Componentes de visualización e interacción para implementación de arquitecturas computacionales abiertas

Harver Andrey Cordero Duarte

Trabajo de Grado para Optar el título de Ingeniero de Sistemas e Informática

Director

Carlos Jaime Barrios Hernández,

PhD. en Informática

Codirector

Jonnathan Ramos Chaux,

Msc en Ingeniería de Sistemas e Informática

Universidad Industrial de Santander

Facultad de Ingenierias Fisicomecánicas

Escuela de Ingeniería de Sistemas e Informática

Bucaramanga

2019

Dedicatoria

Quiero dedicar este logro a:

A mi madre Rosalba Duarte Patiño, con su apoyo incondicional, esfuerzo y confianza depositada en mí, fueron un pilar importante para culminar este logro.

A mis abuelos Roberto Duarte Quintero, que, con su apoyo económico, cariño, apoyo y como un guía que compartió sus experiencias de vida, aportaron a mi crecimiento personal y profesional, y Matilde Patiño de Duarte, como un guía más en mi crecimiento personal, aportaron a grandes rasgos ingredientes necesarios para el profesional que soy ahora.

A mis hermanos, que, con esos lazos de confianza y amistad incondicional, hicieron de mis años en la Universidad Industrial de Santander una grata época de vida.

También dedico este logro a una persona muy importante en mi vida, Angie Milena Sánchez Mendez, que, con su cariño, confianza y apoyo incondicional se convirtió en aquella inspiración que me incito a no rendirme y cumplir cada meta que me propuse en mis últimos semestres, recordándome siempre dar lo mejor de mí.

Y, por último, pero no menos importante, a mi mejor amigo Juan Camilo Pertuz Manosalva, que, con su gran amistad y experiencias inolvidables, hizo de mi carrera de pregrado un grato recuerdo de la que considero, la mejor etapa de mi vida hasta el día de hoy.

Agradecimientos

A la Universidad Industrial de Santander, más que una universidad ese segundo hogar que siempre llevaré en mis recuerdos, que me permitió desarrollar mis competencias en el área profesional y personal, junto a docentes de calidad y personas de relevancia importante en mi vida, además de brindar sus recursos y espacios académicos que hicieron posible el culminar con éxito mi carrera profesional.

A la escuela de Ingeniería de Sistemas e Informática, en especial a la secretaria Lady por su paciencia y gestión a lo largo del proyecto, al SC3UIS por brindarme el espacio y recursos necesarios que fueron la base para el desarrollo de este proyecto de grado.

Al profesor Carlos Jaime Barrios por haberme guiado en el desarrollo y culminación de este proyecto, por haber aceptado la dirección del mismo y quien con su paciencia, dedicación y tiempo invertido en correcciones, sugerencias y guía académica, hizo posible de este proyecto un éxito profesional, personal y académico.

Al profesor Jonnathan Ramos Chaux, por su valiosa ayuda, dirección y apoyo en el desarrollo de este documento y también por cada enseñanza y brindar sus conocimientos y experiencias en el desarrollo de un proyecto encaminado a ingeniería.

Contenido

Pág.

Lista de Figuras

Pág.

Lista de Tablas

Pág.,

Lista de Apéndices

Pág.

Resumen

TITULO: COMPONENTES DE VISUALIZACIÓN E INTERACCIÓN PARA IMPLEMENTACIÓN DE ARQUITECTURAS COMPUTACIONALES ABIERTAS[*]

AUTOR: HARVER ANDREY CORDERO DUARTE[**]

PALABRAS CLAVE: Kinect, Creative, Visualización, Interacción, OpenSource.

DESCRIPCIÓN:

Con la existencia de dispositivos de visualización e interacción desarrollados con políticas de uso privadas, se crean parámetros limitantes, que ejercen control en torno a la aplicación y el manejo que se le dé a los mismos, además de generar costo adicional debido a la necesidad de uso de software licenciado desarrollado para establecer conexión con las arquitecturas para las que fueron desarrolladas. Partiendo de esta idea, este proyecto se realiza con el fin de implementar dichos dispositivos en ambientes embebidos, diseñando una propuesta de una arquitectura computacional de acuerdo a requerimientos que garanticen la integración con arquitecturas que sigan una política abierta, alto rendimiento, portabilidad, eficiencia energética y bajo costo. Como primer paso se hace un análisis del funcionamiento y desempeño de estos dispositivos por medio de una conexión modular haciendo uso de librerías desarrolladas en código abierto con un sistema operativo basado en Linux, esto con la finalidad de identificar casos de uso de acuerdo a requerimientos y lineamientos de implementación para la explotación de la arquitectura. Para lograr los objetivos planteados en este proyecto se estipulan una serie de pruebas de rendimiento estructuradas con la finalidad de cumplir con cada lineamiento de implementación establecido, además de contar con una debida documentación que respalda con el debido cumplimiento del objetivo general y propone una serie de conclusiones que resultan del desarrollo y finalización de este proyecto OpenSource.

[*] Trabajo de Grado en la modalidad de investigación

[**] Facultad de Ingenieria Fisico-mecánicas. Escuela de Ingenieria de sistemas e Informática. Director PhD Carlos Jaime Barrios Hernández, Co-director Msc Jonnathan Ramos Chaux

Abstract

TITLE: COMPONENTS OF VISUALIZATION AND INTERACTION FOR THE IMPLEMENTATION OF OPEN COMPUTING RCHITECTURES [1]

AUTHOR: HARVER ANDREY CORDERO DUARTE[2]

KEYWORDS: Kinect, Creative, Visualización, Interacción, OpenSource.

DESCRIPTION:

With the existence of visualization and interaction devices developed with private use policies, limiting parameters are created, which exert control over the application and the management that is given to them, besides generating additional cost due to the need to use of licensed software developed to establish connection with the architectures for which they were developed. Starting from this idea, this project is carried out in order to implement these devices in embedded environments, designing a proposal for a computational architecture according to requirements that guarantee the integration with architectures that follow an open policy, high performance, portability, energy efficiency and low cost. As a first step, an analysis of the operation and performance of these devices is made through a modular connection using libraries developed in open source with an operating system based on Linux, this in order to identify use cases according to requirements and implementation guidelines for the exploitation of the architecture. To achieve the objectives set forth in this project, a series of structured performance tests are established in order to comply with each established implementation guideline, in addition to having a proper documentation that supports the proper fulfillment of the general objective and proposes a series of conclusions that result from the development and finalization of this OpenSource project.

[1] Undergraduate final Project. Research modality.

[2] Faculty of Physical-Mechanical Engineering, School of Engineering and Computer Systems. Advisor: PhD Carlos Jaime Barrios, Co-advisor Msc Jonnathan Ramos Chaux.

Introducción

Con el desarrollo de dispositivos de hardware desarrollados bajo políticas y condiciones de uso privado, automáticamente crea parámetros limitantes que ejercen un control en torno a la aplicación y manejo de se le dé a los mismos. Los desarrolladores de hardware privado buscan la forma de mantener sus productos a una dependencia total de software licenciado que es desarrollado por ellos mismos, logrando restringir las posibilidades de adaptar estos dispositivos como mecanismos que permitan satisfacer las necesidades propias de uso de cada usuario, además de aumentar costos debido a la actualización del uso de una licencia pagada.

Los proyectos OpenSource facilitan el aporte de una visión diferente en torno al uso de ya mencionados dispositivos, dando oportunidad de que los usuarios que los utilicen puedan adaptarlos con arquitecturas computacionales abierta, ya sea por medio de herramientas, aplicaciones o librerías desarrolladas en software libre, esto con la finalidad de satisfacer las necesidades de uso del propietario del hardware. Teniendo en cuenta lo anteriormente mencionado surge la idea de liberar los dispositivos de hardware privado, a un entorno más cómodo que facilite el uso y aplicación de os mismos con políticas abiertas, reduciendo cada vez más la brecha que limita a los dispositivos con el uso obligatorio de software que no sigue políticas abiertas, licenciado o pago.

El software y hardware libre, propuesto como una alternativa en el uso de los dispositivos de los que hemos hablado y más concretamente los dispositivos usados en el desarrollo de este proyecto como lo son Creative y Kinect para ser implementados con un enfoque diferente para el

que se diseñaron, son una herramienta útil que de alguna forma aportan ventajas significativas en lo que corresponde a reducir costos tratando de mantener un rendimiento similar o si es posible, mejor, que cuando se usan en entornos de políticas privativas.

Desarrollando un caso más particular de la idea anterior, se propone el diseño e implementación de una arquitectura computacional abierta que cumpla requerimientos de alta capacidad de computo, rendimiento, bajo costo, bajo consumo energético y portabilidad, surge la iniciativa de proponer el uso de un sistema embebido que cumpla con los lineamientos mencionados, como una solución alternativa al problema de liberar componentes de hardware dedicados a la visualización e interacción hacia un entorno que siga políticas abiertas, que permitan mayor libertad respecto al uso de los mismos. Como caso particular para el desarrollo de este proyecto se implementa el sensor Kinect y la cámara Creative a las necesidades que surgen de la aplicación de la visión por computadora en arquitecturas que siguen políticas abiertas.

Con el uso de la plataforma de desarrollo en ambientes embebidos Jetson TK1, se logra cubrir completamente las necesidades que surgieron a partir del planteamiento de la problemática que se está tratando, permitiendo el uso de los dispositivos Kinect y Creative en el campo de la visión por computador. El kit de desarrollo Jetson TK1 es una supercomputadora móvil que se usa para garantizar una implementación modular con aplicaciones de software libre que requieran visión por computadora, procesamiento de imágenes y captura de datos en tiempo real en un sistema operativo basado en Linux, lo que hace de este planteamiento una solución viable a la problemática en discusión.

Teniendo en cuenta todo lo anterior se aprecia que la temática de desarrollo del proyecto se enfoca en la implementación de hardware de visualización e interacción en ambientes embebidos que cumplan políticas abiertas, tanto en software como hardware, y que, de igual forma la conexión

entre estos módulos, se aplique a la visión por computador teniendo en cuenta lineamientos de implementación en torno a alto rendimiento en tareas de computo de imagen, portabilidad, eficiencia energética y bajo costo.

1. Objetivos

1.1 Objetivo General

Caracterizar los componentes de arquitecturas computacionales abiertas que permitan la visualización e interacción en ambientes embebidos.

1.2 Objetivos Específicos

- Analizar el funcionamiento y desempeño de dispositivos de visualización e interacción de acuerdo a requerimientos que garanticen la integración con arquitecturas que sigan una política abierta, alto rendimiento, portabilidad, eficiencia energética y bajo costo.
- Diseñar una arquitectura computacional bajo software y hardware abierto para caracterizar dispositivos de visualización e interacción sobre la misma, estableciendo lineamientos de implementación.
- Evaluar el desempeño, funcionalidad y rendimiento de la arquitectura propuesta.

- Identificar casos de uso de acuerdo a requerimientos y lineamientos de implementación para la explotación de la arquitectura

2. Marco Teórico y Metodológico.

2.1 Marco Teórico

En el desarrollo del proyecto se establece una conexión entre dispositivos de adquisición de imagen, visualización e interacción con una arquitectura computacional abierta en ambiente embebido desarrollados para computación de alto rendimiento, que además operan con OpenSource, con la finalidad de aprovechar las características funcionales de dichos dispositivos para aprovechar su funcionalidad en otro tipo de aplicaciones. Teniendo en cuenta lo anterior se establece una conexión en un kit de desarrollo NVIDIA Jetson TK1 (NVIDIA Corporation, 2014) para destinar el uso de los dispositivos Kinect(Einnews, 2010) en el campo de la visión por computadora (Alegre Enrique, 2006) con el uso de software libre, permitiendo que cualquier usuario de la arquitectura propuesta tenga mayor facilidad de uso en proyectos de ingeniería aplicados a la visión por computadora y pueda aportar mejor significativas, tanto de hardware como software, para identificar nuevos casos de uso en la implementación con arquitecturas computacionales abiertas basadas en arquitecturas diferentes a las ya establecidas.

Para una mejor comprensión en el desarrollo del proyecto es necesario definir los tres ejes temáticos que dimensionan el alcance del proyecto, y esto son: Sistemas Embebidos, OpenSource

y Visión por Computadora, teniendo en cuenta que esta última se usa para identificar aplicaciones de la arquitectura estudiada y no como un eje principal para el estudio de los módulos que la componen.

2.1.1 Sistemas Embebidos. Los sistemas embebidos son dispositivos electrónicos cuya finalidad es realizar tareas determinadas. Aportan un grado de agilidad y autonomía en los procesos que se deseen realizar, motivo por el cual muchos de estos sistemas han evolucionado hasta convertirse en SBC(Single Board Computer/Computadoras en una tarjeta) logrando prestaciones cercanas a equipos de cómputo de escritorio, las cuales poseen un costo moderado en comparación a otros sistemas usados en el campo de la visualización científica, la cual demanda gran capacidad de cómputo y el uso de GPU (Unidades de Procesamiento de Gráficos), razón por la que una tarjeta de desarrollo Nvidia Jetson TK1, hacen de este sistema embebido una herramienta útil para el desarrollo de la arquitectura usada para este proyecto; además de poder considerarse como un kit de desarrollo en hardware libre, debido a la libertad de modificación de sus módulos integrados y facilidad de adaptar otros módulos de hardware embebido para el diseño de clústeres para aplicaciones conjuntas, ya sea para tratamiento de datos o como unidades de procesamiento de gráficos de alto desempeño y con un bajo consumo energético; además de tener la característica de ser hardware configurable, lo que le permite la adaptación de software, que generalmente es libre, pero sin restringir la posibilidad de uso de software licenciado.

2.1.1.1 Kit de desarrollo Nvidia Jetson TK1. El kit de desarrollo Jetson TK1 es una plataforma embebida que trabaja sobre Linux y posee un System on a Chip (SOC) que integra módulos que unidos forman una computadora en un único circuito integrado o chip, razón por la cual implica

una mayor capacidad de memoria, menor coste de producción, velocidad de comunicación entre módulos y además funciona con bajo consumo energético. La tarjeta Jetson cuenta con un Tegra TK1 (CPU+GPU+In System Programmable (ISP)), además de tener integrado un sistema operativo Linux4Tegra pre-instalado, es decir un Ubuntu 14.04 con drivers pre- configurados) (Pastor, 2017).

Posee una CPU quad-core 2.3GHz ARM Cortex-A15 y la GPU Tegra K1 de Nvidia; la tarjeta Jetson TK1 incluye similitudes entorno a las características de una Raspberry Pi (Raspberry Pi Foundation, 2015), lo cual lo convierte en un ordenador mini que portable, con características de alto desempeño para su tamaño y el consumo energético, además de funciones orientadas a la transferencia de datos entre la placa base y dispositivos de almacenamiento, como discos duros, discos ópticos y otros dispositivos que demandan altas tensiones.

Entre sus características también posee mini-Periphetal Component Interconnect express (mini-PCIe), el cual es un bus de alta velocidad de comunicación para establecer conexión con periféricos directamente con la placa base, y un ventilador que permite un continuo funcionamiento aún con grandes cargas de trabajo.

Figura 1. Puertos kit de desarrollo Jetson TK1

2.1.2 OpenSource. OpenSource es un término que tiene origen en el contexto del desarrollo de software, para hacer referencia a tecnologías cuyo código es "abierto", es decir, que cumple con las libertades de uso, modificación y mejora, para que cualquier persona pueda usarla y distribuirla. El software libre respeta la libertad de los usuarios y la comunidad y significa que cada usuario tiene la libertad de ejecutar, copiar, distribuir, estudiar, modificar y mejorar el software. Un software es libre si los usuarios tienen las cuatro libertades esenciales: Libertad de ejecutar el programa como desee, con cualquier dispositivo, libertad de estudiar el funcionamiento del programa, y modificarlo para que se acomode a las necesidades del usuario, para ello es necesario el código fuente, libertad de redistribuir copias a otros usuarios y finalmente la libertad de distribuir copias de las versiones modificadas a terceros. (Foundation, 2001)

Es una idea de colaboración mutua, donde programadores y diseñadores mejoran continuamente el código de un programa y comparten sus cambios con el mundo. Esta filosofía surge a principios de los años 90 en la comunidad tecnológica, en respuesta al software patentado

de las empresas del momento. En Finlandia, Linus Torvlads, estudiante de ciencias en computación, inconforme con el sistema operativo "Minix", inicia la implementación de las primeras versiones para el núcleo de Linux: con un tiempo corto la idea se popularizó y equipos de trabajo enteros complementaron el sistema operativo hasta hacerlo estable (Avalos Mera Winton Wladimir, 2015). A partir de este punto se desencadenó una corriente creativa, y con la liberación de Netscape, se creó la Open Source Initiative, un grupo especializado con el objetivo de educar y trabajar por la superioridad de un proceso de desarrollado abierto, con lo cual se registró el término Open Source y se hizo más frecuente esta tecnología (Definicion.de, 2008-2018) (Elena, 2017).

2.1.2.1 Open Software. Existen varias definiciones para Open software, pero básicamente es un concepto que hace referencia a la libertad que poseen los usuarios para ejecutar, copiar, distribuir, estudiar, modificar y mejorar el software. Algunos autores denominan «software libre» (no es lo mismo que software gratis), el cual se rige bajo el contexto de libertad, más no de precio, pues la posibilidad de adquirir un valor económico por el desarrollo y su distribución de dicho software, no está prohibida. Un software libre debe estar en libre disposición para uso comercial, desarrollo y distribución comercial

En la práctica, "Software Libre" y "código abierto", son dos contextos diferentes y expresan conceptos muy diferentes. Existen grupos que denominan al Software Libre como "código abierto" o "software de fuentes abiertas" ("open source"). Aunque de algunas maneras hacen referencia a programas con similitudes, la idea que se proyecta con el término "abierto", no tienen incluidos los valores y derechos relacionados con la libertad, por lo cual la palabra "Libre" hace referencia

a dicha libertad y por lo tanto es un término que mejor se acomoda (Free Software Foundation, Inc., 2016).

2.1.2.2 Open Hardware. Como se mostró anteriormente, el open software ofrece a cada usuario cuatro libertades: libertad de uso, modificación, de estudio de distribución, y redistribución de las mejoras incluidas. En este contexto es bueno mencionar que existen licencias que garantizan y proporcionan control legal sobre estas libertades, como por ejemplo GPL (Gonzáles Iván, 2003).

El open hardware usa una filosofía muy similar en su campo de acción. Es una propuesta, hablando en términos de tiempo, a la par con el open software, pero su modelo de empleo no es tan directo. La idea de compartir diseños de software es un tanto más compleja. Richard Stallman afirma que los ideales del software libre pueden ser aplicados en sus patrones de diseño y especificación, pero no a un circuito físico en sí. Según la naturaleza, el hardware desarrollado en open hardware se puede clasificar en:

2.1.2.2.1 Hardware Estático. Si bien se conocen los componentes tradicionales de cualquier diseño de hardware, son el circuito esquemático, la información del diseño, documentación asociada y su respectivo circuito impreso; el resultado final de estos archivos de diseño es proporcionar un circuito de existencia física (algo que se puede palpar). Teniendo en cuenta esta, y otras diferencias respecto al software, hacen que surjan cierta serie de inconvenientes, entorno al deseo de usar los conceptos y licencias aplicados al software.

El hardware estático puede referirse a aquel conjunto de elementos tangibles de un sistema electrónico. Como bien se sabe, el software carece de una existencia física; motivo por el cual se

dificulta la aplicación directa de las cuatro libertades del software libre, sobre el hardware, debido a la diferencia en la naturaleza; la existencia física que tiene uno sobre el otro.

2.1.2.2.2 Hardware Reconfigurable. Se describe por medio de un lenguaje HDL, (Hardware Description Language, lenguaje de descripción de hardware), el cual permite realizar especificaciones detalladas de su estructura y funcionalidad. Su desarrollo es similar a la estructura de desarrollo del software, se realiza mediante archivos de texto, los cuales contienen su código fuente (Gonzáles Iván, 2003).

Puede aplicarse una licencia libre de manera directa, con la GPL. Las dificultades no surgen por la definición de si es libre o no, o qué debe cumplir para serlo, sino que surgen herramientas necesarias para su desarrollo. Para lograr que un hardware reconfigurable sea libre, sólo basta aplicar la licencia GPL a su código (Gonzáles Iván, 2003).

2.1.3 Visión por Computador. La visión es un mecanismo sensorial de suma importancia en los seres vivos superiores; por este motivo la visión por computadora, como un análogo en la robótica, desempeña un papel igual de importante en la informática y robótica. La visión artificial por computadora consiste básicamente en la obtención y deducción automatizada de la estructura y propiedades presentes en el mundo tridimensional, ya sea de manera dinámica, por medio de una o más imágenes bidimensionales obtenidas en dicho plano.

2.1.3.1 Sensor Kinect. Kinect (originalmente conocido por el nombre en clave «Project Natal» El 13 de junio de 2010), es «un controlador de juego libre y entretenimiento» creado por Alex Kipman, desarrollado por Microsoft para la videoconsola Xbox 360, y desde junio del 2011 para

PC a través de Windows 7 y Windows 8.3. Kinect permite a los usuarios controlar e interactuar con la consola sin necesidad de tener contacto físico con un controlador de videojuegos tradicional, mediante una interfaz natural de usuario que reconoce gestos, comandos de voz y objetos e imágenes. El dispositivo tiene como objetivo primordial aumentar el uso de la Xbox 360, más allá de la base de jugadores que posee en la actualidad. En sí el Kinect compite con los sistemas Wii motecon Wii MotionPlus y PlayStation Move, que también controlan el movimiento para las consolas Wii y PlayStation 3, respectivamente (EcuRed, 2010).

Arquitectura Kinect: Se debe tener en cuenta que la arquitectura del sensor Kinect está constituida de cuatro (4) partes fundamentales:

- Servomotor
- Una cámara RGB
- Una cámara de Profundidad, también llamado sensor 3D
- Cuatro micrófonos

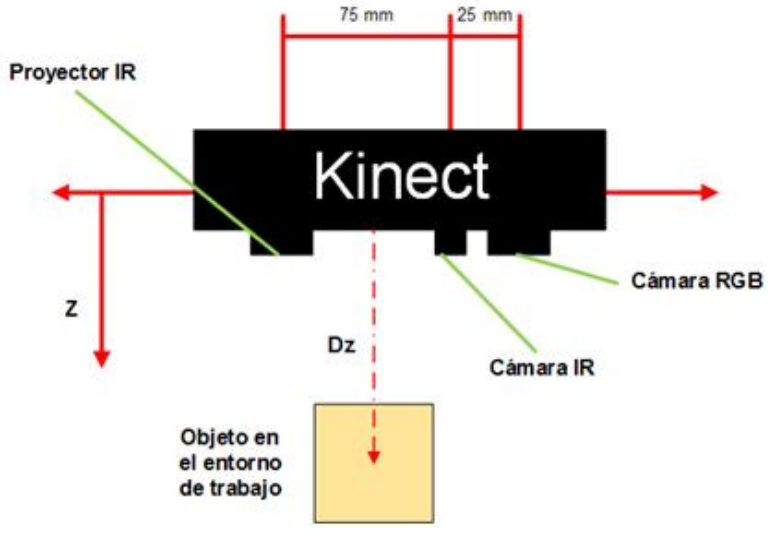

Figura 2. Componentes de arquitectura Kinect. Adaptado de SciELO (2015) http://scielo.sld.cu/scielo.php?script=sci_arttext&pid=S1815-59442015000200004

Físicamente un Kinect es un sensor en forma de rectangular, es una barra plástica negra de aproximadamente 30 cm de ancho, la cual está conectado a un cable que se bifurca en un cable USB y uno de alimentación de energía eléctrica.

Figura 3: Partes que conforman el sensor Kinect

Servomotor: Inclinación motorizada que permite ajustar la cámara hacia arriba o hacia abajo hasta 27º.

Cámara RGB: Se utiliza para capturar la resolución espacial; coordenadas en *X, Y*. Posee una resolución de 8 bits VGA (640 x 480 Píxeles), la cual opera a través de un sensor CMOS con filtro Bayer, lo cual hace posible una captura de imagen a color y el envío de datos a una frecuencia de actualización de 30 frames por segundo.

Sensores 3D de profundidad: Se compone de 2 partes, un proyector de rayos en el espectro infrarrojo y un sensor CMOS monocromático. El sensor que percibe rayos infrarrojos puede

realizar capturas de datos de vídeo en 3D bajo cualquier condición de luz. Este, además opera con resolución VGA a 16 bit de profundidad a una velocidad de 30 frames por segundo, y provee 2048 niveles de sensibilidad. Respecto al proyector de profundidad mediante infrarrojo (retícula izquierda), Sensor monocromático (retícula derecha), tiene la función de calcular la distancia en función del periodo de tiempo que tarda en ser reflejada la luz (Edwin, 2016).

Micrófono Multi-array: Cuenta con cuatro micrófonos ubicados a los extremos del sensor, cada micrófono posee un canal que puede procesar 16 bit, con un rango de muestreo de 16kHz. Por estos micrófonos es que el sensor es tan ancho y se arma como un solo micrófono, el cual es usado para reconocimiento de voz.

Los siguientes componentes no están a simple vista, sin embargo, el sensor Kinect también posee:

Chip-PRIMESENSE PS1080: La función de este chip, es reconstruir la captura de movimiento 3D de los escenarios que se ubican frente a la Kinect, el chip captura el entorno en 3 dimensiones y las transforma las capturas en imágenes sincronizadas en 3D.

Memoria RAM-de 512 Mb

Acelerómetro: Su función es estabilizar la imagen cuando está en movimiento.

Ventilador: Con la función de refrigeración, este dispositivo no está en continuo uso para no interferir con el micrófono.

Data Streams (Flujo de datos):

- 320 × 240 a 16 bits de profundidad @ 30fps
- 640 × 480 32-bit de color @30fps
- Audio de 16-bit @ 16 kHz

Funcionamiento: El sensor Kinect se comporta como un sonar, conociendo el tiempo de cada salida y llegada de la luz tras reflejarse en un objeto, la velocidad absoluta de la luz, manera como se obtiene la distancia a la cual se encuentra el objeto. En un amplio campo visual, el sensor Kinect intenta reconocer la distancia a la que se encuentran los objetos, distinguiendo de igual manera el movimiento en tiempo real. También logra distinguir la profundidad de cada objeto con diferencias de hasta 1 centímetro, su altura y el ancho con diferencias de 3 milímetros. El hardware de Kinect se compone de la cámara y el proyector de luz infrarroja, añadido al firmware y a un procesador que utiliza algoritmos para procesar las imágenes tridimensionales.

2.2 Marco Metodológico.

Para el desarrollo metodológico del proyecto se establece un proceso de diseño secuencial que va desde un punto inicial hasta uno final, y que se divide por etapas o fases. En este modelo solo es posible avanzar si se cumple con la fase anterior, sin embargo, se agrega la característica de permitir la modificación en fases anteriores con la intención de incluir mejoras en la entrega de resultados y mostrar como dichos cambios influyen en el resultado final.

A continuación, se muestra un modelo de la metodología que se implementa en el desarrollo del proyecto para cumplir con éxito los objetivos propuestos:

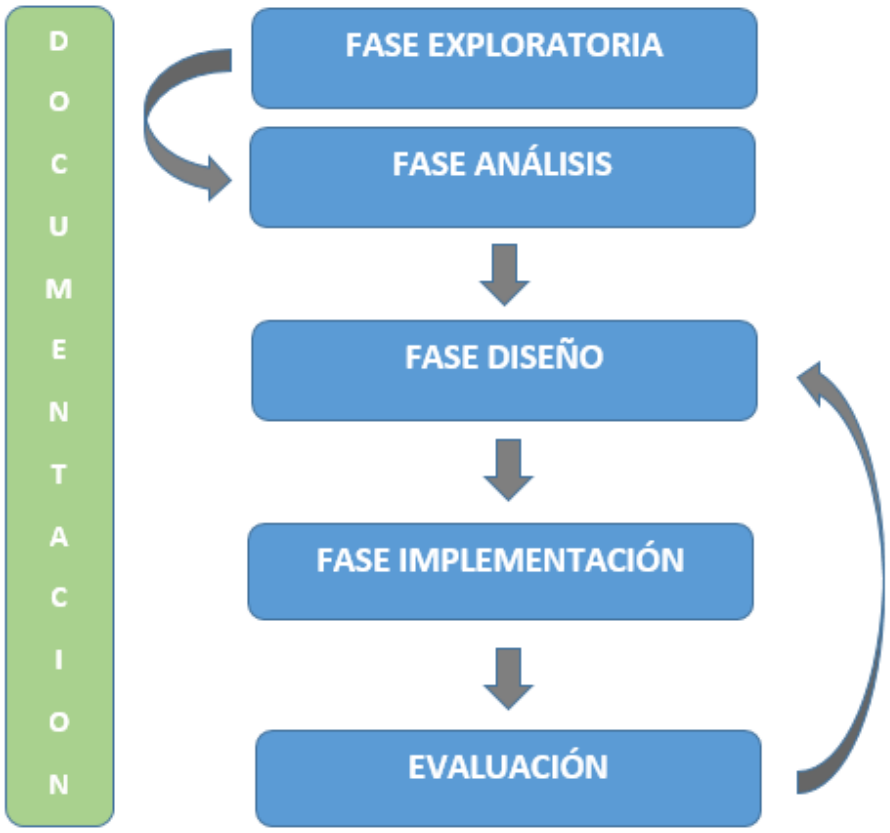

Figura 4: Diseño del modelo de metodología en cascada

2.2.1 Fase Exploratoria. La fase exploratoria corresponde a la fase de investigación donde se discute el eje temático que corresponde al área del proyecto en desarrollo. Se delimita la temática basados en los 3 pilares que soportan el proyecto: Sistemas embebidos, Arquitecturas computacionales abiertas y visón por computadora. En esta fase también se define la pregunta holopráxica que define el tema a estudiar, y esta es: ¿Cómo interactúan los componentes de visualización e interacción en ambientes embebidos implementados en arquitecturas computacionales abiertas?

2.2.1.1 Delimitar la temática correspondiente al proyecto, teniendo en cuenta los 3 pilares teóricos. Sistemas Embebidos, Arquitecturas Computacionales Abiertas y Visualización científica.

2.2.1.2 Delimitar la temática correspondiente al proyecto, teniendo en cuenta los 3 pilares teóricos. Sistemas Embebidos, Arquitecturas Computacionales Abiertas y Visualización científica.

2.2.1.3 Definir la pregunta general que encierra el enunciado holopráxico, luego de la definición del tema a estudiar ¿Cómo interactúan los componentes de visualización e interacción en ambientes embebidos implementados en arquitecturas computacionales abiertas?

2.2.2 Fase de Análisis. En esta fase se identifican las características de comportamiento funcional y desempeño de los dispositivos de captura y visualización, ya sea en entornos no embebidos y embebidos. También se plantea un análisis funcional del kit de desarrollo embebido Jetson TK1 como un módulo de conexión entre los componentes de visualización e interacción Kinect y Creative y las aplicaciones de software abierto que se planean usar, con la finalidad de crear un punto de comparación de tal manera que se pueda determinar si se logra mantener desempeño, funcionalidad y rendimiento.

2.2.2.1 Analizar las características de comportamiento y compatibilidad que tienen los dispositivos de captura y visualización sobre arquitecturas abiertas en sistemas embebidos.

2.2.2.2 Identificar los requerimientos de hardware y software que se ajustan a la demanda de procesamiento de datos producto de la visualización por computadora.

2.2.2.3 Caracterizar la funcionalidad y desempeño haciendo uso de aplicaciones de código abierto que interactúan con dispositivos de captura y visualización.

2.2.2.4 Elegir alguna aplicación de software para medir el rendimiento en un sistema operativo que funcione con políticas abiertas, teniendo en cuenta parámetros que faciliten una medida en torno a alto rendimiento, usabilidad y funcionalidad con los dispositivos de visualización.

2.2.2.5 Realizar una debida documentación obtenida de la experiencia de usar los dispositivos de visualización en la distribución Linux elegida en la arquitectura implementada.

2.2.3 Fase de Diseño. En esta fase se tiene en mente proponer una arquitectura computacional embebida que adapte los módulos de visualización como herramientas de recolección de datos para su posterior uso en las pruebas de desempeño y rendimiento, mediante software desarrollado bajo políticas de GNU.

2.2.3.1 Proponer una arquitectura computacional embebida que se ajuste a los lineamientos de bajo consumo energético, alto rendimiento, portabilidad y bajo costo.

2.2.3.2 Establecer conexión del sensor Kinect y la cámara creative con el sistema embebido propuesto por medio librerías o drivers de código abierto, que faciliten el cumplimiento de los lineamientos establecidos en los objetivos específicos.

2.2.4 Fase de Implementación.

2.2.3.1 Realizar pruebas de desempeño en la arquitectura propuesta teniendo en cuenta los lineamientos ya establecidos.

2.2.3.2 Comparar los resultados de las pruebas realizadas con el software elegido y las métricas descritas para evaluación de desempeño aplicando políticas abiertas en la arquitectura propuesta.

2.2.5 Evaluación. En esta fase se recopila la información obtenida de las fases anteriores para crear una debida documentación que respalde el cumplimiento de cada fase y de los objetivos específicos, con la finalidad de proponer conclusiones parciales que aporten una visión más amplia al momento de generar las conclusiones finales producto del desarrollo y finalización del proyecto.

3. Análisis de funcionamiento y desempeño.

Para lograr el cumplimiento del primer objetivo específico, que consta del análisis de funcionamiento y desempeño de dispositivos de adquisición de imagen e interacción de acuerdo a requerimientos que garanticen la integración con arquitecturas que sigan una política abierta, alto rendimiento, portabilidad, eficiencia energética y bajo costo, primero es necesario identificar una arquitectura embebida que cubra los requerimientos necesarios. Como garantía de cumplimiento de los lineamientos establecidos, se opta por usar el kit de desarrollo Jetson TK1, debido a ser una plataforma de desarrollo portable que cuenta con características hardware que fueron diseñadas como una plataforma de computación visual para el procesamiento acelerado por GPU basados en la arquitectura de computación Nvidia Kepler, la cual es usada en computación de alto rendimiento. Además, dicho kit de desarrollo funciona a bajo consumo energético y salió al mercado soportando una distribución Linux/Ubuntu 14.04, lo que garantiza el uso de políticas de software libre (Berger, 2002).

Una vez elegida la plataforma de desarrollo con su sistema operativo libre, se establece el uso de los drives o librerías que establecen conexión entre el kit de desarrollo Jetson TK1 y los dispositivos de visualización e interacción que se usan en el desarrollo de este proyecto (Sensor Kinect y cámara Creative). Con la finalidad de hacer una comparación entre el funcionamiento de los 2 dispositivos respecto a su desempeño, se establecen métricas que, tomando medida en tiempo de ejecución, proporcionan resultados que garantizan un apropiado análisis de funcionamiento y

desempeño de los dispositivos en la arquitectura computacional abierta que cumple con requisitos anteriormente mencionados.

3.1 Elección del sistema embebido.

Para dar un fundamento de porqué se elige el kit Jetson en el desarrollo de este proyecto, se realizan pruebas de rendimiento y estrés de hardware con la finalidad de asegurar el cumplimiento a los requerimientos que garanticen una integración con arquitecturas que siguen una política abierta, alto rendimiento, portabilidad, eficiencia energética y bajo costo. Sin embargo, en esta sección del documento se tiene en cuenta una descripción teórica de cómo se cumplen dichos lineamientos de acuerdo a las especificaciones del desarrollador y más adelante en un capitulo siguiente se demuestra con pruebas de rendimiento por qué hace de esta plataforma de desarrollo una opción óptima para la ejecución y del proyecto.

En torno a las políticas abiertas el kit de desarrollo Jetson TK1 se presentó en el NVIDIA GTC 2011 como la primera supercomputadora móvil del mundo para sistemas integrados que además soporta un sistema operativo basado en Linux para Tegra, ofreciendo una distribución Ubuntu en su kit de desarrollo. También tiene la ventaja de permitir la integración con otros sistemas embebidos, como Rasperri, para el desarrollo de arquitecturas computacionales abiertas.

Respecto al alto rendimiento el kit de desarrollo Jetson está diseñado con la misma arquitectura NVIDIA kepler, la cual consigue 3 veces más velocidad de procesamiento y eficiencia gracias al multiprocesador de streaming, que permite dedicar más espacio a los núcleos de procesamiento que a la lógica de control. Con la GPU dinámica keppler simplifica la programación en la GPU, facilitando la aceleración de bucles anidados en paralelo, es decir, la GPU puede iniciar

nuevos subprocesos de forma dinámica por sí misma, sin necesidad de volver a la CPU. Además, posee la característica de reducir el tiempo de inactividad de la CPU permitiendo que los múltiples núcleos usen una misma GPU Kepler, lo cual aporta una mejora significativa en la programabilidad y eficiencia. Con estas características se garantiza un alto rendimiento en el procesamiento de datos en el campo de la visión por computador, el cual se usa en el desarrollo de este proyecto (NVIDIA Corporation , 2015).

Finalmente, para el requerimiento de portabilidad, Jetson TK1 es un sistema embebido, característica que hace de este kit una plataforma fácil de transportar e implementar junto a otras arquitecturas computacionales debido a las características técnicas y puertos que facilitan estas conexiones. Además, este sistema embebido funciona con un voltaje de 12V, reduciendo considerablemente el consumo energético en comparación a otras arquitecturas dedicadas a HPC que inclusive en el mercado llegan a superar en precio a Jetson TK1, el cual oscila entre los U$S215, hecho que respalda un bajo costo de capital en el desarrollo de proyectos, como este, dedicados a implementación en arquitecturas computacionales abiertas aplicadas a la visión por computador (2014 NVIDIA Corporation, 2014).

Con lo anterior se elige la arquitectura computacional abierta con la cual se desarrolla el proyecto y seguido a esto se define como se establece conexión con dicha arquitectura y los dispositivos de interacción Kinect y Creative.

3.2 Conexión con el sensor Kinect.

Para establecer conexión del sensor Kinect con el sistema embebido Jetson, se usan controladores de código abierto desarrollado y distribuido por una comunidad llamada OpenKinect, la cual está

conformada aproximadamente por 2000 personas que están interesadas en permitir el uso del sensor Kinect en arquitecturas computacionales que usen distribuciones de sistemas operativos Linux, Os y Windows (2018 NVIDIA Corporation, 2018)

3.2.1 Librería LibFreenect. Libfreenect es un controlador desarrollado por OpenKinect y su función es establecer conexión del sensor Kinect con arquitecturas que se ejecutan en Linux, OSX y Windows. Teniendo en cuenta lo anterior se selecciona esta librería para establecer la conexión con el sensor, debido a su funcionamiento en diferentes arquitecturas que funcionan con sistemas operativos de código abierto. Para tener una mejor idea de cómo instalar, configurar y usar dicha librería se puede dirigir a los anexos de este documento, donde se dará una explicación detallada del uso de la librería LibFreenect, de igual manera también puede consultar la información del repositorio que se encuentra en GitHub (Piedar, 2012)

La librería permite que los desarrolladores puedan crear código usando las funcionalidades presentes en el hardware Kinect; además tiene la disponibilidad para realizar trabajos en lenguajes como C, C++, Java, Python y Ruby (Mauricio, 2016). Por este motivo, y siguiendo la política de código abierto, dicha librería se usa en el desarrollo de este proyecto.

Una vez establecida la conexión con el sensor se realizan las pruebas correspondientes al análisis de desempeño del mismo, teniendo en cuenta criterios de medición que se mencionan más adelante en la sección 3.4.

3.3 Conexión con cámara Creative.

Para la conexión de la cámara creative se usan los drivers incluidos en la librería OpenCV debido a que la cámara usada en el desarrollo de este proyecto es un prototipo desarrollado por Creative Labs (Creative Technology Ltd., 2018) y los controladores para este dispositivo fueron desarrollado únicamente para windows. Este dispositivo fue proporcionado como una herramienta para la investigación de dispositivos de visualización con componentes 3D. Dicho esto al ser una cámara prototipo se tiene poca información y especificaciones técnicas del desarrollador, no es un dispositivo que salió a la venta en el mercado, sin embargo, es una herramienta útil en el desarrollo del proyecto que aporta una idea del funcionamiento de las cámaras 3D en la implementación de casos de uso sobre la arquitectura propuesta.

A largos rasgos se puede concluir que a pesar de no tener drivers desarrollados y optimizados para distribuciones Ubuntu, las herramientas proporcionadas por la librería OpenCV resultan ser útiles para establecer la conexión y ejecutar pruebas que permitan recolectar datos que pertenecen a los frames por segundo y hacer un análisis de desempeño con los mismos.

3.4 Características de puertos en Jetson TK1.

Si bien se obtienen resultados adecuados del sensor Kinect en el kit NVIDIA Jetson TK1, es necesario dar los datos técnicos del mismo para tener un mejor punto de partida respecto al análisis funcional y desempeño de la plataforma de desarrollo, los cuales se muestran en la tabla 2.

Tabla 1.

Especificaciones técnicas de la plataforma de desarrollo TK1.

Caracteristicas Jetson TK1		
Puertos disponibles por expansión con conector 125 pines de 2 mm		**Conector frontal del Panel**
Puertos Cámara	2 Puertos fast CSI-2 Mini (uno 4-lan y otro 1-lane)	Verde: LED de encendido Naranja: LED de HDD Rojo:LED Botón de encendido Morado/Azúl: Botón Reset
Puerto LCD:	Panel de LVDS y eDP Display	API's soportadas con aceleración de hardware:
Puertos pantalla Táctil	Touch SPI 1 x4-lane + a x a-lane CSI-2	CUDA 6.0: (SM3,2 version SM3.5) OpenGL 4.4. OpenGL ES 3.1 . OpenMEM IL multimedia: códec que incluye H.264, VC-1 y VP8 atraves de Gstremer. NPP: Nvidia Performance Primitives (NPP) optimizados para CUDA. OpenCV4Tegra: NEON+GSLS+optimizaciones quad-core CPU. VisionWorks.
Puertos ISC	3 Puertos	
Dispositivo UART (Universal Transmisor-Receptor Asíncrono Universasl	Dispositivo que controla los puertos y demas dispositivos en serie. Transmisor-Receptor Asíncrono Universal	
Puertos I2C	3 puertos: Puerto de protocolo de comunicación serial, encargado de definir la trama de datos y conexiones físicas para transferencia de bits entre dispositivos digitales.	
GPIO	7 pines GPIO a 1.8 V. Los pines de cámara CSI también se pueden usar para GPIOs extra si no se usan ambas cámaras	

3.5 Análisis de funcionamiento y desempeño sensor Kinect.

Para el análisis de funcionamiento del sensor Kinect se tienen en cuenta las especificaciones técnicas proporcionadas por el fabricante. En este caso se enfoca la tasa de cuadros por segundo que el sensor es capaz de obtener cuando es puesto a prueba junto a su cámara para la detección de la profundidad. Teniendo en cuenta lo anterior se tiene que el sensor Kinect funciona a una tasa de 30 cuadros por segundo, capturando imágenes con una resolución de 640x480 píxeles contando con 8 bits por canal y produce una salida usando un filtro de Bayer conformado con un patrón

RGGBD. La cámara cuenta, con la capacidad de realizar un balance automático del objeto, supresión del parpadeo y operaciones relacionadas con saturación de color. No obstante, aunque la cámara trabaja con una resolución de 640x480, puede variarse a una resolución de 1280x1024 obteniendo imágenes con una frecuencia promedio de 10 cuadros por segundo (Pedro, 2015).

1. El proyector IR, proyecta sobre la escena en captura un patrón conocido. El patrón se almacena en memoria local del sensor y se usa como patrón de referencia para la detección de profundidad.

2. La cámara IR hace un filtro y lee únicamente el patrón que es proyectado en la escena.

3. Los valores correspondientes a profundidad para cada posición en el patrón que se obtiene son comparada dos con el patrón de referencia, los valores de profundidad para dichas posiciones se calculan por triangulación.

Para el análisis de desempeño se ejecuta una prueba donde se captura la imagen obtenida con la cámara RGB junto a la imagen obtenida por el proyector y receptor IR, encargados de hacer los cálculos para determinar la profundidad a la que se encuentra el objeto del sensor. A continuación, en la tabla 1 se describen las características funcionales del sensor Kinect, es decir, unas métricas con valores establecidos de fábrica los cuales se refieren a como se espera que funcione el sensor cuando es sometido a ejecutar alguna tarea en tiempo real.

Según el fabricante el sensor tiene la capacidad de realizar capturas de 30 fotogramas por segundo a una distancia horizontal de 2 metros a partir de las cámaras del sensor. Por medio de una aplicación llamada OpenNI, se obtienen capturas de imagen por medio de la cámara RGB de igual forma que se muestra la imagen de profundidad obtenida con el sensor.

Tabla 2.

Características de hardware Kinect

Caracteristicas funcionales Kinect V1-1473	
Caracteristicas funcionales y conexiones	
Campo de visión: Horizontal-58°H, Vertical-40°v, Diagonal-70°D	Interfaz de Datos USB 2.0. Distribución de energía USB 2.0. Interfaz de usuario USB 2.0. Consumo energético de 2.25W.
Tamaño de imagen de profundidad: VGA 640x480	
Resolución Espacial x/y (a 2m de distancia del sensor).	Rango de operación (0.8m - 3.5m).
Resolución de la profundiad x (a 2 m de distancia del sensor).	Tasa de cuadros por segundo/Rendimiento de imagen: 30 fps
Latencia promedio de la imagen con resolución VGA: 40 msec	Temperatura de operación de 0°C a 40°C.

El sensor Kinect cuenta con un mecanismo sensorial de profundidad, una Cámara RGB, un arreglo de cuatro micrófonos y un motor de inclinación. En la tabla 1 se muestran las propiedades funcionales del sensor Kinect V1. Para medir el rendimiento del sensor Kinect se hace una captura de los frames por segundo cuando el sensor está funcionando con su cámara RGB y su proyector y receptor IR al mismo tiempo tal y como se muestra en la figura 5. Como se puede apreciar al lado izquierdo de la figura 5 se ve la función del proyector y receptor IR, tomando una captura de profundidad, mientras que al lado derecho esta la misma imagen capturada, pero por medio de la cámara RGB, y bajo las 2 imágenes se ve una captura de los frames por segundo en el apartado con el título Estimated frames por second: 28.5578432249, valor que comparado con el valor

funcional descrito en la tabla 1, con un valor de 30 fps se comprueba que tiene un desempeño muy cercano al valor estimado por el fabricante a una resolución de 640x480 pixeles.

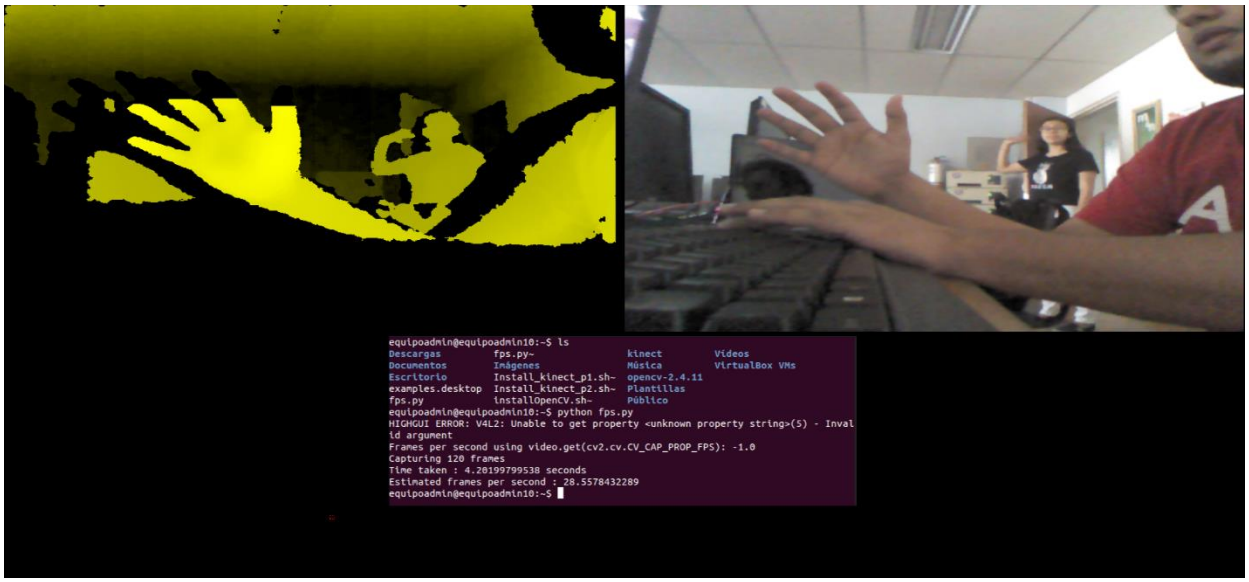

Figura 5. Frames por segundo y captura de profundidad Kinect.

Del sensor Kinect también vale la pena analizar el campo operativo del sensor y comparar los resultados obtenidos con los rangos operativos propuestos por el fabricante. Para esto se colocan figuras a similares a diferentes distancias, dentro y fuera del rango operativo del sensor. Como se ve en la Figura 6, en la toma izquierda se ve dos figuras de color blanco; la figura de atrás se encuentra en el rango operativo del sensor (de 1 m a 2.5 m de profundidad), razón por la que en la imagen de la derecha que muestra la precepción de profundidad del sensor en color amarillo, dicha figura es reconocida, a diferencia de la figura que se encuentra un poco más a delante, la cual se encuentra a 0.8 m del sensor; como se ve en la figura 6 a la derecha de la captura del proyector IR y la Cámara RGB, se aprecia que la imagen se ve en negro lo que significa que el sensor no puede realizar correctamente los cálculos de profundidad. De igual forma sucede lo mismo con las figuras

de color negro y las pelotas de color rojo. En este último caso hay algo interesante que discutir y son las zonas negras que presenta la figura de color negro que se ubica detrás de todas las demás, aunque en la captura de profundidad dicha figura se ve en color amarillo existen algunas regiones de color negro dentro de esta y eso se debe a la cantidad de iluminación presente en la sala; la luz del sol afecta notoriamente el funcionamiento del sensor, como también se puede apreciar en el color de fondo que se ve algo distorsionado en la imagen de profundidad producto de la cantidad de luz en la habitación.

Figura 6. Campo de operación del sensor Kinect

Al aislar los componentes de la cámara RGB y la cámara IR se observa que mediante el uso de librerías de código abierto y arquitecturas computacionales abiertas en un entorno Ubuntu obtenemos un desempeño aceptable al usar el sensor, comparado con la tasa de fps proporcionada por Microsoft, para el desarrollo una tarea de captura de datos en tiempo real que en este caso es obtener una imagen de una escena y a su vez los datos de profundidad de los objetos presentes en la escena que se encuentran a distancian dentro y fuera del rango operativo del sensor.

Teniendo en cuenta la existencia de 2 versiones de hardware para la versión 1, identificados con la referencia numérica 1114 y 1473, respectivamente, se concluye que la librería freenect desarrollada con la implementación de ingeniería inversa para el hardware 1114 presenta problemas de compatibilidad para la versión 1473. Respecto a la versión 1114 no se generan problemas de compilación en el momento de establecer conexión con arquitecturas que funcionan con una distribución Ubuntu por medio de libfreenect, motivo por el cual las pruebas realizadas sobre esta arquitectura presentaron resultados adecuados y aceptables al compararlos con los datos de la ficha técnica respecto a los valores obtenidos en la prueba de capturar imágenes de profundidad con la cámara IR y la imagen de la escena con la cámara RGB a una resolución de 680x480 pixeles, con una tasa de fps equivalente a 28.558 Respecto a la arquitectura Kinect 1473 se presentan algunas dificultades al momento de establecer la conexión debido a los problemas de compatibilidad del controlador freenect, aunque usando la versión de controlador desarrollado para 1114 se puede establecer conexión, las aplicaciones de prueba presentes en el directorio freenect fallan regularmente y se produce un error de acceso al dispositivo identificado con el mensaje LIBUSB_ERROR_ACCESS, sin embargo cuando se usa la aplicación OpenNI se puede acceder a las cámaras RGB e IR y proceder a realizar la captura de datos y fps para un análisis de desempeño de la arquitectura, la cual se encuentra referenciada en la Figura 5.

El motivo por el cual el proyecto para la versión 1473 fue suspendido, es por la aparición de la versión 2 de Kinect, con la cual se inicia el desarrollo de la librería libfreenect2, la cual no es compatible para los sensores Kinect para Windows v1 (1114) o Kinect para Windows Xbox360 (1473) (Xiang, y otros, 2016).

Basados en las anteriores afirmaciones se hace necesario el diseño de la arquitectura computacional la cual sigue unos lineamientos de implementacion y esquemas, estos topicos seran tratados en el siguiente capitulo.

4. Diseño de la arquitectura computacional.

4.1 Lineamientos de implementación con módulo Jetson TK1.

Como se menciona anteriormente en la sección **3.1** para cumplir con los requerimientos que garantizan una integración con arquitecturas que siguen una política abierta, alto rendimiento, portabilidad, eficiencia energética y bajo costo, se desarrolla una serie de pruebas de rendimiento sobre el módulo TK1 para garantizar que efectivamente se cumplen los lineamientos de implementación definidos en los objetivos específicos de este proyecto.

Como descripción por parte del fabricante Nvidia se tiene que el kit de desarrollo Jetson está diseñado con la misma arquitectura NVIDIA kepler, la cual consigue 3 veces más velocidad de procesamiento y eficiencia gracias al multiprocesador de streaming, que permite dedicar más espacio a los núcleos de procesamiento que a la lógica de control. Con la GPU dinámica keppler simplifica la programación en la GPU, facilitando la aceleración de bucles anidados en paralelo, es decir, la GPU puede iniciar nuevos subprocesos de forma dinámica por sí misma, sin necesidad de volver a la CPU. Además, posee la característica de reducir el tiempo de inactividad de la CPU

permitiendo que los múltiples núcleos usen una misma GPU Kepler, lo cual aporta una mejora significativa en la programabilidad y eficiencia (Nvidia Corporation, 2015).

4.1.1 Análisis funcional Jetson TK1. Previamente a las pruebas de rendimiento que se realizan en el módulo TK1, se muestra la información correspondiente a las especificaciones técnicas del mismo, para una mejor idea entre los valores reales obtenidos al realizar dicha prueba y los valores esperados de acuerdo a las capacidades del hardware. Estas características se ilustran en la tabla 3.

Tabla 3.

Información del hardware Jetson TK1

Processor	Graphics	Motherboard	Memory	Disk	Operatin System
ARMv7 rev 3 @2,32GHz Four Cortex-A15 cores	192 852MHz Kepler CUDA cores	Jetson-TK1 Nvidia	2048 MB	16GB SEM16G	Ubuntu 14.04
Core Count: 4	Display Driver: Nvidia 1.0.0	Chipset: Nvidia TegraK1		File-System: ext4	Kernel: 3.10.40-g8c4516e
Scalling Driver: tegra interactive	Screen: 1920x2400	Network: Realtek RTL811/8168/8411		data=ordered relatime rw	Display Server: X Server 1.15.1 Compiler GCC 4.8.4

Para ejecutar las pruebas de rendimiento se elige una aplicación llamada Phoronix Test Suit, que es un software de código abierto para Linux que soporta más de 220 perfiles de pruebas y cuenta con 60 suits de prueba, además permite cargar los resultados de las pruebas realizadas de manera organizada, lo que permite observar los resultados de manera clara y ordenada y formular las conclusiones debidas e inclusive realizar una tabla de comparación de estos datos con otros sistemas en los que se ejecuta la misma prueba.

4.1.2 Pruebes de estrés de Hardware con Phoronix. En la ejecución de la primera prueba se instala el software phoronix para el desarrollo de las pruebas de estrés de Hardware. Una vez instalado el software se elige una prueba de la suit de pruebas que hacen parte de Phoronix. Una aclaración antes de realizar las pruebas es que se detiende el proceso que proporciona una interfaz gráfica, puesto que el uso de una interfaz en el sistema operativo consume recurso hardware al ejecutarse y esto interviene en los resultados finales de la prueba de estrés.

Teniendo en cuenta lo anterior se elige una de las pruebas de las suit; para listar los comandos y pruebas presentes en el software se ejecuta una terminal y ejecutamos *phoronix-test-suite list-tests.* Ahora se selecciona una de las herramientas de pruebas, para este caso inicio con una prueba de rendimiento de CPU,y GPU respectivamente.

4.1.2.1 Prueba de CPU. Para esta prueba se elige el test C-ray (Comparions, 2014) (AnandTech, 2016), el cual es un rastreo de rayos para sistemas Linux escrita por John Tsiombikas y está diseñado para que, en la mayoría de los sistemas no necesite de la memoria RAM, por lo que es muy sensible al rendimiento del procesador.

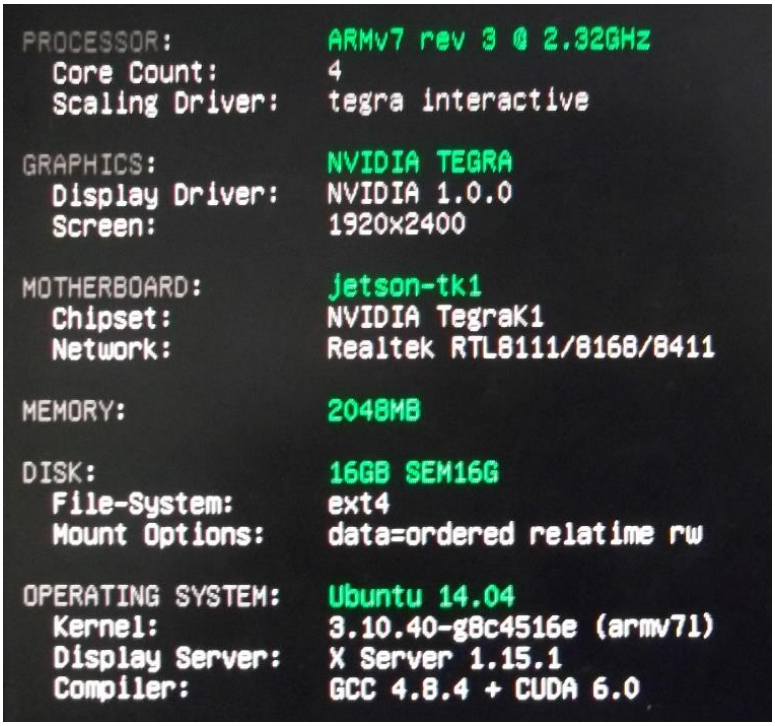

Figura 7: Descripción del sistema al ejecutar la prueba c-ray

Luego de mostrar la información del sistema automáticamente se inicia la prueba y se muestra en pantalla algo similar a lo que muestra la Figura 6. Esta prueba mide el tiempo que tarda en ejecutar las 3 veces el código tras digitar el comando c-ray, en este caso la primera prueba da un aproximado de 59 minutos para finalizar y es el tiempo total que tarda en hacer 3 veces la ejecución del código correspondiente al test.

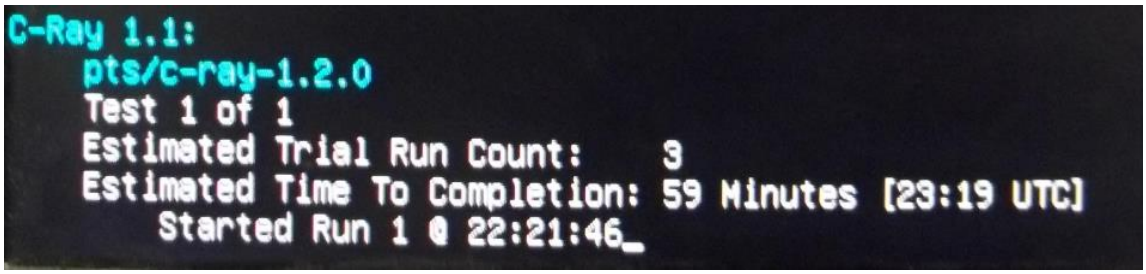

Figura 8: Ejecución de la prueba c-ray en Jetson TK1

4.1.2.1.1 Análisis de la primera prueba. Con los datos obtenidos de la primera prueba se hace una tabla para comparar los resultados de los tiempos de ejecución con otros sistemas embebidos, como Rasperry Bi 2 B, Libre ALL-H3CC H5, Rasperri Pi 3 B, Pine64 1GB, Firely ROC, AML-S905X-CC Le Potato, ODROID-C2, Tinker Boarad, Core i7 3517UE, NVIDIA Jetson TX2. Estos datos se pueden observar en la Figura 7. Cabe mencionar que en el marco del proyecto se tiene al alcance el módulo Jetson TK1 y la placa Raspberry, los demás datos se recopilan de pruebas realizadas por otras usuarios y almacenadas en la base de datos de OpenBenchmarking, claramente ejecutando la misma prueba y con un análisis previo de estas se establecen los parámetros para realizar las pruebas en TK1 bajo condiciones similares.

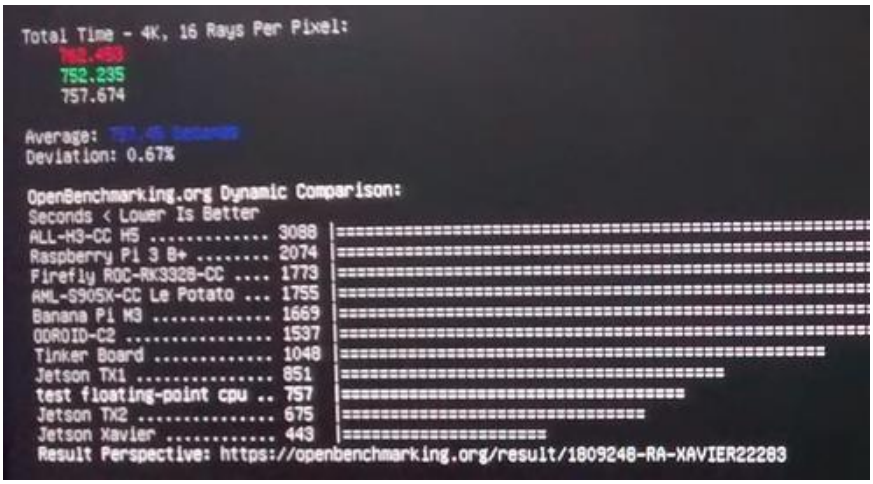

Figura 9: Tiempo que tardó la primera prueba c-ray en ejecutarse en diferentes sistemas embebidos

En la primera fase de la primera prueba se muestran 3 tiempos de las 3 ejecuciones del código C-ray, el primer resultado obtenido es de 762.453 segundos, el segundo muestra una tiempo de 752.235 segundos y en el tercero se obtiene un tiempo de 757.674 segundos y en general se tiene un Average de 757.45 segundos, este último corresponde al promedio de los 3 tiempos anteriores y es un resultado con 0.67% de Deviation o error en la prueba, el cual equivale a una cantidad baja

por lo que los resultados obtenidos tienen una precisión que se puede considerar óptima. Finalmente se presenta una tabla con los tiempos de ejecución de la prueba sobre otras arquitecturas embebidas, donde se resalta el hecho de que entre menor sea el tiempo en segundos el rendimiento es mucho mejor, dejando al Kit Jetson TK1, con el nombre de test floating-point CPU en la Figura 7, como un sistema con mejor desempeño frente a dispositivos Raspberry, Firely, ODROID, entre otros.

4.1.2.1.2 Análisis de la segunda prueba. De igual forma como en la sección 4.1.2.1.1, se obtiene una tabla de resultados con los tiempos tras ejecutar 3 veces el código correspondiente a la prueba C-ray, en este caso se obtienen tiempos de 766.497, 752.165 y 752.808 segundos de los 3 test respectivamente pero con la diferencia de que se obtiene un Average de 757.16 segundos y una Deviation de 1.07%, un error considerablemente más alto que el obtenido en la primera prueba. Sin embargo, se mantiene un rendimiento superior de TK1 respecto a sistemas Raspberry, Firely y ODROID.

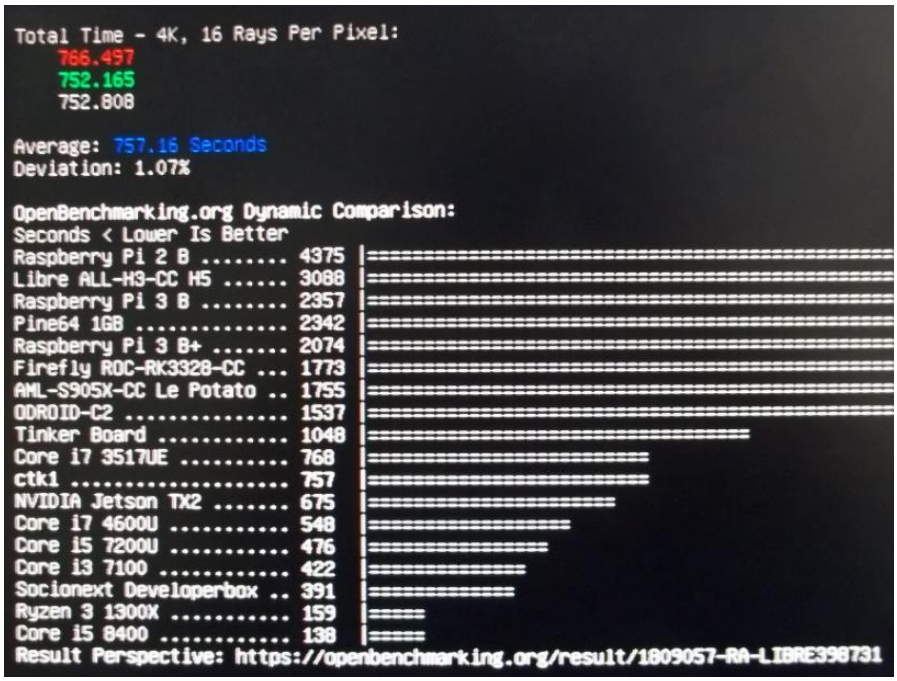

Figura 10: Tiempo que tardó la segunda prueba c-ray en ejecutarse en diferentes sistemas embebidos

4.1.2.1.3 Análisis de la tercera prueba. En la prueba final se obtienen tiempos de 751.794, 753.824 y 751.981 segundos en las 3 nuevas respectivas pruebas con un Average de 757.16 segundos, pero esta vez con una Deviatión de 0.15%, considerablemente más bajo que en la primera prueba y de igual manera conservando la posición el kit TK1 frente a los sistemas Raspberry, Firely y ODROID nuevamente.

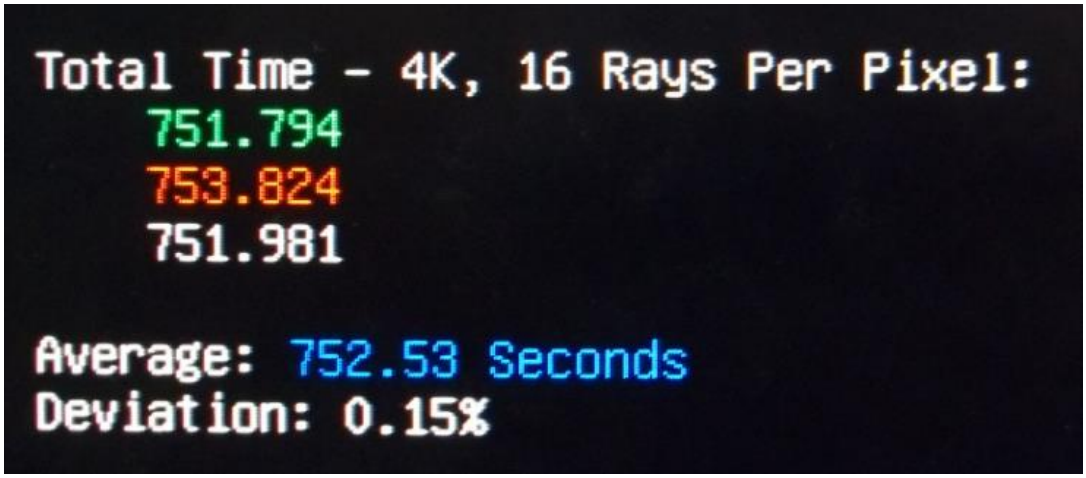

Figura 11: Tiempo que tardó la tercera prueba c-ray en ejecutarse en diferentes sistemas embebidos

Finalmente, obtenidos los resultados de todas las pruebas de rendimiento se ordenan en la tabla 4 con sus respectivas gráficas con la finalidad de mostrar de manera más ordenada la información y sacar las debidas conclusiones.

4.1.2.1.4 Tabla y gráficas de las pruebas con C-ray. En esta sección se ilustran en tablas y gráficos los resultados numéricos obtenidos de la fase anterior y se formulan las debidas conclusiones al final de cada gráfica.

La tabla 4 contiene los datos de tiempo en segundos, en donde cada fila contiene la cantidad de tiempo en segundos que tarda en finalizar cada prueba con C-ray en los sistemas embebidos ALL-M3-CC H5, Raspberry Pi 3, Firely ROC, AMLO Le Potato, ODROID, Jetson TX1 y Jetson TK1. En este caso se realiza un total de 3 pruebas donde se tienen 3 datos diferentes de tiempo por cada sistema embebido en el cual se ejecuta dicho test.

Tabla 4.

Información tiempos obtenidos en sistemas embebidos diferentes

ALL-M3-CC H5	Raspberry Pi 3 B	Firely ROC-RK3328-CC	AML-S905X-CC Le Potato	ODROID-C2	Jetson TX1	Jetson TK1
3088	2074	1773	1755	851	851	757,45
3087	2357	1773	1755	1537	850	757,16
3088	2150	1773	1755	902	851	752,53

Para el sistema ALL-M3 se tienen 3 datos correspondientes al tiempo que tarda en finalizar el algoritmo C-ray, en la primera prueba se obtiene un tiempo de 3088 segundos, la segunda prueba el tiempo es de 3087 segundos y en la última se tiene un tiempo de 3088 segundos. Para el sistema Raspberry Pi 3 se obtienen tiempos de 2074, 2357 y 2150 segundos para la primera, segunda y tercera prueba de C-ray respectivamente. Con el sistema Firely se toman tiempos de 1773, 1773 y 1773 segundos en las 3 respectivas pruebas. El sistema AML Le Potato tiene tiempos de 1755 segundos en sus tres pruebas. Con el sistema ODROID si exite un caso interesante en los tiempos obtenido y es que en su segunda prueba se obtiene un tiempo demasiado alto comparado con los otros 2, esto puede ser ocasionado por el tiempo de espera en el uso de las actividades propias del uso de los E/S.

En la siguiente tabla se hace una comparación de los tiempos en segundos obtenidos por las 9 pruebas realizadas con el comando C-ray sobre la placa Jetson TK1 y se muestra el promedio de los resultados que se tienen. El valor del Average muestra que en promedio tarda 755.715 segundos en terminar la ejecución del algoritmo y una prueba completa se compone de ejecutar este algoritmo 3 veces sobre la arquitectura, en este caso sobre Jetson TK1.

Tabla 5.

Información tiempos obtenidos de las 9 pruebas en TK1

Kit de desarrollo Jetson TK1	
Número de prueba	Tiempo en segundos
primera prueba	762,457
segunda prueba	752,235
tercera prueba	757,674
cuarta prueba	766,497
quinta prueba	752,165
sexta prueba	752,808
séptima prueba	751,794
octava prueba	753,824
novena prueba	751,981
Average	755,715

De la tabla 5 se obtienen los datos para organizar en un diagrama de barras para ver claramente como por prueba han variado los tiempos de ejecución. Se debe tener en cuenta los errores obtenidos en las tres pruebas con C-ray, con valores de 0.67%, 1.07% y 0.15% respectivamente, se puede concluir que el tiempo que tarda en ejecutar el algoritmo de c-ray oscila alrededor del valor 755.715 segundos.

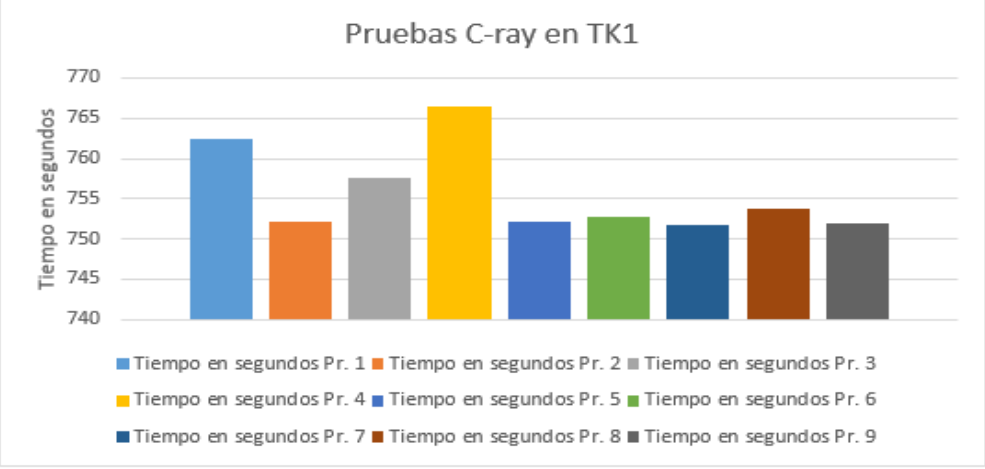

Figura 12: Tiempo en segundos al ejecutar el algoritmo C-ray en Jetson TK1

Si análizamos los tiempos obtenidos en todas las pruebas es evidente una diferencia relativamente alta en los tiempos de la prueba 1, la primera en color azul, y la prueba 4, en color amarillo, esta diferencia intencional se debe a que en esas dos pruebas no se mato el proceso grafico del sistema operativo, razón por la cual existe un aumento significativo en el tiempo que tarda en ejecutarse la prueba de estrés a la CPU. En las demás pruebas, exceptuando ahora la 3, en color gris claro empezando desde la izquierda, que nuevamente tiene un tiempo de ejecución un poco más alto en comparación a los demás, pero no tan alto como las pruebas 1 y 4; bien si antes se menciona que el error se debe al proceso gráfico del sistema operativo, para la prueba 4 si se mata este proceso entonces ¿ A que se debe este resultado? El algoritmo c-ray permite aislar el comportamiento de las operaciones de punto flotante y medir la eficiencia energética y capacidad de proceso de la CPU, cuando se realiza el test se debe asegurar que no existan otras aplicaciones en ejecución pues pueden tomar ciclos de CPU que afectan los resultados del Benchmark y es un gran problema que ocasiona una variación relativamente alta. Otra causa puede también ser producida por un fallo de energía en el momento de la prueba, tambien es recomendable aislar el dispositivo de comunicación a la red y verificar que exista espacio en la memoria cache L1-L2-L3, dependiendo de la CPU, por lo tanto el estado de las memorias afectan significativamente los resultados obtenidos en la prueba.

Finalmente se muestra una tabla con los tiempos de ejecución de las 3 pruebas totales en los diferentes sistemas embebidos, comparando las pruebas realizadas en la TK1 contra otras pruebas realizadas por otras personas y que fueron sacadas de la base de datos de OpenBenchmarking. En estos datos hay un dato curioso y se refiere a los tiempos obtenidos en TK1 y TX1, poniendo a TK1 con mejores resultados que TX1 tal vez debido a que se mato el proceso gráfico en TK1.

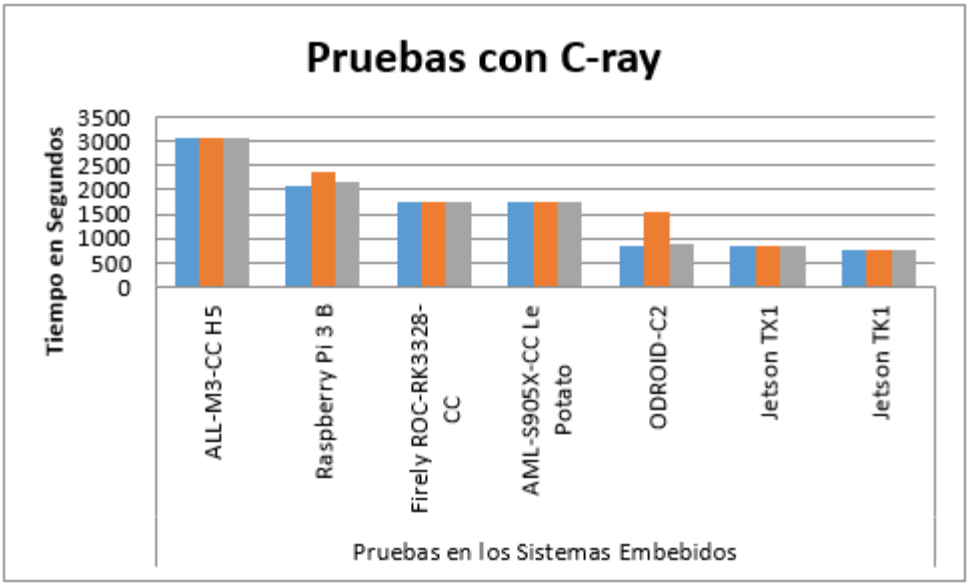

Figura 13: Tiempos obtenidos al ejecutar C-ray en diferentes sistemas embebidos

Cada tiempo obtenido en las pruebas realizadas a los diferentes sistemas se hace aplicando el algoritmo c-ray, sin embargo, se debe resaltar que en este proyecto solo se realizan pruebas para el módulo NVIDIA TK1 y los demás resultados se obtienen de pruebas realizadas bajo los mismos parámetros y se encuentran en la página de OpenBenchmarking referenciada a continuación (Phoronix Media, 2018), donde no solo se realizan pruebas para c-ray sino tambien existe información de benchmark para otros test de estrés, incluido el módulo de desarrollo Jetson TK1.

4.1.3 Pruebas de GPU con CUDA Examples. Si bien en la sección 4.1.2 se realizan pruebas que garantizan un desempeño aceptable o adecuado, en comparación con otros sistemas embebidos, las pruebas realizadas solo lo demuestran para el uso de la CPU, motivo por el cual en esta sección se ejecutan ejemplos de simulaciones con CUDA para medir y observar el funcionamiento de la GPU. Primero CUDA es una arquitectura de cálculo paralelo de Nvidia que

aprovecha la potencia de la GPU para proporcionar un incremento en el rendimiento del sistema (Nvidia Corporation, 2015).

Para esta prueba se usa el CUDA Toolkit, que proporciona un entorno de desarrollo para crear aplicaciones aceleradas por GPU, en esta sección no se desarrolla ninguna aplicación, sino que se usan ejemplos desarrollados por Nvidia. Las bibliotecas de CUDA aceleradas por GPU permiten la aceleración en varios dominios, ya sea algebra lineal, procesamiento de imágenes y video, el aprendizaje profundo y el análisis gráfico (NVIDIA Corporation, 2015).

Las pruebas para la GPU con las simulaciones en CUDA se hacen probando 3 simulaciones de los ejemplos, la primera prueba consta de una simulación de fluidos a una resolución de 512x512 pixeles, tendremos en cuenta los fps de 4 muestras tomadas de la simulación para obtener una media de los frames, la segunda prueba es una simulación con N-body, que básicamente es una simulación gravitacional de N cuerpos y como última prueba se corre sobre la placa Jetson TK1 una simulación de partículas de humo, donde se tienen en cuenta fps cuando se muestra la simulación con 65536 partículas de humo.

4.1.3.1 Simulación CUDA/GL Stable Fluids.

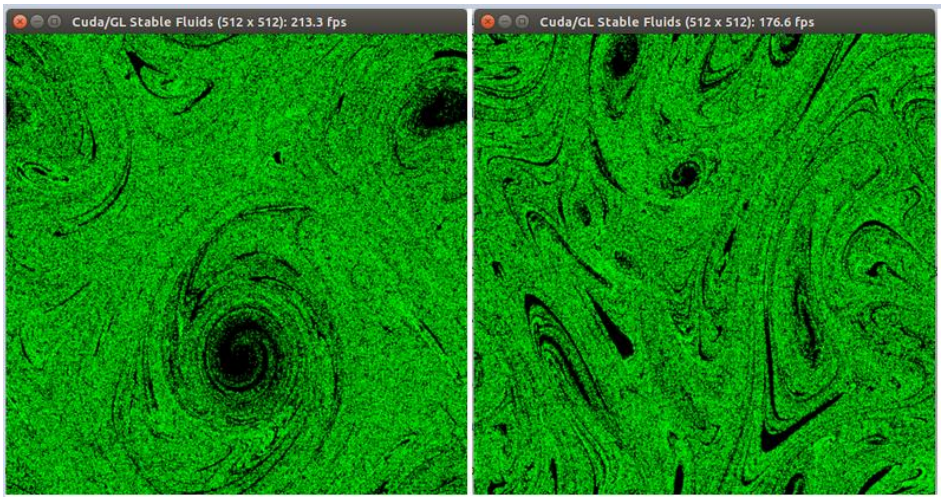

Figura 14: Muestras 1 y 2 de la simulación Stable Fluids de CUDA

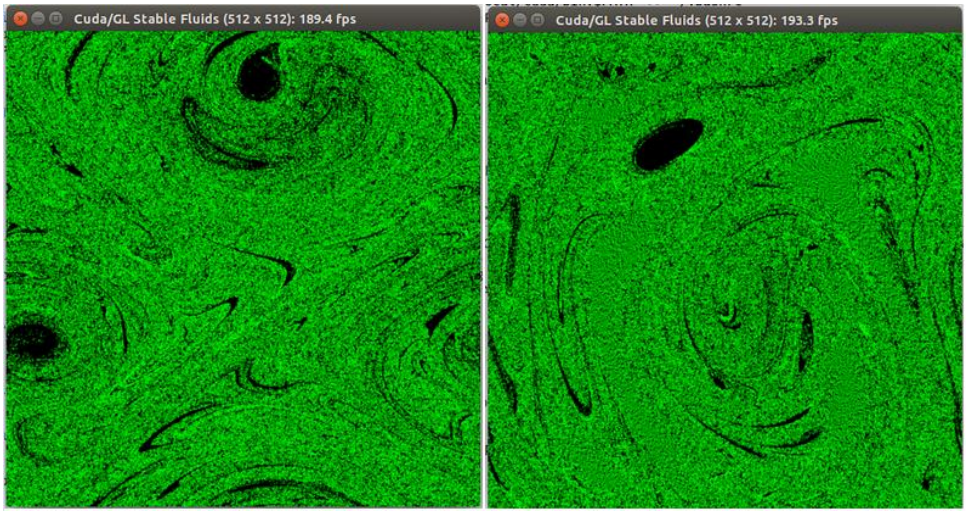

Figura 15: Muestras 3 y 4 de la simulación Stable Fluids de CUDA

Para esta prueba se pone en marcha una simulación de un fluido estable a una resolución de 512x512 pixeles, en el transcurso de la prueba es posible agitar el fluido con el puntero del mouse y crear movimiento en el fluido, esto con la finalidad de ver a cuantos fps se ejecuta la simulación.

En la Figura 13 se genera un movimiento diferente en el fluido y se toman los fps; para la muestra uno se obtiene una cantidad de 213.3 fps y para la segunda muestra un total de 176.6 fps; como se puede observar la muestra 2, a la derecha de la Figura13, posee mayor alteración en las estructura del fluido, razón por la cual se ve una notable disminución en los frames por segundo de la toma.

En la Figura 14 se ve los frames de las tomas 3 y 4, correspontientes a los valores de 189.4 y 193.3 fps respectivamente. Los datos anteriores se ven de manera más ordenada en la Tabla 6 para hacer una mejor comparación de los resultados obtenidos.

Tabla 6.

Fotogramas por segundo, simulación Stable Fluids CUDA

CUDA/GL Stable Fluids	
Muestra	FPS
Muestra 1	213,3
Muestra 2	176,6
Muestra 3	189,4
Muestra 4	193,3

Con los datos organizados en la tabla 6, una mejor forma de comparar los fps obtenidos es en un gráfico como en la Figura 15; además cabe mencionar que el promedio de frames por segundo de la simulación en el instante de tiempo en el que se toman las muestras es de 193.15 fps.

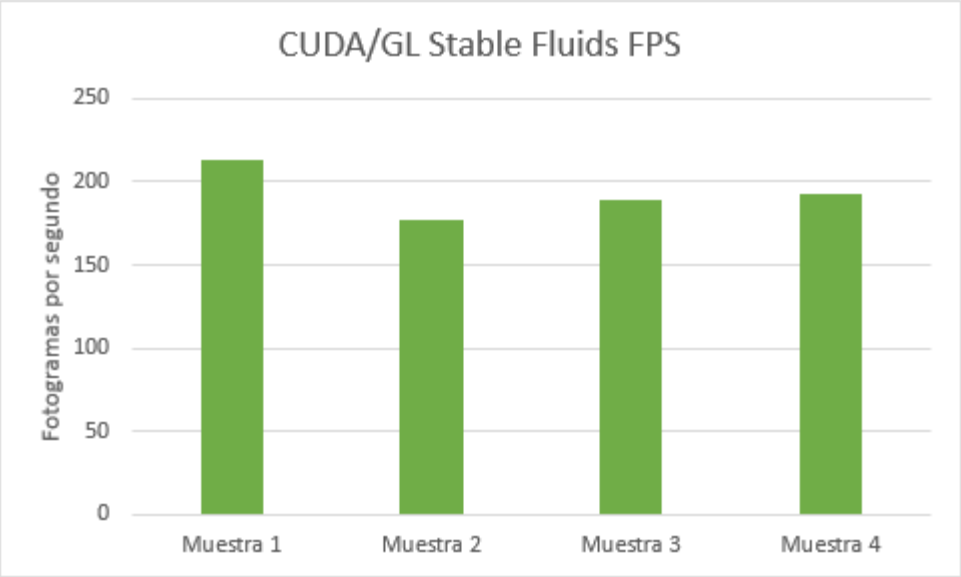

Figura 16. Gráfica de fps CUDA/GL Stable Fluids

Teniendo los datos ordenados en un diagrama de barras como se muestra en la figura 15, se procede a hacer un análisis de los frames por segundos obtenidos de cada prueba. Como primer paso se ejecuta la simulacion con el fluido estático, esto nos da un valor de 230 fotogramas por segundo, luego se agita el fluido para aplicar movimiento a la simulación tal y como se ve en la Figura 13 con la muestra 1; en este caso se reducen la cantidad de fotogramas por segundo a un promedio de 213 y esto se debe a que interacuan muchos más elementos en la simulación que en la primera fase con el fluido en estado de reposo. Si bien en la Figura 13 con la muestra 2 vemos que los fps se reducen a un total de 176 aproximadamente se debe a que en este caso la simulación del fluido ha sido mayormente perturbada, si comparamos la captura obtenida en la muestra 1 con la muestra 2, fácilmente se puede apreciar que el fluido de la muestra 2 presenta mayor movimiento en la superficie lo que significa que entran una cantidad mayor de particulas a interactuar que en la primera muestra y esto directamente significa que se hacen muchos más procesos de cálculo en la GPU y por esta razón hay una baja significativa en la cantidad de frames. Ahora si observamos

la muestra 3 y 4 en la Figura 14 vemos que efectivamente entre menor sea perturbada la superficie del fluido, mayor es la cantidad de frames por segundo, y esto seguirá asi hasta que el fluido se estabilice mostrando nuevamente los fps obtenidos al principio de la simulación, cuando el fluido este en total calma.

4.1.3.2 Simulación CUDA N-Body 1024 Bodies. N-body es una simulación que se aproxima numéricamente a la evolución de un sistema de cuerpos en el cual cada cuerpo interactúa continuamente con los demás. Un ejemplo es una simulación astrofísica donde los cuerpos representan galaxias o estrellas individuales, y cada cuerpo se atraen entre sí a través de la ciencia computacional. (NVIDIA Corporation, 2015).

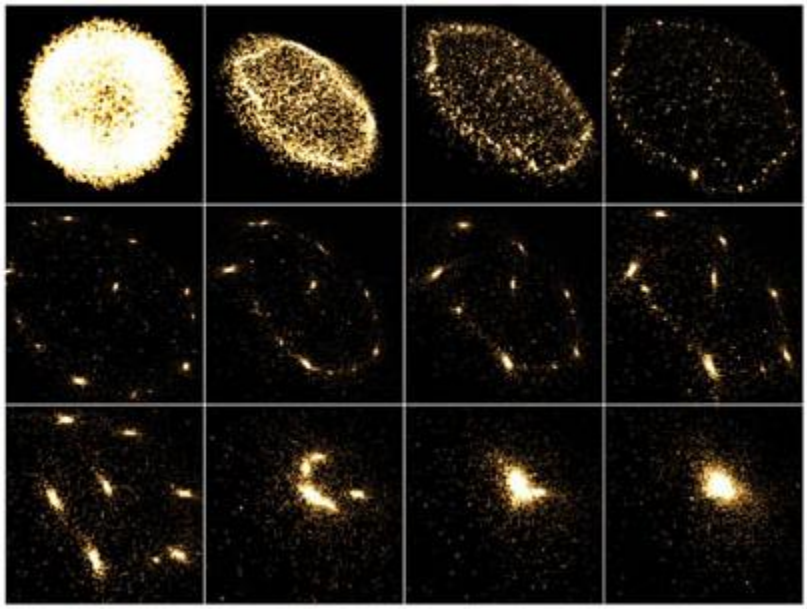

Figura 17: Cuadros de una representación 3D N-Body. Adaptado de GPU Gems (2003). Nvidia/https://developer.nvidia.com/gpugems/GPUGems3/gpugems3_ch31.html

Para el desarrollo de la prueba de rendimiento con N-Body en TK1 se tiene en cuenta las variables de Point Size, Velocity Damping, Softening Factor, Time Step, Cluster Sclae y Velocity Scale, como parámetros que afecta directamente a nuestros datos de interés, que en este caso son los frames por segundo, los GFLOP/s (Gigaflops por segundo) y los BIPS. Antes de continuar es necesario definir las unidades de medida mencionasdas, como los Gigaflpos, que es una medida de velocidad para los computadores y se define como el miles de millones de operaciones de punto flotante por segundo y los BIPS, que se define como la cantidad de instruciones por segundo en miles de millones (Ordenadores y Portátiles, 2014) .

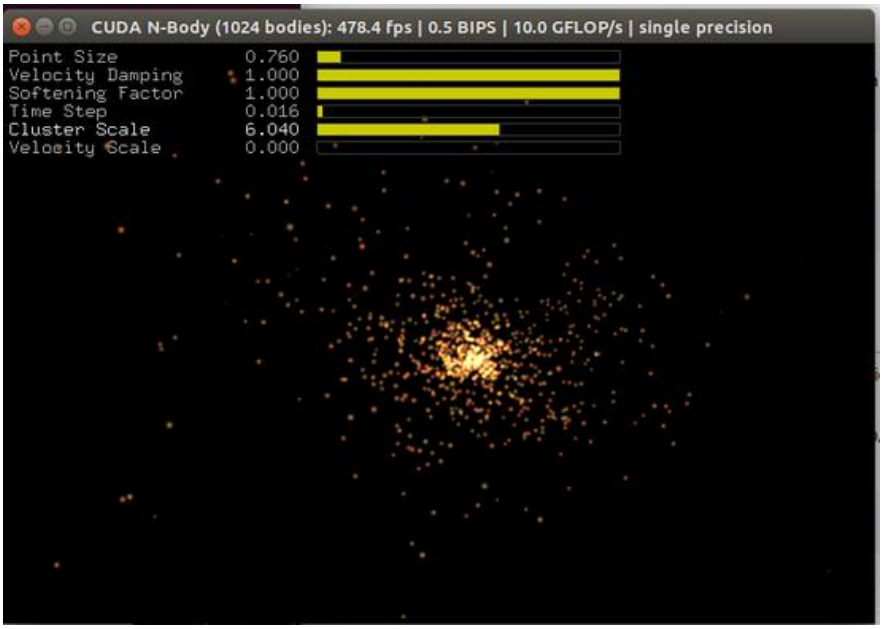

Figura 18: Prueba N-Body en TK1 con 1024 cuerpos.

En la Figura 17 se muestra como con parámetros establecidos en la simulación, referentes al tamaño de punto, la velocidad y la cantidad de cuerpos, entre otros, se obtiene una tasa de 478.4 frames por segundo, de lo cual cabe destacar que se realizan 10 mil millones de operaciones por segundo en la GPU, es decir a una velocidad de 10 Gflpos a 0.5 mil millones de instrucciones por

segundo (BIPS), resultados que nuevamente respaldan el hecho de que el kit Jetson TK1 es una potente herramienta de computo en el procesamiento de datos que requieran GPU.

4.1.3.2 Simulación CUDA Smoke Particles. Para el desarrollo de la tercera prueba se visualiza una simulación de partículas de humo paralelizado en CUDA, en la cual están interactuando 65536 partículas a tasas de 21.6, 23.2, 21.1 y 22.9 cuadros por segundo, datos obtenidos de sacar 4 muestras en tiempo de ejecución.

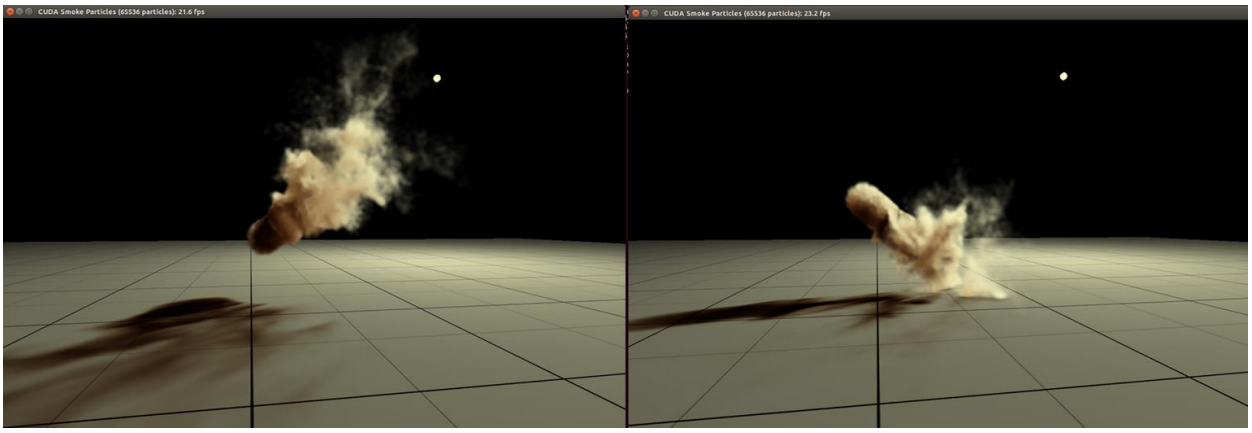

Figura 19: Muestra 1 y 2 de Smoke Particles con CUDA.

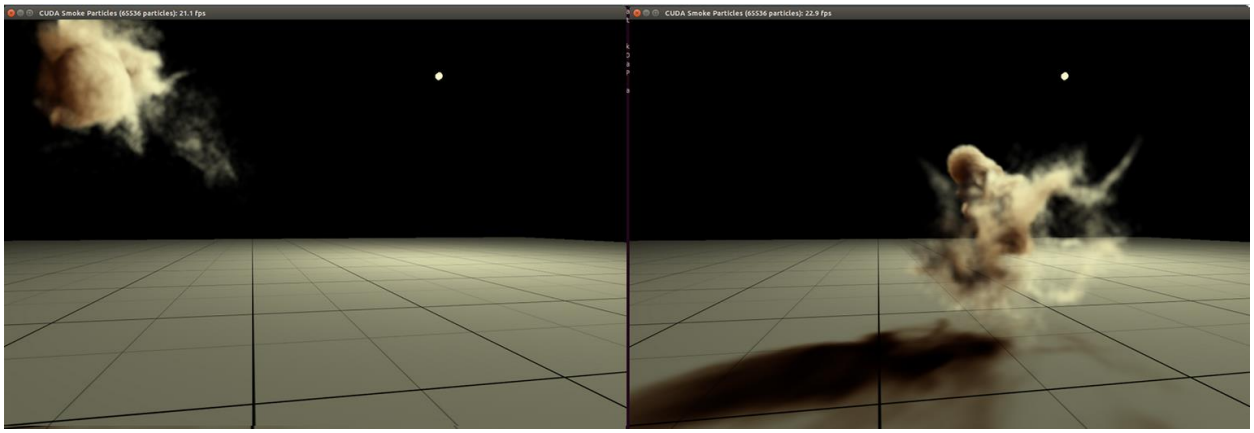

Figura 20: Muestra 3 y 4 de Smoke Particles con CUDA.

Esta simulaicón resulta ser un poco más exigente en procesamiento de datos con GPU. Si bien en la simulación Nbody tan solo se interactúa con 1024 cuerpos, en Smoke Particles interactúan 65535 partículas, y en este caso se puede realizar una conclusión con Nbody porque de igual manera se está estudiando el desempeño de la GPU al observar interacción con partículas presentes en la simulación. Nbody con tan solo 1024 particulas puede ejecutarse a un aproximado de 435 fps mientras que Smoke lo hace con casi 64 veces más la cantidad de partículas de Nbody, y en las muestras que se tomaron se ve una reducción muy significativa en los fps pero resaltando el hecho que aún mantiene una fluidez significativa en el video que genera la simulación y cabe resaltar que se va encontrando un limitante en el kit alrededor de los 70000 cuerpos, que de igual manera es un número considerablemente alto de partículas que interactúan aumentado la cantidad de cálculos en la GPU.

Finalizadas las pruebas de rendimiento, de lo anterior se concluye que los valores cuantitativos en los datos obtenido demuestran que TK1 es un potente kit de desarrollo con capacidades suficientes para el procesamiento de gráficos garantizando así el cumplimiento de los lineamientos anteriormente establecidos. Con lo anterior finalmente se elige TK1 como la plataforma adecuada para el diseño de arquitectura embebida con políticas abiertas, que sirve como un puente para la conexión de los dispositivos de visualización e interacción, que se hablará en el siguiente apartado de este capítulo.

4.2 Esquema de la arquitectura computacional.

Teniendo en cuenta el cumplimiento de los lineamientos establecidos se diseña un esquema de la arquitectura computacional embebida Jetson TK1. A continuación se presenta un esquema de

arquitectura donde se muestran los elementos usados en la realización del proyecto, se hace una breve descripción de las tecnologías usadas en cada parte del desarrollo y el motivo por el que se llegó a usar como una solución que satisface las necesidades establecidas en la descripción del problema y el cumplimiento de los objetivos.

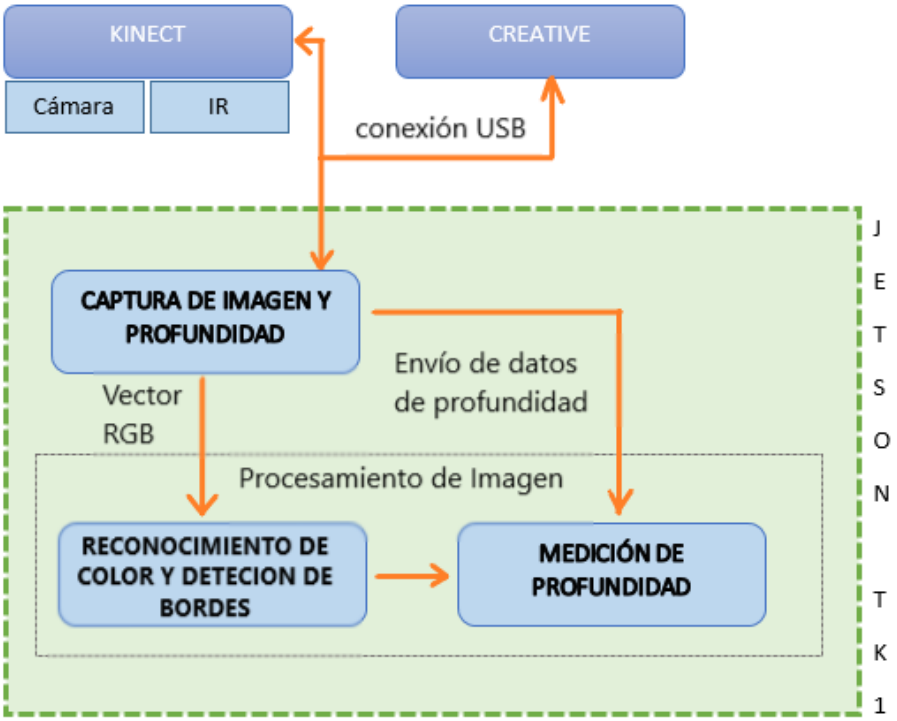

Figura 21: Diagrama de bloques del proyecto

El sistema general usado se ve reflejado en el diagrama de bloques de la figura 4.2, donde el kit de desarrollo Jetson TK1 cubre las necesidades de alto rendimiento, bajo consumo, portabilidad y bajo costo, esto se puede corroborar con la información técnica comentada en un segmento anterior del documento. El kit Jetson adquiere imágenes en tiempo real del sensor Kinect o la cámara Creative para capturar una imagen de color con la respectiva información usada en el proceso de segmentación y los datos de profundidad para el cálculo de la distancia. Como primer

paso y luego de la captura de la imagen, se hace el proceso de selección del área de interés de la imagen descartando aquellos colores que se considera información innecesaria y se procede a reconocer el objetivo con esta imagen, luego se toma la posición del píxel que se encuentra en el centro del área de interés objetivo y se procede a medir la posición con la información de profundidad.

5. Implementación de la arquitectura propuesta con Jetson TK1.

Para lograr la conexión modular de los dispositivos de visualización e interacción, es necesario instalar sobre la placa Jetson TK1 un sistema operativo basado en Linux para posteriormente instalar y configurar, tanto dependencias como las librerías que establecen la conexión vía USB entre el sensor Kinect y la cámara Creative con el kit de desarrollo TK1.

5.1 Flash Jetson TK1 con LT4/ NVIDIA Linux4Tegra.

Antes de empezar con el flash a la tarjeta Jetson TK1 es necesario un host que tenga instalado Ubuntu en la versión 14.04 y un cable micro USB, generalmente suministrado con el kit Jetson. Para mejores resultados puede consultar la información proporcionada en https://developer.nvidia.com/linux-tegra-rel-21.

Como primer paso se abre una terminal en el host con Ubuntu 14.04 y se crea un directorio con el nombre de LT4 y se accede al mismo, esto se hace para guardar los archivos del sistema

operativo del Kit. Para crear el directorio y acceder a la carpeta LT4 creada se ejecutan en terminal los comandos *mkdir ~ /LT4* y *cd ~ /LT4.*

Una vez creada la carpeta y accediendo a ella se descargan los paquetes de soporte de la placa (BSP) NVIDIA Linux4Tegra(LT4) y el sistema de archivos de muestra de NVIDIA con los siguientes comandos en terminal:

```
wget http://developer.download.nvidia.com/mobile/tegra/l4t/r21.1.0/Tegra124_Linux_R21.1.0_armhf.tbz2
wget http://developer.download.nvidia.com/mobile/tegra/l4t/r21.1.0/Tegra_Linux_Sample-Root-Filesystem_R21.1.0_armhf.tbz2
```

Figura 22: Comando sistemas de archivos muestra NVIDIA

Una vez descargados los archivos se extraen con los siguientes comandos:

```
sudo tar xpf Tegra124_Linux_R21.1.0_armhf.tbz2
cd Linux_for_Tegra/rootfs
sudo tar xpf ../../Tegra_Linux_Sample-Root-Filesystem_R21.1.0_armhf.tbz2
cd ../
```

Figura 23: Comando de extracción de los archivos muestra NVIDIA

Luego se aplican los binarios:

```
#sudo is important here
sudo ./apply_binaries.sh
```

Figura 24: Aplicación de los archivos binarios.

Luego de aplicar los binarios se conecta el kit Jetson TK1 al computador host via cable micro-usb y se enciende el módulo en modo recuperación; esto se logra manteniendo presionado el botón Recovery y Power de la placa. Cuando se enciende la placa escriba en consola el comando *lsusb* de la computadora host y asegurese de ver algún dispositivo NVIDIA.

Si efectivamente encuentra algún dispositivo NVIDIA significa que todo hasta el momento se ha configurado con éxito y siendo así se ejecuta el comando:

```
sudo ./flash.sh jetson-tk1 mmcblk0p1
```

Figura 25: Flash Jetson TK1

Cuando finalice el proceso de instalación se ve en la terminal del host *Reset the board to boot from internal eMMC* en dicho caso se presiona el botón de Reset en la placa y si todo esta bien debe iniciar la placa con la distribución 14.04 Linux4Tegra.

Ya configurado el sistema operativo en la placa TK1, actualice el software del sistema operativo y se procede a la configuración de la librería libfreenect en el sistema embebido.

5.2 Configuración librería Libfreenect.

Antes que nada, cabe mencionar que existen 2 versiones de la librería libfreenect. La primera versión está dedicada al Kinect versión 1 y el Kinect de Xbox 360 y la segunda versión de la librería está diseñada para la versión 2 de Kinect. En la implementación de los dispositivos de visualización e interacción de este proyecto se usa la versión de Xbox 360 y el sensor Kinect V2.

5.2.1 LibFreenect para Kinect de Xbox 360. Antes de construir la librería para su uso con el sensor Kinect es necesario tener instalado libusb en una versión igual o superior a la 1.0.18, Cmake en versión igual o superior a la versión 2.8.12 y finalmente Python, superior a 2.7 o 3.3.

Libusb es una blibioteca desarrollada en C que proporciona un acceso genérico a dispositivos USB; su uso está destinado a facilitar la producción de aplicaciones que requieren hardware de comunicación por medio de conexión USB. Mediante un único api multiplataforma, proporciona el acceso a los dispositivos USB en sistemas operativos basados en Linux, OS X, Windows, Android, OpenBSD, etc. (Libusb, 2012-2018).

Cmake es un conjunto de herramientas desarrolladas en código abierto con el propósito de compilar, probar y empaquetar software, además de ser multiplataforma. Cmake se usa para controlar un proceso de cmpilación de software por medio de una plataforma simple y los archivos de configuración independientes al compilador, y generar makefiles junto a espacios de trabajo nativos que se usan para un entorno de compilación a elección del usuario (CMake, 2015).

Adicionalmente y producto de la investigación para el desarrollo de este proyecto también son requeridas las librerías git, freeglut3, pkg-config, build-essential, libxmu-dev, libxi-dev, libudev-dev, libusb-1.0.0-dev, gcc y g++.

Instaladas las paqueterías mencionadas se procede con la configuración de la librería. Para un informe más detallado paso a paso de la instalación puede consultar el apéndice # de este documento o consultar directamente en el repositorio en GitHub presente en el siguiente enlace https://github.com/OpenKinect/libfreenect .

5.2.2 LibFreenect para Kinect V2. Al igual que el apartado anterior, la configuración de libfreenect2 tambien requiere de cumplir algunas especificaciones antes de realizar su

construcción. Respecto al hardware, es necesario contar con puerto 3.0 oues el 2.0 ya no es compatible con esta versión. En el desarrollo del proyecto libfreenect2 se descubrió que la librería funciona con host Intel y NEC USb 3.0 y que para los controladores ASMedia no funciona.

Respecto a la implementación en máquinas virtuales, el uso de la librería no es funcional y eso se debe a la transferencia isócrona de USB 3.0, la cual es bastante delicada. (Xiang, y otros, zenodo, 2016).

Teniendo en cuenta lo anterior, se procede a la construcción de la biblioteca mencionada; para una información más detallada puede consultar directamente en el repositorio en GitHub que pertenece al apartado de LibFreenect2 en el siguiente link https://github.com/OpenKinect/libfreenect2/blob/master/README.md#requirements.

Como primer paso se descarga la librería en el sistema, se aconseja crear una carpeta llamada Kinect acceder a ella desde la terminal y clonar la carpeta libfreenect2 en ella con el comando *git* **clone https://github.com/OpenKinect/libfreenect2.git**, terminada la descarga se accede al directorio con el comando **cd libfreenect2**. Para versiones de Ubuntu 14.04 se accede al directorio depends para descargar y actualizar los Deb files con el comando cd "**depends; ./download_debs_trusty.sh".**

Finalizado el paso anterior se instalan los builds tolos necesarios, en este caso se hace con el comando **"sudo apt-get install build-essential cmake pkg-config".** Para el siguiente paso hay que tener en cuenta que para el funcionamiento adecuado de la librería se debe instalar el libusb =>1.0.20 y si está usando la versión 14.04 use el comando **"sudo dpkg -i debs/libusb*deb"** si usa una versión más reciente de Ubuntu solamente use el comando "**sudo apt-get install libusb-1.0-0-dev".** Instalado el libusb se debe instalar el TurboJPEG, si está usando la versión 14.04/16.04 debe usar en la terminal el comando **"sudo apt-get install libturbojpeg libjpeg-turbo8-dev"** de

lo contrario use **“sudo apt-get install libturbojpeg0-dev”.** Para la instalación de OpenGL, para Ubuntu 14.04 solamente **“sudo dpkg -i debs/libglfw3*deb; sudo apt-get install –f”** y para otras versiones **“sudo apt-get install libglfw3-dev”.**

Finalizadas las configuraciones e instalaciones anteriores, se ingresa al directorio libfreenect2 y se crea un nuevo directorio llamado build para acceder al mismo, esto se hace con el comando **“mkdir build && cd build”.** Creado el directorio build y estando en el procedemos a compilar los make de libfreenect con el comando **“cmake .. -DCMAKE_INSTALL_PREFIX=$HOME/freenect2 && make”,** y finalmente usamos el comando **“sudo make install”.**

Para finalizar se configuran las reglas de udev para el acceso al dispositivo con el comando **“sudo cp ../platform/linux/udev/90-kinect2.rules /etc/udev/rules.d/”** y ahora desconecte y vuelva a conectar el plug USB de Kinect y para probar su funcionamiento ejecute el programa de prueba **“./bin/Protonect”.** (Xiang, y otros, Zenodo, 2016).

6. Aplicación del diseño de arquitectura con Jetson.

Como una de las etapas finales en el desarrollo del proyecto, se identifican casos de uso para la implementación de lo anteriormente enunciado y estudiado. En esta sección se da una idea final del para que se ha diseñado una arquitectura computacional abierta y embebida que permite la conexión modular de los sensores estudiados para 2 aplicaciones, una que fue desarrollada por un grupo de estudiantes de la universidad Javeriana del departamento de electrónica y otra que se

encuentra en una fase pre limar desarrollada como un futuro proyecto de grado en el grupo CAGE, cuya finalidad es por medio de una de las cámaras realizar capturas del entorno estelar para determinar la existencia de cuerpos celestes fugaces.

6.1 Reconocimiento de color y profundidad con el sensor Kinect.

Como se menciona anteriormente la primera aplicación que se prueba en la arquitectura desarrollada, consta de un proyecto ya desarrollado por un grupo de estudiantes de la universidad Javeriana, proyecto en el que se diseña y contruye un sistema dedicado a la visión artificial para una posterior aplicación en la robótica móvil, por medio de la cual logran ensamblar y controlar un robot de prueba con la capacidad de recoger objetos esféricos de color verde y naranja. El desarrollo del robot se logra usando una tarjeta Raspberry Pi en la cual se procesan las imágenes y datos de profundidad adquiridos con el sensor Kinect, y establecen comunicación serial para transmitir la información recolectada a un microcontrolador ARM cortex de VEX. (García Preciado Paola Andrea, 2016).

No obstante, en la aplicación a este proyecto se usa la aplicación de reconocimiento de objetos y áreas de interés de una imagen obtenidas con el sensor Kinect. Para la elaboración del sistema de visión artificial lo primero que se debe lograr es adquirir las señales que capta el sensor Kinect, y para lograr la conexión con el sensor se hace uso de la librería LibFreenect que se menciona en la sección 5.2.1 y la herramienta de desarrollo Python para usar en Linux e instalarla sobre el kit de desarrollo Jetson TK1 (García Preciado Paola Andrea, 2016). Para el procesamiento de las imágenes, es necesaria la librería OpenCV (OpenCV team, 2018). Para mayor información acerca del desarrollo del proyecto de reconocimiento de color y posición con un sensor Kinect para

aplicación de robótica móvil puede consultar el siguiente enlace https://repository.javeriana.edu.co/handle/10554/21442.

Para la captura de imágenes del sensor Kinect, se usa la función frame_RGB(): y frame_depth(): para retornar las matrices obtenidas de la imagen RGB y la imagen de profundidad captada con el proyector y receptor IR. Como ejemplo de las capturas adquiridas con el sensor conectado al kit Jetson TK1 se aprecian en la figura 24.

Figura 26. Captura de profundidad y RGB en Jetson TK1

Obtenido lo anterior se demuestra la conexión con el módulo Jetson teniendo encuentra el uso de librerías desarrolladas en código abierto, además se encuentra una aplicación para el uso de la arquitectura diseñada. Para esta sección solo se muestra la imagen de profundidad obtenida con el sensor pues la finalidad era identificar un caso de uso de acuerdo a los lineamientos propuestos y de los cuales en secciones anteriores se hace su respectivo análisis funcional y desempeño, y este ejemplo de uso ayuda a ver la explotación de la arquitectura, No obstante existe una documentación con mayor profundidad en el documento ya mencionado, donde se tienen en cuenta filtros de color, trasformación de espacios de color, algoritmos para detección de bordes y las respectivas pruebas de rendimiento implementadas, temas que salen del contexto de la investigación y diseño de la arquitectura que se propone en el desarrollo de este proyecto de grado.

Para finalizar la breve descripción de este caso de uso se anexa el código para el reconocimiento de color y profundidad obtenido del documento de grado 1548-Reconocimiento de color y posición con un sensor Kinect para aplicación de robótica móvil desarrollado por los estudiantes Paola Andrea García Preciado y Sergio Nicolás González Forero de la universidad Pontificia Javeriana.

7. Conclusiones

Al analizar el funcionamiento y desempeño de los dispositivos, se concluye que las librerías desarrolladas para la versión 1 de libfreenect, haciendo uso de ingeniería inversa, funcionan mejor y aportan resultados aceptables en la versión 1 de Kinect u 1114 que en la versión de Xbox 360 o 1473, aunque esto no significa que no funcione para la versión 1473. Respecto a la versión 2 de la librería se ve una mejora significativa, adaptada únicamente para la segunda versión del sensor Kinect, con la cual se planea realizar un estudio más a fondo a futuro, pues es un dispositivo al que se tuvo acceso finalizando el proyecto pero que es un aporte significativo a trabajos futuros con la implementación de la arquitectura computacional propuesta con TK1. Ahora teniendo en cuenta las limitantes para la cámara creative debido a ser un prototipo de cámara no comercial, los controladores genéricos presentes en la librería OpenCV son una herramienta que facilita la conexión con la cámara y que además permitieron realizar pruebas de desempeño y conocer las limitaciones de este dispositivo, aun sin tener especificaciones técnicas del mismo.

Analizando las pruebas realizadas en el módulo Jetson TK1, los tiempos obtenidos en la prueba con C-ray para CPU; se concluye que se obtienen mejores resultados cuando no se trabaja en el entorno gráfico, pues no intervine en los tiempos de ejecución de las operaciones de punto flotante, realizados en la CPU. De estos tiempos se puede decir que las diferencias en los valores son producto de procesos secundarios que ocupan la CPU durante el desarrollo de la prueba sumado a que la cantidad de memoria disponible en la memoria caché no es suficiente para almacenar todos los datos necesarios para la ejecución, aunque el algoritmo trata de aislar el procesador para realizar las operaciones, también se debe tener en cuenta que dicho programa es una caja negra, dicho eso, el proceso trata de funcionar de tal manera que puede segmentar los datos y almacenarlos para ser procesados en una cola de prioridad pero no hay manera de garantizarlo, por ello, los tiempos de ejecución varían.

A lo largo de la exploración de las capacidades de computo de la GPU en el módulo TK1 se desarrollan unas pruebas de simulación, con la finalidad de medir el desempeño de los gráficos y conceptualizar una idea del límite de explotación de la GPU. En cada simulación se presentan datos específicos, para tener un mejor análisis de cada dato obtenido, como por ejemplo con la simulación de un fluido estable que luego de ser expuesto a perturbaciones en su superficie, se percibe un cambio significativo en la tasa de fotogramas por segundo, y esto se debe a la cantidad de procesos que debe ejecutar la GPU para responder a los cambios en el fluido, dando así una idea general de cómo responde la Tegra al aumento significativo de las operaciones de punto flotante consecuencia de las alteraciones producidas, en otras palabras se puede inferir que se obtienen resultados óptimos en el procesamiento de gráficos. Además de la prueba mencionada, también se tiene en cuenta una simulación donde existen representaciones matriciales de cuerpos que interactúan entre sí, y se obtienen datos de 0.5 mil millones de instrucciones por segundo,

realizando un aproximado de 10 mil millones de operaciones, motivo por el cual se concluye que la GPU Tegra tiene la capacidad de computo necesaria para cubrir el lineamiento de alto rendimiento, además de realizar estos cálculos a bajo consumo energético, un punto a favor importante en la implementación de esta arquitectura con los dispositivos de visualización e interacción que se usan en el desarrollo de este proyecto.

Referencias Bibliográficas

Alegre Enrique, G. P. (2006). *Conceptos y Métodos en visión por computador.* España.

AnandTech. (16 de Junio de 2016). *AnandTech*. Obtenido de https://www.anandtech.com/show/10353/investigating-cavium-thunderx-48-arm-cores/18

Avalos Mera Winton Wladimir, J. S. (2015). *Diseño e implementación de guías de usuario para los procesos de mantenimiento preventivo y correctivo de computadores en el laboratorio de redes y mantenimiento de la universidad técnica de cotopaxi extensión la Maná.* La Maná: Tesis de Grado- Univiersidad Técnica de Cotopaxi.

Berger, A. S. (2002). *Embedded systems design, an introduction to process, tool and techniques.* CRC Press.

CMake. (28 de Julio de 2015). *Cmake.org*. Obtenido de https://cmake.org/

Comparions, S. P. (28 de Octubre de 2014). *Future Technology Research*. Obtenido de http://www.futuretech.blinkenlights.nl/c-ray.html

Creative Technology Ltd. (2018). *Creative Labs*. Obtenido de https://sg.creative.com/corporate/about?_ga=2.106507459.1341142154.1539903766-144102482.1539903766

Definicion.de. (2008-2018). *Definición.DE*. Obtenido de https://definicion.de/open-source/

EcuRed. (15 de Noviembre de 2010). Obtenido de http://www.pulso.uniovi.es/wiki/index.php/Kinect

Edwin, M. L. (2016). *Programa de procesamiento de imágenes adquiridas por medio del sensor Kinect para determinar la posibilidad una víctima en determinada zona.* Bogotá.

Einnews. (Junio de 13 de 2010). *EIN World News Report*. Obtenido de https://world.einnews.com/pr_news/56709028/microsoft-fully-unveils-kinect-for-xbox-360-controller-free-game-device

Elena, D. (24 de Octubre de 2017). Obtenido de https://blogthinkbig.com/tendencias-de-la-tecnologia-open-source

Foundation, F. S. (2001). *El sistema Operativo GNU*. Obtenido de https://www.gnu.org/philosophy/free-sw.es.html

Free Software Foundation, Inc. (18 de Noviembre de 2016). *GNU*. Obtenido de http://www.gnu.org/philosophy/categories.es.html

García Preciado Paola Andrea, G. F. (2016). *Reconocimiento de color y posición con un sensor Kinect para la aplicación de robótica móvil.* Bogotá.

Gonzáles Iván, G. J.-A. (Septiembre de 2003). *Hardware libre: clasificación y desarrollo de hardware reconfigurable en entornos GNU/Linux.* Madrid. Obtenido de http://www.iearobotics.com/personal/juan/publicaciones/art4/pres-hardware-libre.pdf

Libusb. (2012-2018). *libusb*.

Mauricio, M. F. (2016). *Implementación de videojuegos como herramienta para el desarrollo motor y cognitivo de niños.* Buenos Aires.

NVIDIA Corporation . (Abril de 2015). *NVIDIA.* Obtenido de https://la.nvidia.com/object/jetson-tk1-embedded-dev-kit-la.html

NVIDIA Corporation. (09 de Abril de 2014). *Nvidia Developer*. Obtenido de https://www.nvidia.com/object/jetson-tk1-embedded-dev-kit.html

Nvidia Corporation. (2015). *NVIDIA*. Obtenido de KEPLER: La arquitectura de HPC más rápida del mundo: https://www.nvidia.es/object/nvidia-kepler-es.html

NVIDIA Corporation. (2018). *NVIDIA*. Obtenido de https://www.nvidia.es/object/nvidia-kepler-es.html

Nvidia Corporation. (6 de agosto de 2015). *NVIDIA*. Obtenido de https://www.nvidia.es/object/cuda-parallel-computing-es.html

NVIDIA Corporation. (6 de Agosto de 2015). *NVIDIA*. Obtenido de https://developer.nvidia.com/cuda-toolkit

NVIDIA Corporation. (Noviembre de 2014). *NVIDIA*. Obtenido de http://developer.download.nvidia.com/embedded/jetson/TK1/docs/3_HWDesignDev/JTK1_DevKit_Specification.pdf

Nvidia Corporation. (s.f.).

OpenCV team. (2018). *OpenCV*. Obtenido de https://opencv.org/

Ordenadores y Portátiles. (2014). *Ordenadores y portatiles*. Obtenido de 2014 Ordenadores y Portátiles

Pastor, M. L. (2017). Configuración y ejecución de algoritmos de visión artificial en latarjeta Nvidia Jetson TK1 DevKit. En M. L. Pastor, *Configuración y ejecución de algoritmos de visión artificial en latarjeta Nvidia Jetson TK1 DevKit* (pág. 18). Madrid.

Pedro, M. F. (2015). *Evaluación del Sensor Kinect para Aplicaciones Médicas de Análisis de la Expresión Facial*. Cuba.

Phoronix Media. (5 de Septiembre de 2018). *OpenBenchmarking*. Obtenido de http://openbenchmarking.org/result/1809057-RA-LIBRE398731

Piedar, R. R. (7 de Marzo de 2012). *OpenKinect*. Obtenido de https://openkinect.org/wiki/Main_Page

Raspberry Pi Foundation. (2015). *Raspberry Pi Blog*. Obtenido de https://www.raspberrypi.org/documentation/hardware/raspberrypi/README.md

Xiang, L., Echtler, F., Kerl, C., Wiedemeyer, T., Lars, hanyazou, . . . Yuan. (28 de Abril de 2016). *zenodo*. Obtenido de https://zenodo.org/record/50641#.W8o-0fa22Uk

Xiang, L., Echtler, F., Kerl, C., Wiedemeyer, T., Lars, hanyazou, . . . Yuan. (28 de Abril de 2016). *Zenodo*. Obtenido de https://zenodo.org/record/50641#.W8n1Bva22Uk

Apéndices

Apéndices A. Script bash para la configuración de libfreenect y OpenNI en Ubuntu

```
#Instalación y configuración de LibFreenect y OpenNI para Ubuntu
#Instalar los repositorios Universe
Sudo apt-add-repository universe
Sudo apt-get update
#Dependencias necesarias para compilar la biblioteca libfreenect
sudo apt-get -y install git sudo apt-get install cmake freeglut3-dev pkg-config build-essential libxmu-dev libxi-dev libusb-1.0-0-dev
#Crear un directorio llamado Kinect y acceder a el
mkdir ~/kinect
cd ~/Kinect
#Clonar el repositorio de Git libfreenect y acceder al directorio con la libreria
git clone https://github.com/OpenKinect/libfreenect.git
cd libfreenect/
#Construcción de la libreria libfreenect
mkdir build; cd build
#Para compilar la aplicación OpenNI se usa el commando acontinuación, si quiere compilar
#los demás makefile usar cmake ..
```

```
cmake .. -DBUILD_OPENNI2_DRIVER=ON && make

cmake .. && make

sudo make install

sudo ldconfig /usr/local/lib64/

sudo adduser $USER video

sudo adduser $USER plugdev

#Clonar OpenNI

cd ~/kinect

git clone https://github.com/occipital/OpenNI2.git

#Instalar las dependencias necesarias para OpenNi

sudo apt-get -y install libudev-dev freeglut3-dev doxygen graphviz

cd ~/kinect/OpenNI2/

make

#Copiar el driver Freebect en el directorio OpenNi

cp -L ~/kinect/libfreenect/build/lib/OpenNI2-FreenectDriver/libFreenectDriver.so
~/kinect/OpenNI2/Bin/x64-Release/OpenNI2/Drivers/
```

Apéndices B. Configuración de libfreenect para usar con Python

```
#Intalar las dependencias necesarias para su uso con Python
sudo apt-get install -y install cython python-dev python-numpy
#Acceso a la carpeta donde se encuentra el archive de instalación
cd kinect/libfreenect/wrappers/python
#Instalar libfreenect para python
sudo python setup.py install
```

Apéndices C. Instalación y configuración de OpenCV

```
#Actualizar las paqueterías
sudo apt-get update
sudo apt-get upgrade
#Instalar las OS Libraries
#Remover cualquier instalación previa de x264
sudo apt-get remove x264 libx264-dev
#Intalación de las dependencias necesarias
sudo apt-get install -y build-essential checkinstall cmake pkg-config yasm git gfortran
libjpeg8-dev libjasper-dev libpng12-dev
#si está usando Ubuntu 14.04
sudo apt-get install libtiff4-dev
# Si está usando Ubuntu 16.04
sudo apt-get install libtiff5-dev
#Continuar con la instalación de las dependencias
sudo apt-get install -y libavcodec-dev libavformat-dev libswscale-dev libdc1394-22-dev
libxine2-dev libv4l-dev libgstreamer0.10-dev libgstreamer-plugins-base0.10-dev qt5-default
libgtk2.0-dev libtbb-dev libatlas-base-dev libfaac-dev libmp3lame-dev libtheora-dev
libvorbis-dev libxvidcore-dev libopencore-amrnb-dev libopencore-amrwb-dev x264 v4l-utils
#Instalar las librerías Python
```

```
sudo apt-get install -y python-dev python-pip python3-dev python3-pip

sudo -H pip2 install -U pip numpy

sudo -H pip3 install -U pip numpy

#Descargar opencv desde GitHub

git clone https://github.com/opencv/opencv.git

cd opencv

git checkout 3.3.1

cd ..

#Descargar opencv_contrib desde GitHub

git clone https://github.com/opencv/opencv_contrib.git

cd opencv_contrib

git checkout 3.3.1

cd ..

# Compilar e instalar OpenCV con los modulos contrib

cd opencv

mkdir build

cd build

#Run CMake

cmake -D CMAKE_BUILD_TYPE=RELEASE \

        -D CMAKE_INSTALL_PREFIX=/usr/local \

      -D INSTALL_C_EXAMPLES=ON \

      -D INSTALL_PYTHON_EXAMPLES=ON \

       -D WITH_TBB=ON \
```

```
      -D WITH_V4L=ON \
      -D WITH_QT=ON \
    -D WITH_OPENGL=ON \
    -D OPENCV_EXTRA_MODULES_PATH=../../opencv_contrib/modules \
    -D BUILD_EXAMPLES=ON ..
#Compilar e instalar
# find out number of CPU cores in your machine
nproc
# substitute 4 by output of nproc
make -j4
sudo make install
sudo sh -c 'echo "/usr/local/lib" >> /etc/ld.so.conf.d/opencv.conf'
sudo ldconfig
```

Apéndices D. Código de reconocimiento de color y objetos

```
# -*- coding: utf-8 -*-
 # Se importan las librerias a usar
 from freenect import*
 from numpy import*
 from cv2 import*
 from time import*
 try:
          from skimage import filters
 except ImportError:
          from skimage import filter as filters
from skimage.transform import hough_circle
import sys

#Funcion de Adquisicion RGB kinect
 def frame_RGB():
          array,_ = sync_get_video()
          array = cvtColor(array,COLOR_RGB2BGR)
          return array
```

```
#Funcion para adquisicion de profundidad (depth) Kinect
def frame_depth():
        array,_ = sync_get_depth()
        #array = array.astype(uint8)
        return array


#Funcion que retorna imagen binaria donde lo verde es blanco
#y el resto es negro
def filtLAB_Verde(img):
        lab = cvtColor(img, COLOR_BGR2Lab)
        # pongo los valores verdes para hacer la mascara
        #verde_bajo = array([35, 110, 138]) -> im1 #verde_bajo = array([34, 97,
        143]) -> im1
        #verde_alto = array([102, 136, 174]) -> im1 #verde_alto = array([126, 118,
        174]) -> im1
        #verde_bajo = array([44, 111, 144]) -> im2 #verde_bajo = array([32, 108,
        131]) -> im2
        #verde_alto = array([101, 128, 172]) -> im2 #verde_alto = array([77, 128,
        153]) -> im2
        #verde_bajo = array([20, 126, 132]) -> im3 #verde_bajo = array([60, 86,
        125]) -> im3
        #verde_alto = array([80, 136, 154]) -> im3 #verde_alto = array([250, 124,
        179]) -> im3
```

```
        verde_bajo = array([20, 76, 132])
        verde_alto = array([240, 121, 215])

        mascara = inRange(lab, verde_bajo, verde_alto)

        er = ones((7,7),uint8) #matriz para erosion

        dil = array([[0,0,0,1,0,0,0],
        [0,1,1,1,1,1,0],
         [0,1,1,1,1,1,0],
         [1,1,1,1,1,1,1],
         [0,1,1,1,1,1,0],
        [0,1,1,1,1,1,0],
         [0,0,0,1,0,0,0]],uint8) #matriz para dilatacion

    mascara = erode(mascara,er,iterations = 1) #aplico erosion
    mascara = dilate(mascara,dil,iterations = 2) #aplico dilatacion
    return mascara

def filtLAB_Naranja(img):
        lab = cvtColor(img, COLOR_BGR2Lab)
        # pongo los valores de rango naranja para hacer la mascara
        #naranja_bajo = array([20, 146, 139]) -> im1
```

```
        #naranja_alto = array([76, 168, 166]) -> im1
        #naranja_bajo = array([35,147,144]) -> im2
        #naranja_alto = array([152,172,187]) -> im2
        #naranja_bajo = array([47, 136, 138]) -> im3
        #naranja_alto = array([228, 183, 194]) -> im3
    naranja_bajo = array([20, 136, 152])
        naranja_alto = array([235, 192, 198])


        mascara = inRange(lab, naranja_bajo, naranja_alto)


        er = ones((7,7),uint8) #matriz para erosion


        dil = array([[0,0,0,1,0,0,0],
        [0,1,1,1,1,1,0],
        [0,1,1,1,1,1,0],
        [1,1,1,1,1,1,1],
        [0,1,1,1,1,1,0],
        [0,1,1,1,1,1,0],
        [0,0,0,1,0,0,0]],uint8) #matriz para dilatacion


# matriz para erosion y dilatacion
mascara = erode(mascara,er,iterations = 1) #aplico erosion
mascara = dilate(mascara,dil,iterations = 2)#aplico dilatacion
```

```
return mascara


def buscar_pelotasVN():
        init=time()
        frame = frame_RGB() #leo frame
        depth = frame_depth() #leo profundidad depth
        depth = resize(depth,(0,0),fx=0.5, fy=0.5)
        showdepth = depth.astype('uint8')
        showdepth = cvtColor(showdepth, COLOR_GRAY2BGR)



        mascaraV = resize(frame, (0,0), fx=0.5, fy=0.5)
        mascaraN = mascaraV
        frame = mascaraV
        frame = medianBlur(frame,5)
        #imwrite('frame.jpg',frame)


        color=time()
        mascaraV = filtLAB_Verde(frame)
        mascaraN = filtLAB_Naranja(frame)
        tc=time()-color


        edge=time()
```

```
#Toma el valor absoluto -> convertScaleAbs( grad_y, abs_grad_y )
#explicacion Sobel_x + Sobel_y (addWeighted, suma grad_x y grad_y)
#https://github.com/opencv/opencv/
#blob/master/samples/cpp/tutorial_code/ImgTrans
#/Sobel_Demo.cpp

nuevoV = filters.sobel(mascaraV)
nuevoN = filters.sobel(mascaraN)
nuevoV = nuevoV.astype("int")
nuevoN = nuevoN.astype("int")
mascaraV[:,:] = nuevoV[:,:]*255
mascaraN[:,:] = nuevoN[:,:]*255

diler = array([[0,1,0],
        [1,1,1],
        [0,1,0]],uint8) #Matriz para dilatacion y erosion

mascaraV = dilate(mascaraV,diler,iterations = 1) #aplico dilatacion
mascaraN = dilate(mascaraN,diler,iterations = 1) #aplico dilatacion

te=time()-edge
```

```
#Hallo los circulos que esten en deteccion de bordes

circuloV = HoughCircles(mascaraV,HOUGH_GRADIENT, 1, 40, param1=60,
                    param2=24,minRadius=0,maxRadius=0)
circuloN = HoughCircles(mascaraN,HOUGH_GRADIENT, 1, 40, param1=60,
                    param2=24,minRadius=0,maxRadius=0)

#Para obtener la distancia depV y depN se utilizo la informacion de esta
#pagina: https://openkinect.org/wiki/Imaging_Information (Agosto 18)
#Esa regresion se le hicieron modificaciones para disminuir el error
#hallando una aproximacion de la forma 1/(Bx+C), donde x es el valor
#en bytes obtenido por el sensor

#Para la alineacion
cteX=9
cteY=9 #Valores alineacion RGB y Depth 320 x 240 (18/2)
#circle(rgb, (80-cteX,50+cteY),40,(0,0,255),5)

centimg = round(frame.shape[1]/2) #centro de la imagen donde son 0 grados horizontal
centVert= round(frame.shape[0]/2) #centro vertical 0 grados vertical

#Si encontro al menos un circulo
if circuloV is not None:
```

```
            circuloV = circuloV.astype("int")
            xV = circuloV[0,0,0]
            xVd=xV + cteX
            yV = circuloV[0,0,1] yVd=yV - cteY
            verde=True
    if xVd >= frame.shape[1]:
            xVd = 319
    if yVd >= frame.shape[0]:
            yVd = 239
    #Paso el pixel de coordenadas RGB a depth
    depV = 1/(depth[yVd,xVd]*(-0.0028642) + 3.15221) #para obtener dato es en
    coordenada (y,x)-#(480x640)
    depV = round(depV,4) #cuatro cifras decimales
    if depV < 0:
            depV=0
    else:
            verde = False
    if circuloN is not None:
            circuloN = circuloN.astype("int")
            xN = circuloN[0,0,0]
            xNd=xN + cteX
            yN = circuloN[0,0,1]
            yNd=yN - cteY
```

```
            naranja=True
            if xNd >= frame.shape[1]:
                xNd = 319
            if yNd >= frame.shape[0]:
                yNd = 239
#para obtener dato es en coordenada (y,x)->(480x640)
depN = 1/(depth[yNd,xNd]*(-0.0028642) + 3.15221)
depN = round(depN,4) #cuatro cifras decimales
if depN < 0:
            depN=0
else:
            naranja = False
if naranja or (verde and naranja):
  c1,c2=1,0
            bethaN = abs(centVert - yNd)*0.17916 #0.17916 son grados/Px en vertical
                (43grad/240)
            bethaN = (bethaN*pi)/180
            depN = depN*cos(bethaN) # centro valor vertical para ubicar la distancia en 0grad
            #Vertical
            alphaN = (xNd - centimg)*0.1781 #0.1781 son los grados por pixel (grad/px) 320
x 240
            alphaN = (alphaN*pi)/180 # en radianes
            xm = depN*sin(alphaN)
```

```
        ym = depN*cos(alphaN)
elif verde and (not naranja):
        c1,c2=0,1
        bethaV = abs(centVert - yVd)*0.17916 #0.17916 son drad/Px en vertical
(43grad/240)
        bethaV = (bethaV*pi)/180
        depV = depV*cos(bethaV) # centro valor vertical para ubicar la distancia en 0grad
        #Vertical
        alphaV = (xVd - centimg)*0.1781 #0.1781 son los grados por pixel (grad/px)
        alphaV = (alphaV*pi)/180 # en radianes
        xm = depV*sin(alphaV)
        ym = depV*cos(alphaV)
else:
        c1,c2=0,0
        xm,ym=0,0
        t=time()-init
print('FIN',t,'EDGES',te,'COLOR',tc)
# print('c1',c1,'c2',c2,'xm',xm,'ym',ym)
#imshow('NARANJA',mascaraN)
#waitKey(1)
#imshow('VERDE',mascaraV)
#waitKey(1)
#imshow('FRAME',frame)
```

```
#waitKey(1)

return(c1, c2, xm, ym)
```