunta. En el presente trabajo se utiliza por defecto el algoritmo de mundo gris, con la posibilidad de seleccionar el algoritmo de la teoría Retinex mediante la propiedad *normaliza* del complemento HDR.

- $normaliza = 0$: aplicación del algoritmo de mundo gris.
- $normaliza = 1$: aplicación del algoritmo de la teoría Retinex.

Algoritmo de mundo gris

Permite corregir el problema del color gris en la imagen causado por la diferencia de promedios en los canales. La secuencia de pasos de este algoritmo se muestra en la figura 3.12.

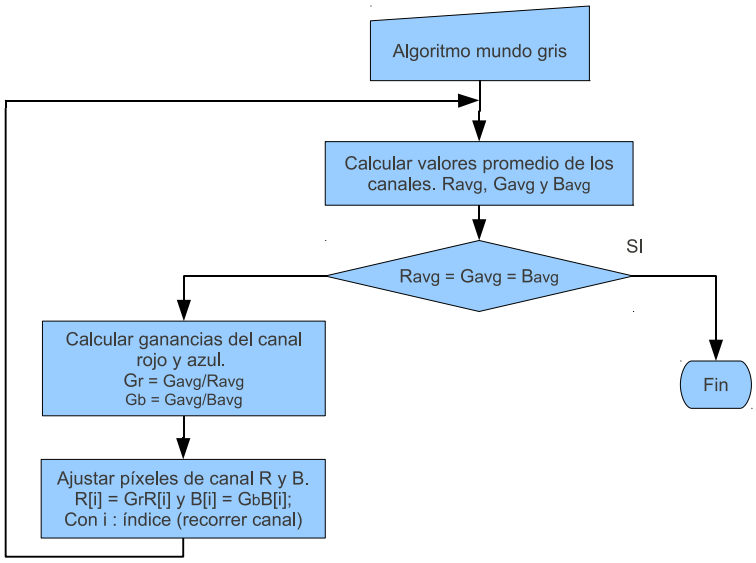

Figura 3.12: Pasos del algoritmo de mundo gris

Teoría Retinex

Permite corregir el problema con regiones blancas en la imagen causadas por la diferencia de los valores máximos de cada canal. La secuencia de pasos de este algoritmo se muestra en la figura 3.13.

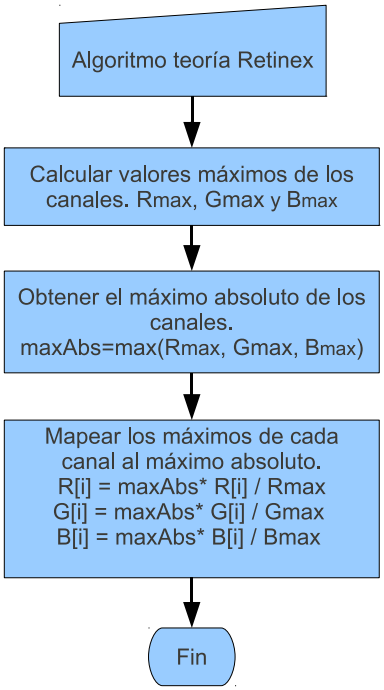

Figura 3.13: Pasos del algoritmo de la teoría Retinex

La compresión o reducción logarítmica de los valores de luminancia de la imagen HDR es llevaba a cabo por el operador de tonos logarítmico adaptativo.

3.4.2. Algoritmo de mapeo logarítmico adaptativo

El operador de tonos se aplica a los valores de luminancia de la imagen. Un convetidor del espacio de color RGB a XYZ permite tener un canal de luminancias que puede ser mapeado con el operador de tonos logarítmico adaptativo, cuyo proceso se muestra en la figura 3.14.

A partir del mapeo de tonos de la luminancia de la imagen HDR, los valores del canal X y Y están dados por

$$\begin{aligned} f_{scale} &= \frac{l_{out}}{l_{in}} \\ X &= X f_{scale} \\ Z &= Z f_{scale} \end{aligned} \tag{3.2}$$

donde f_{scale} es el factor de escala.

La correción gamma se aplica a los canales mapeados para ajustar los valores de los píxeles a la no linealidad de los dispositivos de visualización. Esta se muestra en (2.18). Se realiza antes de la conversión del espacio de color XYZ a RGB.

Las etapas del mapeo de tonos se agregan al complemento HDR y se utiliza un módulo

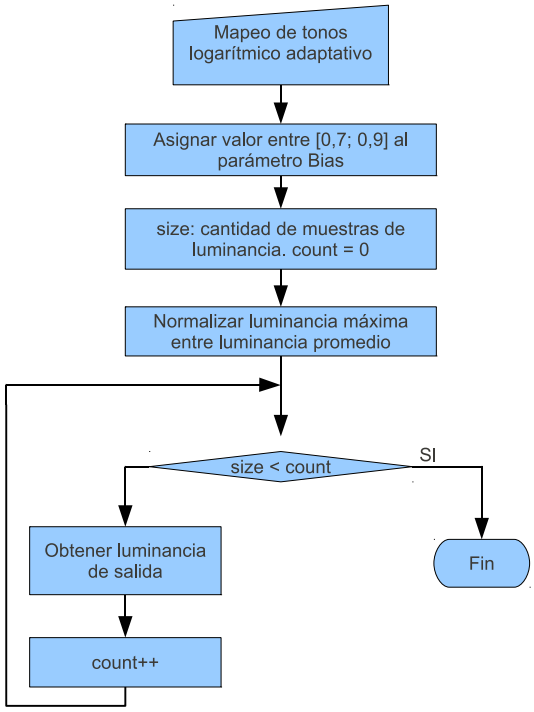

Figura 3.14: Pasos del algoritmo mapeo de tonos logarítmico adaptativo

convertidor de sRGB a YCbCr, como se muestra en la figura 3.15.

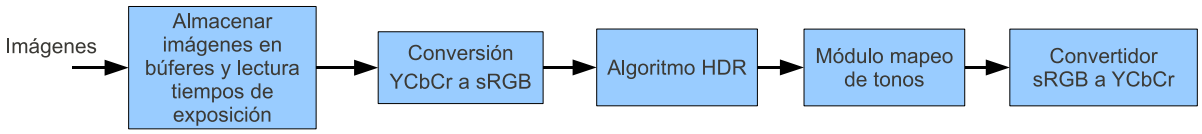

Figura 3.15: Diagrama de bloques del complemento HDR con la integración de las etapas del módulo mapeo de tonos

Al complemento se le restringen las capacidades a dos formatos de entrada de imágenes: *x-raw-yuv* y *x-raw-rgb*. Los convertidores de YCbCr a sRGB y viceversa, se utilizan en el caso de que el formato de entrada sea *x-raw-yuv*.

En las siguientes secciones se presenta la implementación propuesta para el complemento HDR en el ARM (gsthdrimagearm) y en el DSP (gsthdrimagedsp).

3.5. Implementación del complemento gsthdrimagearm

Cada imagen obtenida por el sistema de captura para la generación de HDR con mapeo de tonos es leída mediante el elemento de GStreamer *multifilesrc*, enviada al complemento y

escrita en un archivo con formato jpeg, png, etc.

El proceso anterior se realiza mediante la creación de una aplicación de GStreamer que hace uso de elementos como: *multifilesrc*, *multifilesink*, *dmaienc_jpeg* y el complemento, mostrado en la figura 3.15.

Cada imagen que es leída y enviada al complemento se almacena en una estructura (inbufs). La función *hdrimage_algorithm()* que realiza el procesamiento de los algoritmos recibe la estructura y parámetros como: cantidad de imágenes, tiempos de exposición, factor gamma, paŕametro Bias, tamaño de imagen, propiedades que indican si es necesaria la calibración, aplicar el modelo de mundo gris y el formato de entrada de las imágenes (YUV o RGB).

En la estructura *hdr* se encuentran los búferes, matrices y variables para el procesamiento de los algoritmos de HDR y mapeo de tonos:

- *exposureData*: matriz tridimensional comprendida por tres canales, cada uno con la cantidad de imágenes de entrada dadas por la propiedad *imgCount* y los datos de cada canal.
- *choutbuf*: matriz bidimensional, comprendida por los canales del búfer de salida y sus datos.
- *rcurve*: matriz bidimensional comprendida por tres canales R, G y B, y los valores de la respuesta del sensor de cada canal.
- *ti*: búfer que contiene el tiempo de exposición de las imágenes de entrada.
- *rcurve_prev*: búfer para el procesamiento en la calibración fotométrica, que almacena la respuesta anterior.
- *rcurve_filt*: búfer utilizado en la normalización de la curva del sensor. Corresponde al filtro de la curva de respuesta.
- *weights*: búfer para la función de ponderación.
- *sum*: búfer acumulador para el procesamiento en la función *applyResponse*.
- *cardEm*: búfer acumulador para el procesamiento en la función *applyResponse*. Representa la cardinalidad de E_m.
- *to_sort*: búfer utilizado en la normalización de la curva del sensor.
- *channel*: matriz bidimensional comprendida por tres canales y sus datos. Utilizado en la separación de canales de los búferes de entrada.
- *lum_in*: búfer que almacena el canal de luminancia de la imagen HDR, para el procesamiento de mapeo de tonos.
- *lum_out*: búfer que almacena los valores de luminancia obtenidos a partir del mapeo de tonos.
- *imgsCount*: variable que contiene la cantidad de imágenes de entrada del sistema.
- *sizeRGB*: variable que contiene el tamaño de los canales.
- *input_level*: variable para la cantidad de niveles representables en las imágenes de entrada.
- *input_multiplier*: variable para multiplicar los datos de entrada, mapeándolos al rango entre 0 y *input_level*.
- *gauss*: variable ponderada de *input_multiplier*, para la asignación de pesos en el cálculo de la función *weights*.

- *max_response*: valor máximo de los datos de entrada.
- *min_response*: valor mínimo de los datos de entrada.
- *to_sort_size*: variable utilizada en el cálculo de la función *weights*, para el cálculo de los bordes de la función.

La estructura *hdr* y sus miembros (búferes y matrices) son reservados en memoria dinámica mediante la función *malloc()* e inicializados.

Considerando el formato UYVY para las imágenes de entrada, los canales de U y V se encuentran submuestrados. Además cada búfer que contiene los datos de la imagen se debe separar en canales de Y, Cb y Cr. Este proceso de separar los datos en canales se muestra en la figura 3.16 y cada canal es almacenado en la matriz *exposureData*.

Los canales poseen el mismo tamaño con la finalidad de reutilizarlos en los módulos de conversión de espacios de colores. El canal U y V al estar submuestreados, presentan la mitad de los datos en comparación al canal Y.

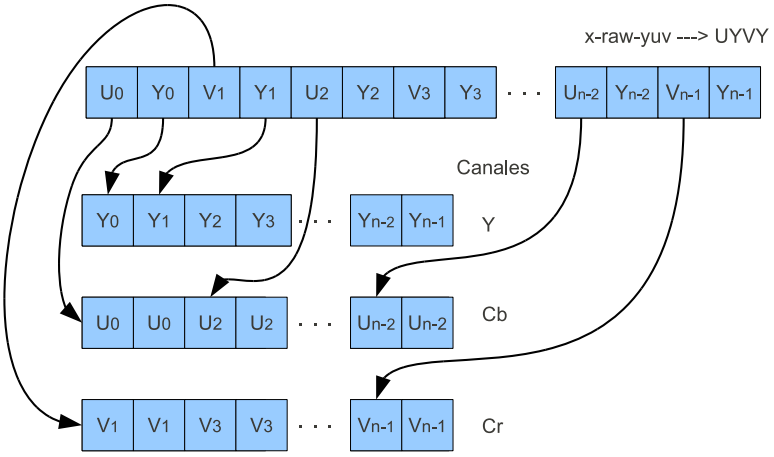

Figura 3.16: Separación del formato UYVY en canales Y, Cb y Cr

Los datos en los canales U y V se almacenan de uno por medio, esto debido a que en la conversión de YCbCr a sRGB se deben obtener valores de R, G y B, a partir de datos submuestrados (canales U y V) y el canal Y, utilizando la matriz *channel* para la lectura de los canales YCbCr y escritura de los datos de sRGB, como se muestra en la figura 3.17.

La función de ponderación se obtiene mediante la función *weightsComposite* y es almacenada en el búfer *weights*. La propiedad *calibrat* indica al sistema la aplicación de la calibración fotométrica al sensor.

La propiedad *calibrat* da dos casos:

- *calibrat* = 0: La respuesta del sensor es cargada desde un archivo de texto llamado *respCurve.txt* al búfer *rcurve*.
- *calibrat* = 1: La respuesta del sensor se obtiene mediante la calibración fotométrica y

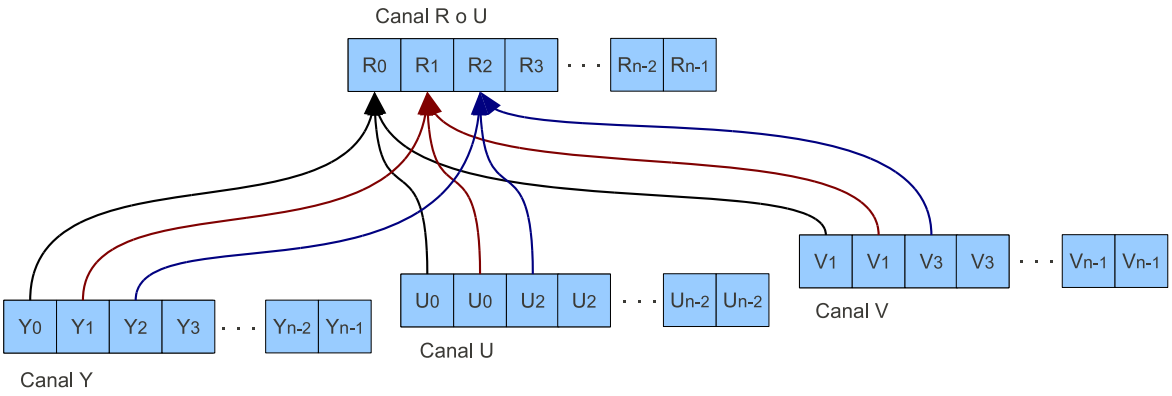

Figura 3.17: Conversión de los canales YCbCr al canal R

ponderación de las respuestas de cada canal. La respuesta que se encuentra en el búfer *rcurve* es almacenada en el archivo de texto *respCurve.txt*.

La calibración fotométrica la realiza la función *getResponse()*, la cual tiene como parámetro la estrucutra *hdr*. Esta función desarrolla un método iterativo de la forma de relajación Gauss-Seidel, además utiliza funciones como *normalize_rcurve()* y *applyResponse()* en el proceso de convergencia.

La función *applyResponse()* se encarga de aplicar la respuesta por canal para generar la imagen HDR. Esta función recibe como parámetros los búferes *rcurve*, los canales de las imágenes de entrada pertenecientes al mismo tipo (ejemplo: canal rojo de las imágenes de entrada), el tamaño de los canales y un búfer de salida (*choutbuf*) que contendrá los datos HDR del canal.

La función *toneMapOperator()* ofrece la funcionalidad de mapeo de tonos. Esta función tiene como parámetros los canales de la imagen HDR, el factor gamma, el valor de Bias, el tamaño de los canales y la propiedad *gwa* que indica la aplicación del algoritmo de mundo gris.

La imagen HDR con mapeo de tonos es la composición de los canales R, G y B. Estos canales se deben convertir al espacio de color YCbCr. La salida del complemento es un búfer con el formato UYVY. El proceso que muestra la creación del búfer en este formato a partir de los canales Y, Cb y Cr, se aprecia en la figura 3.18.

Los valores de índice impar en el canal U e índice par en el canal V (color rojo en la figura 3.18) no se consideran en la creación del búfer de salida del complemento, esto debido al submuestreo en los canales U y V del formato UYVY.

La imagen HDR con mapeo de tonos almacenada en el búfer de salida (*outbuf*) es enviada al elemento *multifilesink* de la aplicación *hdr_testArm*.

El complemento *gsthdrimagearm* y la aplicación *hdr_testArm* se desarrollan en el lenguaje de programación C, bajo el entorno de desarrollo del SDK de RidgeRun, haciendo uso de paquetes como autotools para garantizar la portabilidad del código. También los accesos a memoria se realizan por medio de punteros para optimizar el tiempo de ejecución.

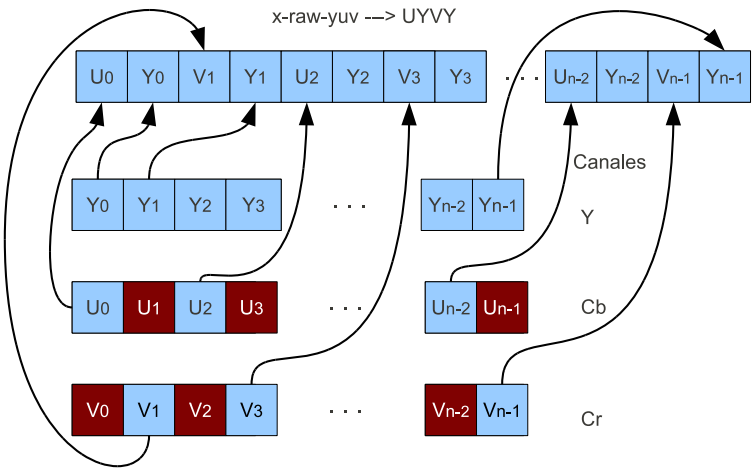

Figura 3.18: Separación del formato UYVY en canales Y, Cb y Cr

En el sistema de generación de imágenes HDR con mapeo de tonos, el complemento y la aplicación son de punto flotante debido a la naturaleza de los algoritmos de HDR y mapeo de tonos. Además una imagen HDR es una representación digital de píxeles con valores de tipo flotantes.

3.6. Implementación del complemento gsthdrimagedsp

El complemento *gsthdrimagedsp* se crea para ser ejecutado en el procesador DSP, conformado por los módulos mostrados en la figura 3.15.

La forma de desarrollar aplicaciones para el procesador de DSP es diferente a la forma en que se programa en el procesador ARM o en una computadora de escritorio [23]. La creación de un codec, un servidor y una aplicación cliente, es una forma de programar para un procesador DSP, haciendo uso de las bibliotecas de intercomunicación de procesadores.

Algunas restricciones que se encuentran en la programación del DSP son:

- No se permite el manejo de archivos, lectura/escritura, etc.
- Modo de reservar y liberar memoria mediante CMEM.
- Uso de la capa iUniversal. Búferes de XDM.
- Paso de párametros se realiza mediante extensión de estructuras de XDM.
- Cantidad limitada de búferes de XDM, 16 búferes de entrada, 16 de entrada-salida y 16 de salida.
- No se pueden utilizar funciones del estándar ANSI de entrada/salida, ejemplo la función printf.
- Opciones de depuración limitadas, GTrace, CE_Debug, etc.
- Limitaciones de memoria.

- Procesador de punto fijo.
- Tipo de datos en los búferes XDM.

La implementación del complemento en el procesador de DSP se da primeramente en punto flotante (migrar el código del complemento *gsthdrimagearm*) al DSP, como métricas de comparación, para posteriormente pasar ciertas funciones a punto fijo y analizar rendimiento en punto fijo.

El desarrollo de la aplicación en el procesador DSP se divide en tres secciones: aplicación cliente, servidor y codec. El complemento *gsthdrimagedsp* se implementa en la aplicación cliente, la cual mediante el servidor establece la comunicación con el codec.

3.6.1. Aplicación cliente

En la aplicación cliente se establece la comunicación entre los procesador ARM y DSP, mediante el uso de la API de iUniversal, pasando parámetros y búferes hacia el codec.

El codec posee estructuras extendibles para los argumentos de entrada *IHDR_InArgs* y salida *IHDR_OutArgs*, una estructura de parámetros *IHDR_Params*, una de parámetros dinámicos *IHDR_DynamicParams* y una de estado *IHDR_Status*. Estas estructuras se utilizan en el espacio de la aplicación cliente con la finalidad de pasar datos al codec. La estructura objeto *hdr* se utiliza solo en el espacio de codec.

La estructura *IHDR_Params* se extiende con los parámetros: calibrat, gwa, imgCount, luminance, biasVal y gammafact. En la aplicación cliente se crea una estructura del tipo *IHDR_Params* , a la cual se le asignan los valores dados por el usuario en las propiedades del complemento o valores obtenidos en el proceso.

La función *UNIVERSAL_create()*, permite pasar la estructura de parámetros al codec, invocando las funciones *algAlloc* y *algInit* de la API de iUniversal. En la función *algInit* se inicializa la estructura parámetros del codec con los datos enviados a través de *UNIVERSAL_create()*. La función *algAlloc* reserva memoria para el objeto de la estructura *hdr* del codec.

La secuencia de pasos que permiten extender la estructura *IHDR_Params* y pasarlos al codec mediante la función *UNIVERSAL_create()*, se aprecia en la figura 3.19.

En el caso del complemento *gsthdrimagearm*, los búferes de entrada son almacenados en la estructura *inbufs*. Para el complemento *gsthdrimagedsp* los búferes de entrada son almacenados en búferes del tipo XDM, mediante la función *Memory_contigAlloc()*.

La función *Memory_contigAlloc()* reserva memoria de forma dinámica en la memoria compartida del procesador ARM y DSP. Los búferes XDM son pasados como parámetros al codec por medio de la función *process()* de la API de iUniversal. Esta función tiene como parámetros los búferes XDM de entrada, salida y entrada-salida. Además el objeto del codec y las estructuras de argumentos de entrada y salida.

La aplicación *hdrtest_dsp* está comprendida por el elemento *multifilesrc*, el complemento

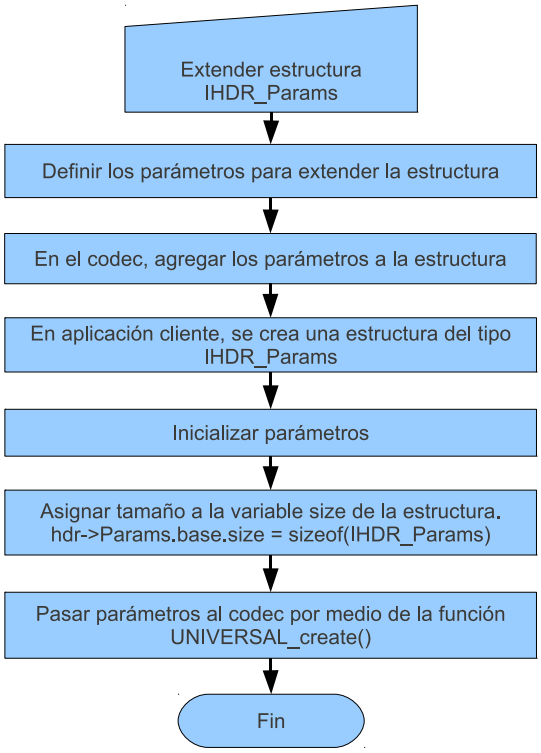

Figura 3.19: Proceso para pasar parámetros al codec mediante la estructura *IHDR_Params*

gsthdrimagedsp y el elemento *multifilesink*. El elemento *multifilesrc* se encarga de leer las imágenes y enviarlas al complemento; estas imágenes se almacenan en búferes de XDM. La cantidad de búferes de entrada XDM a utilizar está delimitada por la cantidad de imágenes de entrada del sistema, es decir, por la propiedad *imgCount*.

Los tiempos de exposición de las imágenes de entrada son leídos de un archivo de texto. Estas deben pasarse al codec por medio de un búfer de entrada de XDM.

Los búferes XDM son del tipo *int8* (entero con signo de 8bits). Los datos de entrada de las imágenes son del tipo *uint8* (entero sin signo de 8bits). En el caso del búfer de tiempos de exposición, los datos son del tipo *float* (flotante), por lo que se debe coercionar el puntero del búfer de tiempos de exposición a *(XDAS_Int8 *)* y asignarlo al búfer de entrada de XDM.

La recuperación de los datos de tiempos de exposición en el codec se realiza coercionando el puntero del búfer XDM al tipo (*float **) y asignándolo al búfer *ti* de la estructura *hdr*. Este proceso se muestra en la figura 3.20.

La cantidad de búferes de entrada de XDM está dada por la cantidad de imágenes de entrada más uno (búfer de tiempos de exposición). Cabe resaltar que la cantidad máxima disponible de búferes de entrada XDM es 16.

El búfer de salida con la imagen HDR con mapeo de tonos se asigna a un búfer de XDM de salida. Esta a su vez se copia a un búfer de GStreamer *outbuf*, el cual es la salida de la aplicación cliente. El búfer *outbuf* es enviado al elemento *multifilesink* de la aplicación

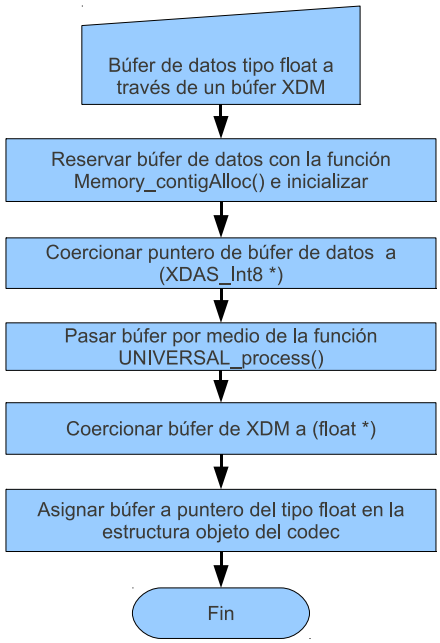

Figura 3.20: Proceso para pasar datos tipo float a través de un búfer XDM

hdrtest_dsp para ser escrito en un archivo en el espacio de color YCbCr en el formato UYVY.

La respuesta del sensor es escrita o leída dependiendo de la propiedad *calibrat*. La respuesta se asigna a un búfer de entrada-salida de XDM. Cuando la propiedad indica lectura el búfer de XDM se comporta como un búfer de entrada y en el caso de escritura, el comportamiento es de un búfer de salida de XDM. La lectura/escritura de la respuesta del sensor en el archivo *respCurve.txt* se lleva a cabo en la aplicación cliente. El búfer de la respuesta del sensor posee valores tipo *float*. Casos dados por la propiedad:

- *Lectura*: se debe coercionar el puntero del búfer de respuesta a (*XDAS_Int8 **) y asignarlo al búfer de entrada-salida de XDM. Para recuperar los datos del búfer de respuesta en el espacio del codec se debe coercionar el puntero del búfer de entrada-salida de XDM a tipo (*float **) y asignarlo al canal R de la matriz *rcurve*. La lectura de la respuesta desde el archivo de texto *respCurve.txt* se realiza mediante la función *loadRespFunc*.
- *Escritura*: se realiza la calibración fotométrica y ponderación de la respuesta de los canales en el codec para obtener la respuesta con valores tipo *float*. La respuesta se asigna al canal R de la matriz *rcurve*, el puntero a este canal se coerciona al tipo (*XDAS_Int8 **) y se asigna al búfer de entrada-salida de XDM. En la aplicación cliente el puntero al búfer de entrada-salida se coerciona al tipo (*float **) con la finalidad de recuperar los datos. Estos datos son almacenados en el archivo de texto *respCurve.txt* mediante funciones de manejo de archivos.

El proceso de lectura/escritura de la respuesta en la aplicación cliente se observa en la figura 3.21.

En la figura 3.22 se muestra un diagrama de bloques con las funciones que intervienen en el

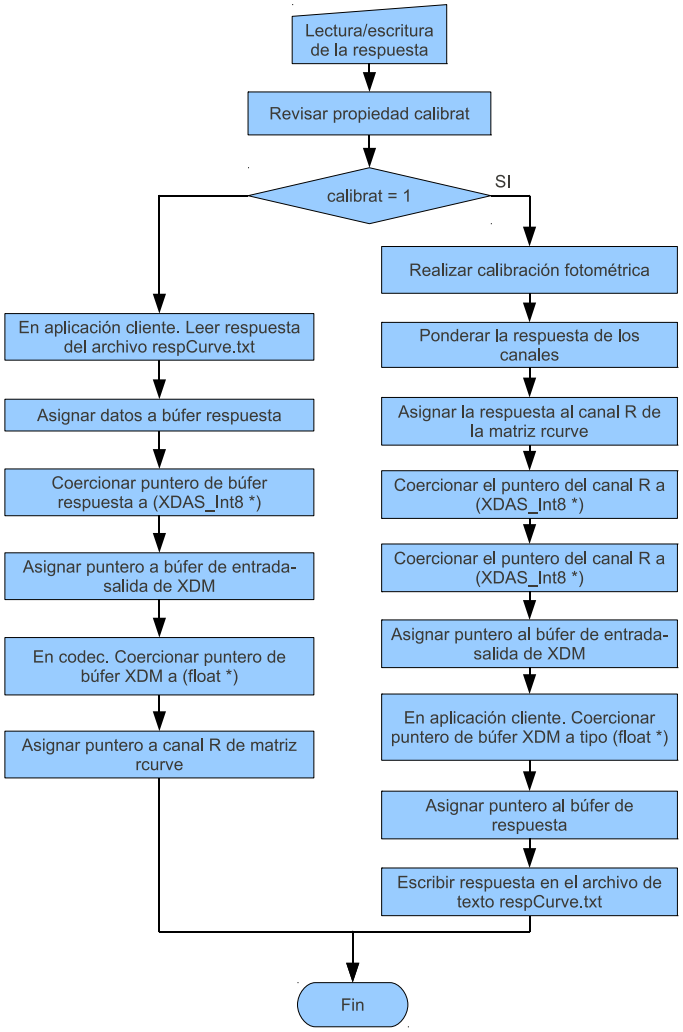

Figura 3.21: Proceso para la lectura/escritura de la respuesta del sensor

paso de parámetros de la aplicación cliente al codec.

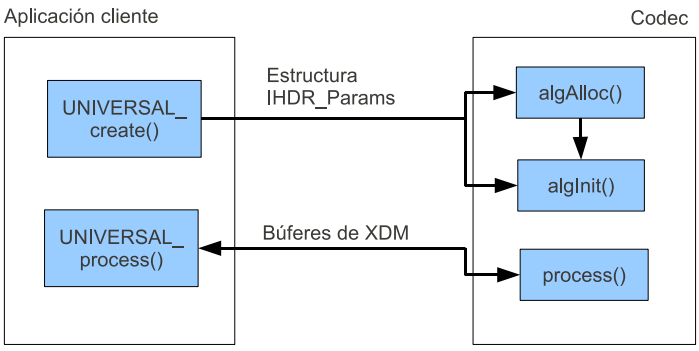

Figura 3.22: Funciones que intervienen en el paso de parámetros de la aplicación cliente al codec

La formas de reservar memoria de acuerdo a la API de iUniversal utilizan la funciones:

- *memtab*: tabla de memoria (memory table). Permite reservar memoria para búferes unidimensionales en el segmento de memoria DDR2 del procesador de DSP. La función *memtab* se utiliza en la función *ialgAlloc* del codec.
- *Memory_contigAlloc*: permite reservar en el segmento de memoria compartida. La reserva de memoria se realiza desde la aplicación cliente y los datos son enviados al codec mediante los búferes de XDM.

Los búferes de XDM son unidimensionales, por lo que las matrices utilizadas en el procesamiento de los algoritmos de HDR y mapeo de tonos se descomponen en búferes unidimensionales para utilizar la función *memtab* o *Memory_contigAlloc()* en la reserva de memoria. Las limitaciones de las funciones para reservar memoria son:

- *memtab*: utiliza un segmento de memoria limitado de 6MiB, desde la dirección $0 \times 87A00000$ a 0×88000000. Con $1MiB = 2^{20}$ bytes.
- *Memory_contigAlloc()*: utiliza el segmento de memoria compartida, CMEM, el cual es de 30MiB de la dirección 0×88000000 a $0 \times 89E00000$. Cantidad de búferes de entrada y entrada-salida de XDM limitado a un máximo de 32.

En la figura 3.23 se presenta el mapeo de memoria por defecto del SDK de RidgeRun en el sistema BeagleBoard-xM. El total de la memoria es de 512MiB.

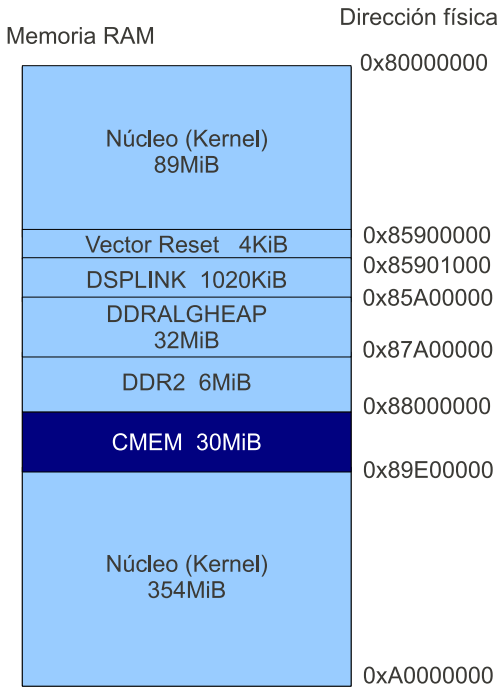

Figura 3.23: Mapeo de memoria del sistema BeagleBoard-xM usando el SDK

La cantidad de memoria requerida por las matrices y búferes de procesamiento de los algoritmos, considerando que la resolución de imágenes de entrada es de 640×480 (614400 bytes) y la cantidad de niveles (*input_level*) es de $2^8 = 256$, es:

- *exposureData*: matriz tridimensional de *tres canales* $\times$ *imgCount* $\times$ *sizeRGB* del tipo

float, 10, 55MiB.
- *choutbuf*: matriz bidimensional de *tres canales* × *sizeRGB* del tipo *float*, 3, 52MiB.
- *rcurve*: matriz bidimensional de *tres canales* × *input_level* del tipo *float*, 3KiB.
- *rcurve_prev*: búfer de tamaño *input_level* del tipo *float*, 1KiB.
- *rcurve_filt*: búfer de tamaño *input_level* del tipo *float*, 1KiB.
- *weights*: búfer de tamaño *input_level* del tipo *float*, 1KiB.
- *sum*: búfer de tamaño *input_level* del tipo *float*, 1KiB.
- *cardEm*: búfer de tamaño *input_level* del tipo *uint16_t*, 0, 5KiB.
- *to_sort*: búfer de tamaño *to_sort_size* del tipo *float*, 12B.
- *channel*: matriz bidimensional de *tres canales* × *sizeRGB* del tipo *float*, 3, 52MiB.
- *lum_in*: búfer de tamaño *sizeRGB* del tipo *float*, 1, 17MiB.
- *lum_out*: búfer de tamaño *sizeRGB* del tipo *float*, 1, 17MiB.

La cantidad total de memoria requerida por los búferes y matrices de procesamiento para los algoritmos es 19, 01MiB. La cantidad de memoria que permite la función *memtab* no es suficiente para los búferes y matrices. Se utiliza la función *Memory_contigAlloc()* para la reserva de estos.

Un aumento en la resolución o en la cantidad de imágenes de entrada produce un aumento en la cantidad de memoria requerida; por ejemplo: incrementar en uno la cantidad de imágenes de entrada mateniendo la resolución requiere un adicional de memoria de 3, 52MiB.

En el caso de un aumento de la resolución a 1024×768 manteniendo la cantidad de imágenes de entrada requiere 31, 07MiB de memoria adicional. Con la resolución de 1024×768, un incremento de una imagen de entrada equivale a 16, 45MiB de memoria adicional.

El segmento de memoria compartida se amplia de 30MiB a 160MiB como se muestra en la figura 3.24, con la finalidad de disponer de un espacio mayor de memoria para la aplicación de generación de imágenes HDR con mapeo de tonos.

Veintiseis son los búferes XDM requeridos para pasar los datos de los búferes y matrices de procesamiento, considerando tres imágenes como la cantidad de entrada del sistema. Adicionalmente se tienen tres búferes para las imágenes de entrada, uno para la respuesta del sensor y uno para los tiempos de exposición, para un total de 31 búferes. Esto queda próximo a los 32 búferes de XDM disponibles. En el caso de aumentar las imágenes de entrada en uno, se requieren adicionalmente tres búferes XDM. El sistema queda limitado a tres imágenes de entrada.

Un objeto que contiene la estructura con los búferes y matrices de procesamiento de los algoritmos permite eliminar la limitación dada por la cantidad de búferes XDM. La estructura *IHDR_Process* se utiliza en la aplicación cliente y en el codec. El puntero de la estructura se pasa por medio de un búfer de XDM, coercionando al *tipo XDAS_Int8* y en el codec se coerciona al tipo de la estructura para recuperar los datos.

Algunas consideraciones en la creación de la estructura (reserva de memoria) en la aplicación cliente son:

- El procesador DSP ve el segmento de memoria como direcciones físicas.

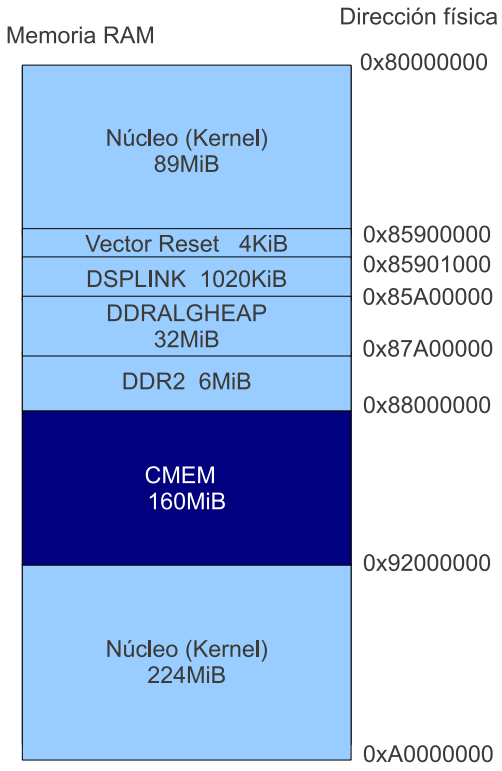

Figura 3.24: Mapeo de memoria modificado del sistema BeagleBoard-xM usando el SDK

- El procesador ARM ve CMEM como direcciones virtuales.
- Los búferes de XDM hacen la traducción de dirección virtual a física.

El puntero de la estructura es mapeado a una dirección física por medio del búfer XDM. Los miembros de la estructura mantienen su dirección virtual, lo que impide que el procesador de DSP pueda utilizar los datos de los miembros de la estructura, debido a que no reconoce las direcciones de memoria.

Mediante la función *CMEM_getPhys()* se traducen direcciones virtuales a físicas. Los búferes de la estructura *IHDR_Process* se reservan mediante la función *Memory_contigAlloc()*, la cual da un puntero a memoria virtual, este puntero a su vez es traducido por la función *CMEM_getPhys()* a una dirección física.

Las matrices de la estructura se reservan como un búfer, como se muestra en la figura 3.25.

Para una matriz bidimensional se reserva la memoria de los dos búferes (datos y control). La separación del búfer de datos se realiza asignando a cada posición del búfer de control la dirección física donde inicia cada segmento de datos. La dirección física del búfer de control es asignada al puntero de la matriz en la estructura. Esta forma permite que el procesador DSP reconozca las direcciones de los miembros de la estructura y pueda leer los datos.

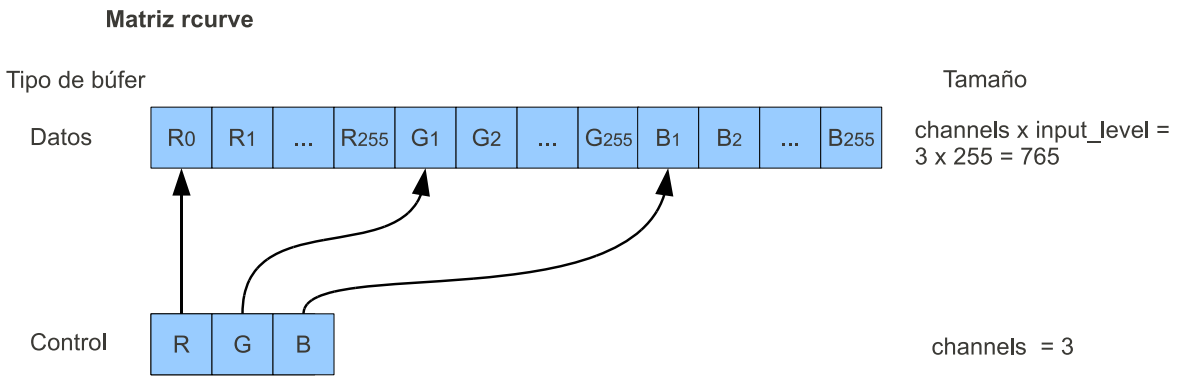

Figura 3.25: Reserva de memoria en CMEM de las matrices en la estructura *IHDR_Process*

3.6.2. Servidor

El servidor establece la comunicación entre la aplicación cliente *gsthdrimagedsp* y el codec *hdr*. Este es creado con un asistente de Texas Instruments que ofrece un paquete servidor genérico. Además posee la funcionalidad de enlazar bibliotecas con el código que es ejecutado en el procesador DSP (código en el codec).

El archivo *link.cmd* permite hacer el enlazado del código externo (bibliotecas) con el codec. La biblioteca TMS320C64x+ IQMath posee dos archivos que deben ser enlazados con el codec, estos son: IQmath_c64x+.lib y IQmath_RAM_c64x+.lib, para la utilización de las funciones y las tablas para el cálculo de las funciones matemáticas, trigonométricas y operaciones aritméticas.

3.6.3. Codec

En el codec se da la implementación de los bloques mostrados en la figura 3.15, es decir, se porta el código de la función *hdrimage_algorithm()* del complemento *gsthdrimagearm*, considerando:

- La estructura *IHDR_Process*.
- La inicialización de los miembros de la estructura.
- El manejo de archivos se hace desde la aplicación cliente.
- Los algoritmos son en punto flotante.
- Los búferes de entrada y salida son del tipo XDM.

En la función *Process* de la API de iUniversal que se encuentra en el codec se invoca la función *hdrimage_algorithm()*, la cual implementa los algoritmos en la generación de imágenes HDR con mapeo de tonos para el procesador de DSP.

La arquitectura del procesador de DSP es de punto fijo y posee un emulador de punto flotante. El emulador permite desarrollar aplicaciones de punto flotante en la arquitectura de punto fijo del DSP. Esto tiene una consecuencia: la reducción del rendimiento del procesador,

debido a la carga de operaciones de punto flotante a realizar en el emulador.

Migración de código a punto fijo

Dos formas de portar el código del codec a punto fijo con la biblioteca TMS320C64x+ IQmath son:

- *Local*: código dentro de cada función es convertido a punto fijo antes de realizar el procesamiento y los resultados son convertidos a punto flotante.
- *Global*: se reserva memoria para los búferes y matrices con punto fijo, se realiza el procesamiento de los algoritmos en punto fijo. La respuesta del sensor y los tiempo de exposción se almacenan en archivos de texto en punto fijo.

En el presente proyecto se portan ciertas funciones de manera local a punto fijo. Con la finalidad de analizar el rendimiento en la arquitectura de DSP entre las funciones se encuentran: *applyResponse()* y *responseLinear()*. La primera corresponde a la función que se encarga de generar la imagen HDR a partir de la respuesta de sensor y los canales de la imagen de entrada. La segunda función da la respuesta del sensor para la primera iteración en la calibración fotométrica.

En la figura 3.26 se muestra el proceso para convertir el procesamiento de una función en punto fijo. Algunas funciones de la biblioteca que se utilizan son:

- *_FtoIQ()*: convertir una variable tipo *float* a *_iq*.
- *_IQmpy()*: multiplicación de dos números del tipo *_iq*.
- *_IQdiv()*: división de dos números del tipo *_iq*.
- *_IQtoF()*: convertir un número del tipo *_iq* a *float*.

En la etapa de almacenamiento y visualización la imagen HDR con mapeo de tonos se escribe en un archivo y se convierte a un formato donde la imagen sea observable mediante algún visor de imágenes.

3.7. Etapa de almacenamiento y visualización

Este módulo está comprendido por elementos de GStreamer. Para almacenar la imagen se utiliza *multifilesink*, el cual almacena la imagen en un archivo en el formato *x-raw-yuv* o *x-raw-rgb*.

La conversión de formatos para poder observar la imagen se lleva a cabo por una tubería o línea de proceso de GStreamer. Convierte del formato *x-raw-yuv* a *jpeg*. El factor de calidad es el máximo permitido en el elemento (valor de 97).

gst-launch filesrc location= image_0.yuv ! 'video/x-raw-yuv, format=(fourcc)UYVY, width=(int)640, height=(int)480, framerate=(fraction)1/1' ! dmaienc_jpeg qValue=97 ! filesink location= img_0.jpg

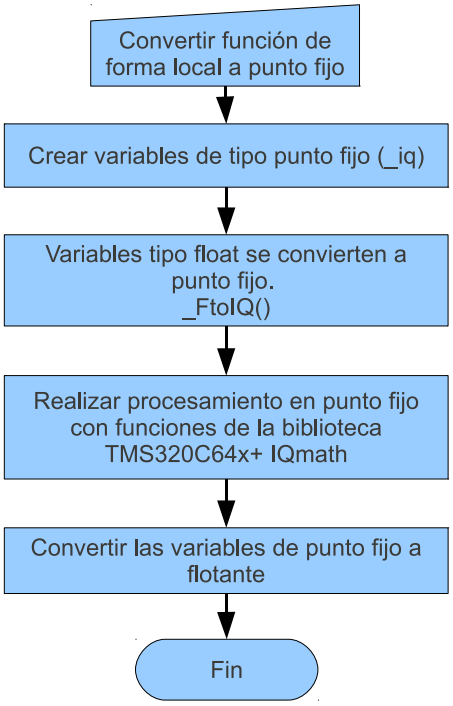

Figura 3.26: Proceso para convertir una función de forma local a punto fijo

Capítulo 4

Resultados y Análisis

En el presente capítulo se muestran los resultados del sistema de captura, el complemento *gsthdrimagearm* y *gsthdrimagedsp*. Los tiempos de ejecución de las funciones que constituyen los complementos y la comparación entre estos y el paquete de software *pfstools*. También se muestran histogramas de las imágenes con diferentes exposiciones y la imagen HDR con mapeo de tonos.

4.1. Sistema de captura

El módulo de captura permite obtener imágenes a diferentes exposiciones de una escena estática. Para el caso de generación de imágenes HDR con mapeo de tonos se capturan tres imágenes, una imagen a exposición normal, una imagen sub-expuesta a -2EV y una imagen sobre-expuesta a +2EV, como se muestra en las figura 4.1.

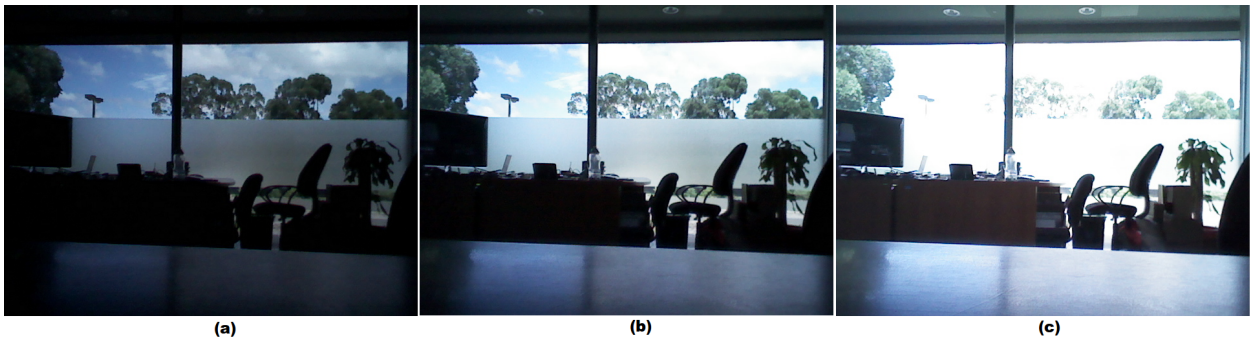

Figura 4.1: Escena estática obtenida por el sistema de captura. a). Imagen sub-expuesta. b). Imagen a exposición normal. c). Imagen sobre-expuesta

La captura de la imagen a exposición normal se da bajo los requisitos de habilitación de AE (Auto Exposición) y AWB (Auto Balance de Blancos), esto con la finalidad de que el sensor se ajuste al medio en que se encuentra. El sensor se ajusta a la escena en el aspecto de zonas oscuras y saturación presentes. Esto permite tener un valor por referencia para determinar la cantidad a sub y sobre exponer en la captura de las siguientes imágenes.

Antes de capturar la imagen a exposición normal se leen los registros *fine_integration_time* y *coarse_integration_time* del sensor MT9V113 para determinar a partir de la relación (3.1) el tiempo de exposición de la imagen. Para el caso de la imagen a exposición normal es de $1,409279$ ms.

La imagen sub-expuesta se captura a -2EV, lo que significa reducir los valores obtenidos en los registros *FINE* y *COARSE* de la imagen a exposición normal (imagen (b) en la figura 4.1) entre cuatro, reduciendo la cantidad de intensidad de luz incidente en el sensor en una proporción de cuatro.

Despreciando el búfer de configuración, se obtiene la imagen sub-expuesta que se muestra en (a) de la figura 4.1, con tiempo de exposición de $0,352320$ ms.

Para el caso de la imagen sobre-expuesta se contempla el búfer de configuración del sensor, causa de la escritura de los registros *FINE* y *COARSE* en un factor de cuatro veces los valores de la imagen a exposición normal, dando un aumento de $+2$EV (cuatro veces la cantidad de intensidad de luz incidente en el sensor). El tiempo de exposición de la imagen (c) de la figura 4.1 es $4,580338$ ms.

4.2. HDR y mapeo de tonos

La implementación de los algoritmos de HDR y mapeo de tonos en los procesadores ARM y DSP generan diferencias en los tiempos de ejecución debido a la arquitectura de cada procesador. Los valores medios y varianzas de tiempos de ejecución se obtienen con muestras de 30 ejecuciones o iteraciones del proceso de generación de imágenes HDR con mapeo de tonos.

En las siguientes subsecciones se presentan los resultados y análisis del sistema implementado en un computador de escritorio (PC, *Personal Computer*). Se utiliza el paquete de software pfstools desarrollado por el Instituto Max Planck [35]), en el procesador ARM y el procesador DSP.

4.2.1. Paquete de software *pfstools* e implementación en el procesador ARM con accesos por punteros

Los tiempos de ejecución con el software *pfstools* en la PC y el complemento *gsthdrimagearm* en el procesador ARM se obtienen a partir de la secuencia de imágenes que se muestran en la figura 4.2. La resolución de las imágenes es de 1024×768 píxeles.

Resultados con el software *pfstools*

Mediante el software *pfstools* se obtienen tiempos de ejecución en la calibración fotométrica del sensor de captura y la generación de la imagen HDR con mapeo de tonos.

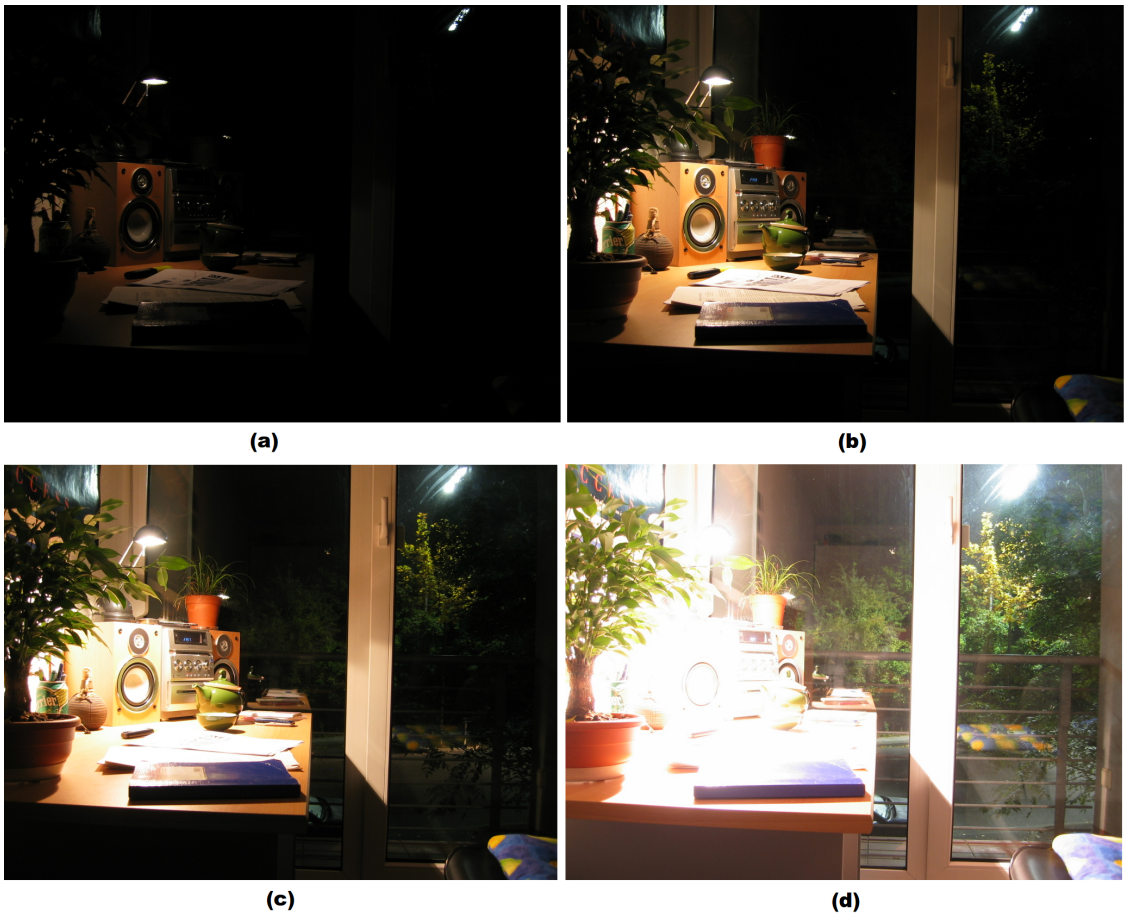

Figura 4.2: Escena estática del software *pfstools* con tiempo de exposición de (a) 0, 00897884 ms (b) 0, 15845 ms (c) 0, 528167 ms (d) 5, 28167 ms [35]

Las líneas de proceso para la calibración fotométrica y la generación de HDRI con mapeo de tonos son

- *pfsinhdrgen sample.hdrgen — pfshdrcalibrate -v -s camera.response*
- *pfsinhdrgen sample.hdrgen — pfshdrcalibrate -v -f camera.response — pfstmo_drago03 -b 0.8 — pfsgamma -g 1.3 — pfsout imghdr.png*

Los tiempos de ejecución en la calibración fotométrica y generación de la imagen HDR con mapeo de tonos que se muestra en la figura 4.3 se aprecia en la tabla 4.1.

Resultados del complemento *gsthdrimagearm*

En la figura 4.4 se muestra la imagen HDR con mapeo de tonos obtenida en el procesador ARM. Los parámetros de entrada al sistema son la secuencia de imágenes mostrada en la figura 4.2, el factor gamma de 1, 3 y el parámetro de bias de 0, 8.

La calibración fotométrica del sensor de captura esta conformada por las funciones:

- *initialization*
- *getChannelYUV*

Figura 4.3: Imagen HDR generada a partir de las imágenes de la figura 4.2, el software pfstools y un mapeo de tonos global

Figura 4.4: Imagen HDR obtenida a partir de las imágenes de la figura 4.2 en el procesador ARM con un mapeo de tonos global

- *YUV2RGB_conversion*
- *weights*
- *responseLinear*
- *getResponse*

y la generación de la imagen HDR con mapeo de tonos por:

- *applyResponse*, genera la imagen HDR.
- *toneMapOperator*
- *RGB2YUV_conversion*
- *setChannelYUV*

Tabla 4.1: Tiempos de ejecución con el paquete de software pfstools en la PC, arquitectura i686 y el complemento gsthdrimagearm en el procesador ARM de la BeagleBoard-xM

Tipo	Parámetro	i686 (s)	ARM cortex A8 (s)	Diferencia (s)	Porcentaje
Calibración	Promedio	59,1736	31,4863	27,6873	46,79
	Desv. estándar	1,9827	0,010724		
Generación	Promedio	2,8113	38,2981	-35,4868	1262,29
	Desv. estándar	0,119583	0,103557		
Process	Promedio	61,9849	69,7844	-7,7995	12,58
(calibrat=1)	Desv. estándar	2,002533	0,230434		

La función *process* comprende la calibración y generación cuando la propiedad *calibrat* del complemento *gsthdrimagearm* es uno, en el caso contrario, la función *process* comprende la generación de la imagen HDR a partir de la respuesta del sensor que se lee de un archivo de texto denominado *respCurve.txt*.

Los tiempos de ejecución de la calibración fotométrica y generación de la imagen se aprecia en la tabla 4.1.

La imagen generada por el software *pfstools* mostrada en la figura 4.3 presenta saturación del canal rojo con respecto a la imagen generada por el complemento *gsthdrimagearm*. Esta diferencia se da por la implementación del algoritmo de mundo gris y ajustes en la normalización de los píxeles de la imagen HDR en el complemento *gsthdrimagearm*. Esto produce un mayor tiempo de procesamiento en la generación de la imagen HDR con mapeo de tonos como se muestra en la tabla 4.1.

El tiempo de ejecución en la operación de generación de HDRI con mapeo de tonos en el complemento *gsthdrimagearm* es de $38,2981$ s. La misma operación con el software *pfstools* tiene una duración de $2,8113$ s. La diferencia de $35,4868$ s se debe a la implementación del algoritmo de mundo gris, conversión del espacio de color RGB a sRGB, sRGB a XYZ y viceversa, en el complemento *gsthdrimagearm*. Otro factor es la arquitectura en que se ejecuta y el lenguaje de programación en que se desarrolló la aplicación.

- *pfstools*: arquitectura AMD Athlon(tm) 64 X2 Dual Core Processor 6000+ (3GHz) y lenguaje de programación C++.
- *gsthdrimagearm*: arquitectura ARM Cortex-A8 core (1GHz) y lenguaje de programación C.

La operación calibración fotométrica mediante el complemento *gsthdrimagearm* es un $46,79\,\%$ más rápida que el software *pfstools* (ver tabla 4.1), esto se debe a optimizaciones en el código del complemento *gsthdrimagearm* y acceder los búferes por medio de punteros.

El tiempo total del proceso mediante el complemento *gsthdrimagearm* es $12,58\,\%$ mayor que el software *pfstools* (tabla 4.1). La implementación en el procesador ARM está limitada en recursos computacionales (sistema embebido) a diferencia del software *pfstools* que se ejecuta

en una PC.

Las figuras 4.3 y 4.4 presentan problemas con el color blanco (zonas de las luces) debido a que el máximo relativo de cada canal no es igual al máximo absoluto de los canales de la imagen. La teoría Retinex soluciona este problema, pero se debe considerar que esta teoría es incompatible con el algoritmo de mundo gris, por lo que al seleccionar el algoritmo de la teoría de Retinex mediante la propiedad *normaliza* del complemento *gsthdrimagearm* permite atenuar el defecto en las zonas blancas pero a su vez produce la saturación del rojo que se elimina mediante el algoritmo de mundo gris.

4.2.2. Implementación de los algoritmos en coma flotante en procesador ARM y DSP

Las imágenes de entrada del sistema para la evaluación de los algoritmos en coma flotante en los procesadores ARM y DSP son las que se muestran en la figura 4.1. La figura 4.5 muestra la imagen HDR con mapeo de tonos obtenida a partir de la secuencia de imágenes de la figura 4.1.

Figura 4.5: Imagen HDR con mapeo de tonos

Los tiempos de ejecución de las funciones que comprenden la función *process* obtenidos en el procesador ARM y en el procesador de DSP, se muestran en la tabla 4.2.

La función *toneMapOperator* esta comprendida por las funciones:

- *RGB2XYZ_conversor*
- *grayWorldAlgorithm*
- *tmoDrago03*
- *gammaCorrection*
- *XYZ2RGB_conversion*

Tabla 4.2: Tiempos de ejecución en el procesador ARM y DSP de las funciones del complemento

Función	Parámetro	DSP (s)	ARM (s)	Diferencia (s)	Porcentaje
getChannelYUV	Promedio	0,111501	0,042515	0,068986	61,87
YUV2RGB_conversion	Promedio	1,375262	0,160410	1,214852	88,34
	Desv. estándar		0,022293		
weights	Promedio	0,003029	0,001115	0,001914	63,19
responseLinear	Promedio	0,000773	0,000171	0,000602	77,88
getResponse	Promedio	42,125230	5,514449	36,610781	86,91
	Desv. estándar		0,005568		
applyResponse	Promedio	4,425193	1,293702	3,131491	70,77
toneMapOperator	Promedio	33,449203	14,086347	19,362856	57,89
	Desv. estándar	0,024597	0,002236		
RGB2YUV_conversion	Promedio	0,579566	0,204453	0,375113	64,72
process	Promedio	85,956699	21,542142	64,414557	74,94
	Desv. estándar	0,411975	0,069936		

Tabla 4.3: Tiempos de ejecución en el procesador ARM y DSP del mapeo de tonos

Función	Parámetro	DSP (s)	ARM (s)	Diferencia (s)	Porcentaje
tmoDrago03	Promedio	9,523678	3,424935	6,098743	64,04
gammaCorrection	Promedio	6,315024	3,370557	2,944467	46,63
XYZ2RGB_conversion	Promedio	7,083009	3,368194	3,714815	52,45
toneMapOperator	Promedio	33,449203	14,086347	19,362856	57,89
	Desv. estándar	0,411975	0,069936		

En la tabla 4.3 se muestran los tiempos de ejecución de las funciones que realizan el mapeo de tonos de la imagen HDR.

La imagen HDR con mapeo de tonos presenta problemas en los bordes (figura 4.5), esto debido al movimiento de la escena en las imágenes de entrada del sistema, estos leves movimientos en la captura de imágenes producen bordes con defectos. Estos defectos son conocidos como artefactos fantasma (Ghost Artefacts).

El tiempo de ejecución total (función *process*) del complemento *gsthdrimagedsp* es $74,94\,\%$ mayor que el complemento *gsthdrimagearm* (tabla 4.2). La causa de este resultado es la implementación de una aplicación de coma flotante en una arquitectura de coma fija (procesador DSP), donde las operaciones son llevadas acabo mediante un emulador produciendo un mayor tiempo de ejecución.

Los tiempos de ejecución de las funciones comprendidas en la calibración fotométrica y generación de la imagen HDR con mapeo de tonos en el complemento *gsthdrimagedsp* son mayores que el complemento *gsthdrimagearm* (tablas 4.2 y 4.3). El intervalo en que los

Tabla 4.4: Tiempos de ejecución en el procesador DSP para las funciones *applyResponse* y *responseLinear* en coma flotante y coma fija

Función	Parámetro	DSP-flotante (s)	DSP-fija (s)	Diferencia (s)	Porcentaje
applyResponse	Promedio	4,425193	0,991735	3,433458	77,59
ResponseLinear	Promedio	0,000773	0,000035	0,000738	95,47
process	Promedio	85,956699	82,281642	3,675057	4,28
	Desv. estándar	0,411974	0,549313		

tiempo de ejecución son mayores es entre 45 % y 90 %.

4.2.3. Implementación de los algoritmos en coma flotante y coma fija en el procesador de DSP

Con la finalidad de observar la optimización que produce el portar código de la aplicación de generación de imágenes HDR con mapeo de tonos a coma fija y ser ejecutado en el procesador DSP, se migran las funciones *applyResponse* (genera la imagen HDR) y *responseLinear* mediante la biblioteca IQMath. Los tiempos de ejecución se muestran en la tabla 4.4.

En la tabla 4.4 se muestra que el tiempo de la función *applyResponse* en coma fija se reduce en $77,59\,\%$ con respecto a la implementación en coma flotante y la función *responseLinear* en $95,47\,\%$. La reducción de estas funciones se refleja en la reducción de $4,28\,\%$ del tiempo de ejecución total de la aplicación (función *process*).

La reducción del tiempo de ejecución se debe a la optimización del código mediante la biblioteca IQMath y la arquitectura de coma fija del procesador DSP (se evita el uso del emulador de coma flotante).

4.2.4. Comparación de los algoritmos en coma flotante en el procesador ARM y coma fija en el procesador de DSP

En la tabla 4.5 se muestran los tiempos de ejecución de las funciones *applyResponse* y *responseLinear* en los complementos *gsthdrimagearm* y *gsthdrimagedsp*, considerando los tipos de arquitectura: procesador ARM con coma flotante y procesador DSP con coma fija.

La implementación en coma fija del procesador DSP muestra una disminución en el tiempo de ejecución con respecto a la de coma flotante en el procesador ARM. El tiempo de ejecución de la función *applyResponse* se reduce en $23,34\,\%$ y la función *responseLinear* en $38,60\,\%$ (tabla 4.5).

El tiempo de ejecución total de la aplicación (función *process*) con coma flotante en el procesador ARM es $21,542142$ s y en el procesador DSP con optimización en punto fijo de las funciones *applyResponse* y *responseLinear* es $82,281642$ s.

Tabla 4.5: Tiempos de ejecución en el procesador ARM en coma flotante y en el procesador de DSP en coma fija para las funciones *applyResponse* y *responseLinear*

Función	Parámetro	ARM-flotante (s)	DSP-fija (s)	Diferencia (s)	Porcentaje
applyResponse	Promedio	1,293702	0,991735	0,301967	23,34
ResponseLinear	Promedio	0,000171	0,000105	0,000066	38,60
process	Promedio	21,542142	82,281642	-60,7395	281,96
	Desv. estándar	0,069937	0,549313		

La función *process* comprendida por funciones no optimizadas para la arquitectura (funciones de coma flotante que utilizan el emulador produciendo un aumento en el tiempo de ejecución) y funciones optimizadas en coma fija (*applyResponse* y *responseLinear*) posee un aumento de $60,7395$ s con respecto al tiempo de ejecución en el procesador ARM.

La optimización en coma fija se realiza en una función del proceso de calibración fotométrica (*responseLinear*) y en la función *applyResponse* del proceso de generación de HDRI. La optimización en el tiempo de ejecución de dichas se debe a que la arquitectura del procesador DSP está optimizada para aplicaciones multimedia. Además de la optimización por coma fija se pueden utilizar técnicas de optimización como los *intrinsics*.

4.2.5. Análisis de imágenes HDR con mapeo de tonos por medio de histogramas

En las siguiente subsección se analiza la relación entre histograma de frecuencia e imágenes con diferentes exposiciones y HDR con mapeo de tonos.

Imagen HDR con mapeo de tonos obtenida en el procesador DSP imágenes de entrada del sistema de captura

En la figura 4.6 se muestra una imagen a exposición normal con el respectivo histograma de frecuencias.

La distribución de frecuencias de la imagen a exposición normal presenta dos picos pronunciados que corresponden a zonas oscuras (aproximadamente 7000 píxeles) y zonas con saturación (aproximadamente 26000 píxeles).

En la figura 4.7 se muestra una imagen sub-expuesta a -2EV con respecto a la imagen a exposición normal (figura 4.6) y el respectivo histograma de frecuencias.

Al sub-exponer en -2 EV se reduce la cantidad de luz que incide en el sensor en un factor de 4. Esto permite reducir las zonas de saturación en la imagen a consecuencia de un aumento de las zonas oscuras. La imagen sub-expuesta presenta una distribución de frecuencias de píxeles cercanas a cero. La cantidad de píxeles en la región oscura es aproximadamente 17000.

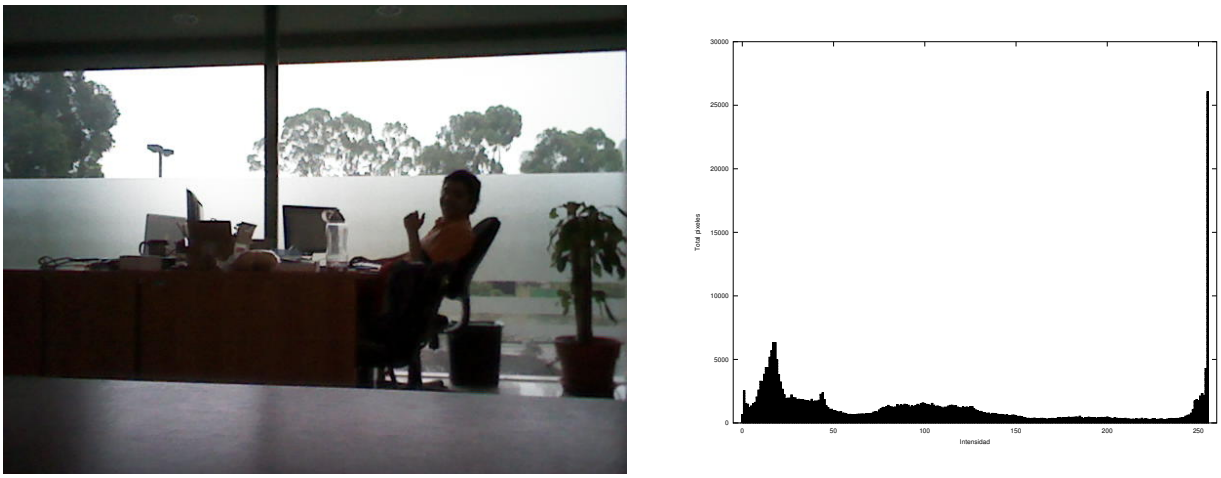

Figura 4.6: Imagen a exposición normal con respectivo histograma de frecuencias

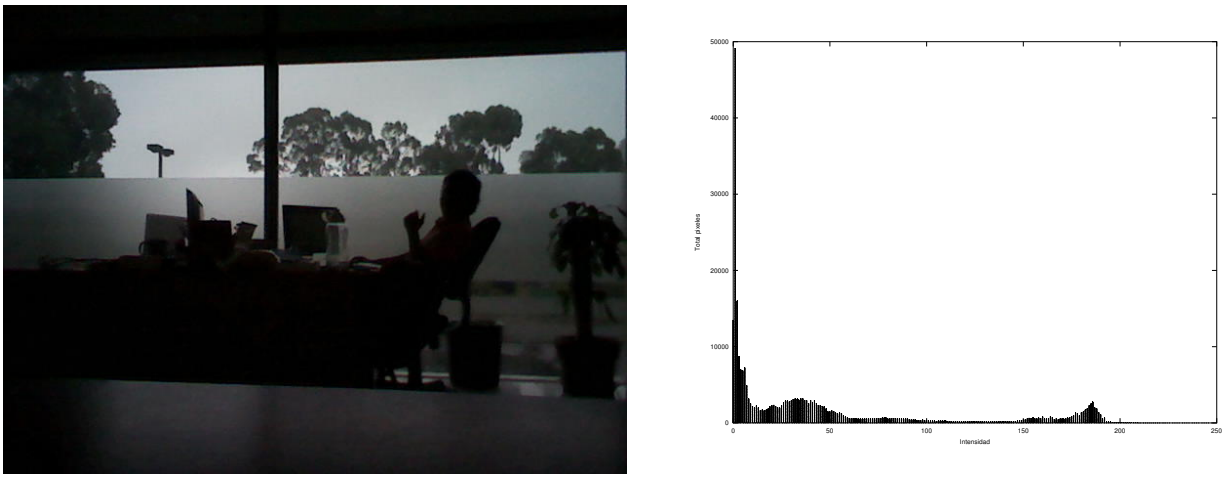

Figura 4.7: Imagen sub-expuesta con respectivo histograma de frecuencias

En la figura 4.8 se muestra una imagen sobre-expuesta a +2EV con respecto a la imagen a exposición normal (figura 4.6) y el respectivo histograma de frecuencias.

AL sobre-exponer en +2 EV se aumenta la cantidad de luz que incide en el sensor en un factor de 4. Esto permite aumentar las zonas de saturación en la imagen a consecuencia de una reducción de las zonas oscuras. La imagen sobre-expuesta presenta una distribución de frecuencias de píxeles cercanas a 255. La cantidad de píxeles en la región de saturación es aproximadamente 66000.

La figura 4.9 se muestra la imagen HDR con mapeo de tonos obtenida a partir de la secuencia de imágenes de la figura 4.2 y el respectivo histograma de frecuencias.

La imagen HDR con mapeo de tonos presenta un histograma de frecuencias uniformemente distribuido en el rango de valores entre 0 y 255. Los extremos (zonas oscuras y con saturación) tienden a cero evitando la pérdida de información en las regiones oscuras y de saturación.

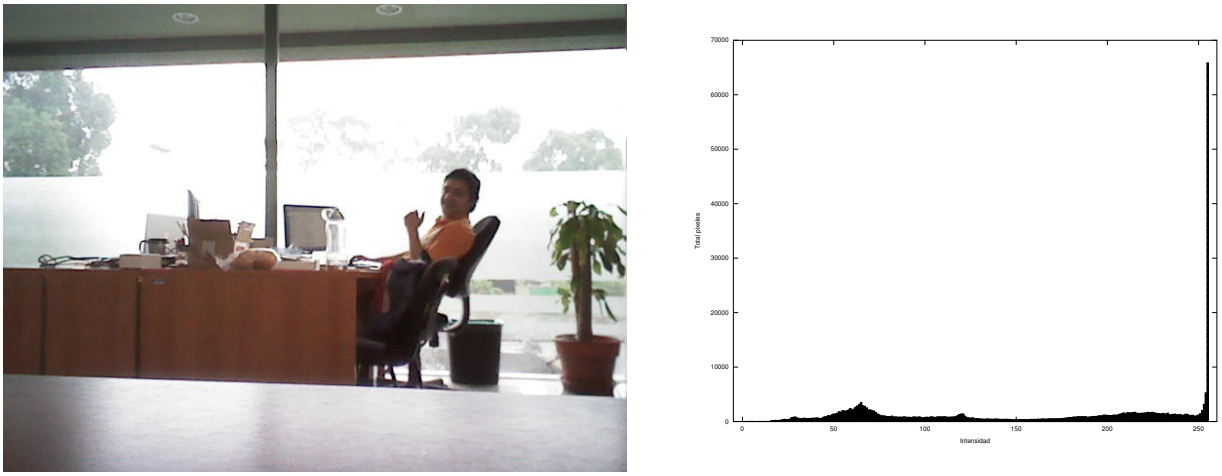

Figura 4.8: Imagen sobre-expuesta con respectivo histograma de frecuencias

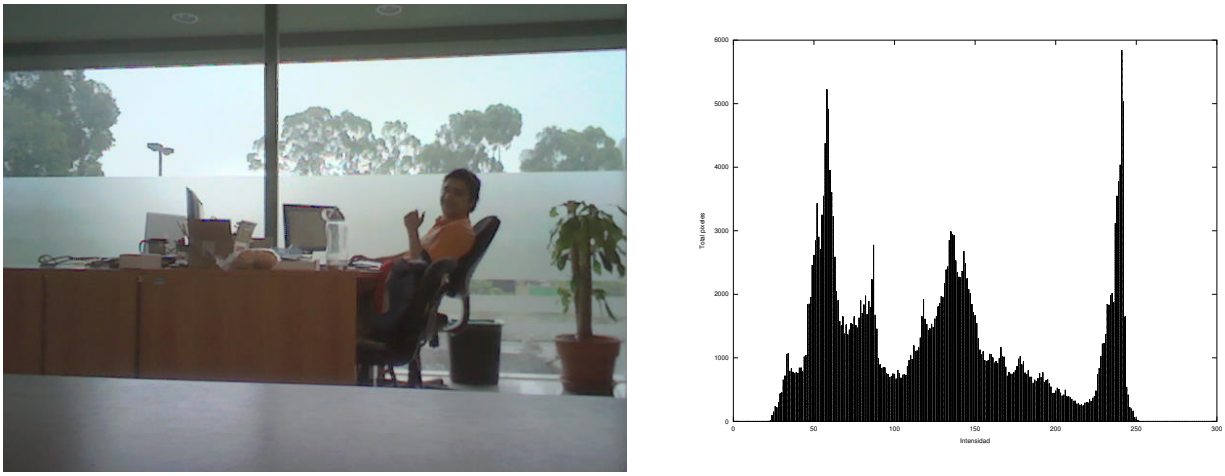

Figura 4.9: Imagen HDR con mapeo de tonos obtenida a partir de las imágenes de la figura 4.2 y respectivo histograma

Capítulo 5

Conclusiones y Recomendaciones

5.1. Conclusiones

En el presente trabajo se muestra el proceso de diseño e implementación de la aplicación de generacion de imágenes HDR con mapeo de tonos en la plataforma BeagleBoard-xM. Para ello se utilizó el sensor de captura MT9V113 y la interfaz de programación GStreamer.

Se ha diseñado un módulo de captura el cual permite obtener imágenes a diferentes exposiciones considerando la imagen a exposición normal como referencia. La imagen a exposición normal contempla el balance automático de blancos y el ajuste automático de exposición. La señal de activación que inicia el proceso de captura de las imágenes con diferentes exposiciones se implementó mediante el evento del botón de usuario en la tarjeta BeagleBoard-xM.

Se muestran también los módulos de generación de imágenes de alto rango dinámico y mapeo de tonos, con la implementación en el procesador ARM en coma flotante y en el procesador DSP en coma flotante con optimización de dos funciones por coma fija.

En los módulos de generación de HDR y mapeo de tonos en el procesador DSP se realiza el paso de parámetros de tipo flotante mediante los búferes de XDM y la recuperación de los mismos en el codec que se ejecuta en el procesador DSP y en la aplicación del procesador ARM, principalmente en el proceso de lectura/escritura de la respuesta del sensor.

El análisis de requisitos de memoria compartida CMEM y la modificación de esta permite la implementación del complemento *hdrimagedsp*.

En la implementación del complemento *hdrimagedsp* se muestra el proceso de pasar una estructura con miembros del tipo punteros como parámetro a través de un búfer de XDM, haciendo uso de la función *Memory_contigAlloc()* la cual permite reservar memoria en el segmento de memoria contigua y *CMEM_getPhys()* que permite obtener la dirección física mediante la traducción de una dirección virtual, considerando que el procesador ARM ve el segmento de memoria compartida como direcciones virtuales y el procesador DSP como direcciones físicas.

En el proceso de calibración fotométrica del sensor de captura el tiempo de ejecución en

la implementación del procesador ARM se reduce en $49,79\,\%$ con respecto al paquete de software pfstools para cuatro imágenes con diferentes exposiciones y resolución de 1024×768 píxeles.

La implementación de los algoritmos de mundo gris y la teoría Retinex permite eliminar la saturación a nivel global de la imagen HDR con mapeo de tonos y problemas en regiones blancas, respectivamente.

La implementación de la función *applyResponse* en coma fija para el procesador DSP utiliza tan solo un $23,34\,\%$ del tiempo de ejecución de la implementación en coma flotante del procesador ARM. Además, el tiempo de ejecución de la función *responseLinear* en coma fija del procesador DSP se reduce en $38,60\,\%$ con respecto a la implementación en coma flotante del procesador ARM. Esto para el caso de tres imágenes con resolución 640×480 píxeles como entrada al módulo de generación de HDRI.

La implementación propuesta en este trabajo de los algoritmos de HDR y mapeo de tonos en coma flotante en el procesador DSP posee un mayor tiempo de ejecución ($74,94\,\%$) que la implementación en el procesador ARM, para el caso de imágenes con resolución 640×480 píxeles.

La generación de imágenes HDR con mapeo de tonos a partir de tres imágenes con diferentes exposiciones (normal, -2EV y $+2$EV) permite distribuir uniformemente los píxeles en el rango entre 0 y 255 de la imagen HDR, con tendencia a cero en cuanto a cantidad de píxeles en los extremos (zonas oscuras y saturadas).

5.2. Recomendaciones

La arquitectura del procesador DSP es de coma fija por lo que portar el código de coma flotante a coma fija del complemento *gsthdrimagedsp* permitiría optimizar el tiempo de ejecución de la aplicación de generación de imágenes con mapeo de tonos. Adicional a esta optimización se pueden realizar optimizaciones con técnicas como: redución de la sobrecarga de ciclo, *restrict qualifiers*, utilización de estructuras por referencia, utilización del pragma *MUST_ITERATE()* y *_nassert()*, optimización en declaraciones *if* y el uso de intrinsics.

La generación de imágenes HDR con mapeo de tonos se restringe a escenas estáticas para evitar los *Ghost Artefacts*. En los sistemas de captura en los que el usuario interviene físicamente con la activación de la cámara se producen leves movimientos en la escena generando píxeles defectuosos en los bordes de la imagen HDR con mapeo de tonos; una alternativa para evitar el problema de movimiento en el módulo de captura es la implementación de un módulo de activación remota.

Bibliografía

[1] J. Arines. *Tesis de Doctorado: Imagen de alta resolución del fondo de ojo por deconvolución tras compensación parcial*. Universidade de Santiago de Compostela, 2006.

[2] H. Verkuil M. Rubli B. Dirks, M. H. Schimek. Video for linux two api specification. revision 0.24 [online]. 2008 [visitado el 31 de enero de 2012]. URL http://v4l2spec.bytesex.org/spec-single/v4l2.htmlh.

[3] D. M. Ritchie B. W. Kernighan. *The C Programming Language*. Prentice Hall, 2nd edition, 1991.

[4] Beagleboard.org. Beagleboard-xm product details [online]. 2011 [visitado el 31 de enero de 2012]. URL http://beagleboard.org/hardware-xM.

[5] M. Beuchot. *Introducción a las ciencias de la computación con JAVA*. UNAM. Facultad de Ciencias, 1era edition, 2007.

[6] P. Debevec and J. Malik. Recovering high dynamic range radiance maps from photographs. In *Proceedings of SIGGRAPH 97*, August 1997.

[7] S. Pattanaik E. Reinhard, G. Ward and P. Debevec. *High Dynamic Range Imaging: Acquisition, display and image-based lighting*. Morgan Kaufmann Publisher, 2005.

[8] T. Annen F. Drago, K. Myszkowski and N. Chiba. Adaptive logarithmic mapping for displaying high contrast scenes. In *Computer Graphics Forum, proceedings of Eurographics 2003 22(3)*, page 419–426, 2003.

[9] R. C. González and R. E. Woods. *Digital Image Procesing*. Prentice Hall, 2002.

[10] M. D. Grossberg and S. K. Nayar. What can be known about the radiometric response function from images? In *Proceedings of European Conference on Computer Vision (ECCV), Copenhagen*, May 2002.

[11] J. Palmieri H. Pennington, D Wheeler and C. Walters. D-bus tutorial. tutorial en línea [online, visitado el 31 de enero de 2012]. URL http://dbus.freedesktop.org/doc/dbus-tutorial.html.

[12] Aptina Imaging. Image sensors [online]. 2011 [visitado el 31 de enero de 2012]. URL http://www.aptina.com/products/image_sensors/.

[13] Cambridge in colour. Dynamic range in digital fotography [online, visitado el 31 de enero de 2012]. URL http://www.cambridgeincolour.com/tutorials/dynamic-range.htm.

[14] Texas Instruments Incorporated. *Codec Engine Algorithm Creator User'sGuide.LiteratureNumber : SPRUED6C.TexasInstruments*, 2007.

[15] Texas Instruments Incorporated. *xDAIS-DM (Digital Media) User Guide. Literature Number: SPRUEC8B.* Texas Instruments, 2007.

[16] Texas Instruments Incorporated. Rtsc [online]. 2008 [visitado el 31 de enero de 2012]. URL http://processors.wiki.ti.com/index.php/Category:RTSC.

[17] Texas Instruments Incorporated. Tms320c64x+ iqmath library [online]. 2008 [visitado el 31 de enero de 2012]. URL http://www.ti.com/litv/pdf/sprugg9.

[18] Texas Instruments Incorporated. Cmem overview [online]. 2010 [visitado el 31 de enero de 2012]. URL http://processors.wiki.ti.com/index.php/CMEM_Overview.

[19] Texas Instruments Incorporated. Dsplink [online]. 2010 [visitado el 31 de enero de 2012]. URL http://processors.wiki.ti.com/index.php/Category:DSPLink.

[20] Texas Instruments Incorporated. Getting started with iuniversal [online]. 2010 [visitado el 31 de enero de 2012]. URL http://processors.wiki.ti.com/index.php/Getting_started_with_IUNIVERSAL.

[21] Y. Lam. *Combining Gray World and Retinex Theory for Automatic White Balance in Digital Photography.* Universidad de Hong Kong, 2005.

[22] S. Borman M. A. Robertson and R. L. Stevenson. Estimation-theoretic approach to dynamic range improvement using multiple exposures. In *Journal of Electronic Imaging, vol. 12, no. 2*, April 2003.

[23] B. Pérez M. Gruner and P. Alvarado. Diseño de algoritmos pds para arquitecturas híbridas de texas instruments: una comparación de estrategias de implementación. In *Proceedings of Embedded Technologies Conference 2011, Costa Rica*, page 31–37, 2011.

[24] B. C. Madden. Extended intensity range imaging. In *Technical report, GRASP Laboratory, University of Pennsylvania*, 1993.

[25] F. Maldonado. *Tesis de Doctorado: Optimización del proceso de teñido de telas.* Universidad Nacional de Quilmes, Argentina, 2005.

[26] M. Mancuso and S. Battiato. An introduction to the digital still camera technology. In *ST Journal of System Research - Special Issue on Image Processing for Digital Still Camera*, pages 200–1, 2001.

[27] T. Mitsunaga and S. K. Nayar. Radiometric self calibration. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, Fort Collins*, June, 1999.

[28] K. Moriwaki. Adaptive exposure image input system for obtaining high-quality color information. In *Systems and Computers in Japan*, July 1994.

[29] GNOME org. Gnome developer center [online]. 2011 [visitado el 31 de enero de 2012]. URL http://developer.gnome.org/.

[30] F. G. Ortiz. *Tesis de Doctorado: Procesamiento morfológico de imágenes en color. Aplicación a la reconstrucción geodésica*. Biblioteca virtual Miguel de Cervantes. Universidad de Alicante, 2002.

[31] K. N Plataniotis and A. N Venetsanopoulos. Color image processing and applications. In *2000-SpringerMonograph*, 2000.

[32] Charles Poynton. Color FAQ [online, visitado el 31 de enero de 2012]. URL http://www.poynton.com/notes/colour_and_gamma/ColorFAQ.html.

[33] G. Krawczyk R. Mantiuk and H. P. Seidel. High dynamic range imaging pipeline: Perception-motivated representation of visual content. In *Proceedings of SPIE (6492-212). Human Vision and Electronic Imaging XII*, 2007.

[34] D. Rodriguez. Hdr-rango dinámico extendido [online, visitado el 31 de enero de 2012]. URL http://web.me.com/danimolowny/Tecnicas_Fotograficas/HDR.html.

[35] M. Goesele V. Havran G. Krawczyk R. Mantiuk K. Myszkowski T. Aydın, M. Čadík and A. Yoshida. High dynamic range image and video processing [online]. 2011 [visitado el 31 de enero de 2012]. URL http://www.mpi-inf.mpg.de/resources/hdr/index.html.

[36] A. Wingo R. S. Bultje W. Taymans, S. Baker and S. Kost. Gstreamer application development manual (0.10.35.1) [online]. 2011 [visitado el 31 de enero de 2012]. URL http://gstreamer.freedesktop.org/data/doc/gstreamer/head/manual/html/index.html.

[37] C. Wright. Yuv colorspace [online]. 2004 [visitado el 31 de enero de 2012]. URL http://softpixel.com/~cwright/programming/colorspace/yuv/.

[38] I. ZStardust. Archivo:eyesection-es.svg [online, visitado el 31 de enero de 2012]. URL http://upload.wikimedia.org/wikipedia/commons/b/ba/Eyesection-es.svg.

[39] E. Zuñiga and J. Hidalgo. *Tesis de Licenciatura: Plataforma de software empotrado para la implementación de algoritmos de audio y video en el DSP de la arquitectura OMAP-L138*. Tecnológico de Costa Rica, 2010.

Apéndice A

Ejemplos de HDRI con mapeo de tonos

Figura A.1: Escena estática 1: Secuencia de imágenes con diferentes exposiciones e imagen HDR con mapeo de tonos

Figura A.2: Escena estática 2: Secuencia de imágenes con diferentes exposiciones e imagen HDR con mapeo de tonos