EVALUACIÓN DEL ALGORITMO DE PUNTOS Y LÍNEAS EN UN SISTEMA ORB-SLAM DE COMPUTACIÓN PARALELA

DAVID ALEXÁNDER LLANTÉN CASTRILLÓN

Director

Msc. Elena Muñoz España

Codirector

Msc. Juan Fernando Flórez Marulanda

UNIVERSIDAD DEL CAUCA

Facultad de Ingeniería Electrónica y Telecomunicaciones

Programa de Ingeniería en Automática Industrial

2022

CONTENIDO

1. INTRODUCCIÓN

La solución al problema SLAM (por su siglas en inglés, *Simultaneus Localization And Mapping*); planteado por primera vez en la Conferencia de Robótica y Automática de IEE en 1986, en San Francisco [1]; se determina como la capacidad de ubicar un vehículo autónomo en un entorno y ubicación desconocido y que él sea capaz de, a partir de las imágenes que va observando, construir un mapa y simultáneamente navegar sobre él; eliminando así la necesidad de conocer a priori las infraestructuras artificiales del ambiente [2].

Algunos autores [3] consideran resuelto el problema del SLAM 2D para dimensiones reducidas, pero no para entornos de grandes dimensiones o cuando se trabaja con sistemas que requieren información visual 3D [4]. Existen dos métodos que buscan dar solución al problema del SLAM 3D: LiDAR SLAM [5], [6] que utiliza sensores de láser y Visual-SLAM [7] que extrae características para el mapeo del entorno a partir de datos que entrega una o más cámaras [8]. ORB-SLAM [9], [10] es uno de los métodos más utilizados dentro del método Visual-SLAM. En él, el "punto" es tipo de marca sobre la característica más importante, y aunque este procedimiento presenta un buen rendimiento, aún es propenso a fallar en entornos planos de texturas bajas donde es complejo encontrar un conjunto de características puntuales [11]. Por ese motivo, en [11] se plantea añadir una extensión de líneas a las características puntuales, pues los segmentos de línea entregan una mejor información de estructuras geométricas. Sin embargo, procesar líneas y puntos requiere una mayor carga computacional, que hace difícil alcanzar el objetivo SLAM en tiempo real que está especificado en por lo menos 10fps [8].

Los sistemas de computación paralela y los aceleradores se están abriendo paso poco a poco en el mundo de la *informática de alto rendimiento* (HPC, por sus siglas en inglés, *High Performance Computing)* por lo que las tarjetas gráficas han ido evolucionando hacia la programabilidad general y han ido ganando terreno en la HPC; más aún con la introducción de la arquitectura *Compute Unified Device Architecture* (CUDA) por parte de NVIDIA, que es un modelo de programación que proporciona las interfaces adecuadas para explotar mejor la potencia de los sistemas de computación paralela [12], [13].

El presente trabajo integra el método de ORB-SLAM2 y su extensión de líneas con una tarjeta de procesamiento paralelo de NVIDIA, GeForce GTX 1650 Ti. Con el fin de evaluar su precisión y su comportamiento en tiempo real.

Para lograr evaluar el desempeño de ejecución en tiempo real del algoritmo de puntos y líneas para el problema de SLAM para una plataforma de computación paralela, inicialmente fue necesario construir el sistema que integra el método de puntos y líneas

en dicha plataforma; luego se especificó el protocolo de pruebas y los indicadores de desempeño de la implementación en la plataforma, y finalmente, se especifica en qué condiciones el sistema de puntos y líneas logra un mejor desempeño en tiempo real.

Este documento presenta representa el resultado de un trabajo de estudio y evaluación que buscó tener como resultado un análisis profundo de un algoritmo de puntos y líneas en una tarjeta de computación paralela, preferiblemente, la tarjeta NVIDIA Jetson Nano.

Inicialmente el documento presenta una investigación sobre el funcionamiento del algoritmo madre de estos sistemas, el denominado ORB-SLAM2, el cual a su vez es una versión mejorada de código abierto de ORB-SLAM, este documento expone el método en el que el algoritmo procesa las imágenes y encuentra las marcas sobre la imagen con las que realiza todo su funcionamiento. Posteriormente, en esta misma sección, el documento se enfoca en la forma del procesamiento de la imagen para lograr una adecuada extracción de línea para posteriormente hacer la validación y la integración de las líneas con el sistema de puntos ORB.

En la siguiente sección del documento, se presenta la forma de instalación del algoritmo PL-SLAM dentro de una tarjeta de computación paralela con sistema operativo Ubuntu 18.04. En ella se definen todas las dependencias y librerías necesarias para que el algoritmo funcione correctamente, además se muestra la forma correcta de instalar el sistema incluyendo las que se encuentran obsoletas para el sistema operativo mencionado.

En el capítulo 4, se definen las secuencias con las que el algoritmo será ejecutado con el fin de hacer una evaluación comparable con las que se han encontrado en la literatura; dentro de él se muestran las secuencias y se hace una breve descripción de cada una de ellas.

Finalmente, en el capítulo 5, se hace un análisis de los resultados obtenidos de la ejecución del algoritmo con las trayectorias previamente mencionadas. Inicialmente se muestran tablas que presentan el rendimiento del sistema en todas las ejecuciones que se hicieron para cada trayectoria. Después se hace una comparación de la precisión del algoritmo con otros y de la velocidad del mismo algoritmo probado en otro sistema.

2. LOCALIZACIÓN Y MAPEO SIMULTÁNEO (SLAM)

Este capítulo da una descripción detallada del algoritmo ORB-SLAM2, mencionando también algunos componentes del algoritmo del cual se basa el mencionado, ORB-SLAM; posteriormente, explica la integración de puntos en el sistema SLAM y finalmente, detalla las integraciones de puntos y líneas en los sistemas de localización y mapeo simultáneo.

2.1. ORB-SLAM2

El problema de localización simultánea y construcción de mapas (SLAM) plantea si es posible que un vehículo autónomo, con un modelo cinemático conocido, comience en una ubicación desconocida y se desplace en un entorno igualmente desconocido que contiene una población de características puntuales y luego construya, a partir de esos puntos de referencia, de forma incremental un mapa de este entorno mientras utiliza simultáneamente este mapa para calcular la ubicación absoluta del vehículo [14], [15].

Dentro del campo de la robótica móvil al problema de SLAM se le han planteado soluciones que se basan en diferentes métodos. Los métodos que obtienen información a partir de datos que entregan cámaras monoculares o de profundidad son conocidos como Visual-SLAM [7], [8], de los cuales ORB-SLAM es uno de los más utilizados, pues su sistema basado en el "punto" presenta un buen rendimiento, aunque constantemente está siendo trabajado para mejorar su precisión y su velocidad en tiempo real.

ORB-SLAM funciona con tres procesos: seguimiento, mapeo local y cierre de bucle [16]. El seguimiento comienza con la inicialización del mapa y se encarga de ubicar el robot, basándose en datos de profundidad que obtiene de los primeros fotogramas, y finaliza con un fotograma clave que ubique de manera precisa la posición del robot. El mapeo local se encarga de colocar nuevos puntos del mapa a medida que las observaciones van aumentando, triangula puntos con varios fotogramas claves vecinos siempre que se hayan emparejado algunas características ORB. Finalmente, el cierre de bucle es la contribución más interesante de ORB-SLAM, cuando el robot regresa al punto de partida debe conectar el último fotograma clave con los fotogramas claves iniciales. Con los tres procesos funcionando adecuadamente se logra que la trayectoria estimada del robot sea más precisa [17].

Con los puntos ORB toma forma la estructura de la reconstrucción 3D de la escena; la triangulación de cada punto desde distintos lugares de la misma escena facilita la depuración de fotogramas clave, aunque un solo punto ORB esté presente en varios fotogramas[10]. A medida que se van tomando fotogramas, la supervivencia de los

puntos ORB también se evalúa; con esto se logra una mejor robustez y se genera un mapa compacto que sólo crece si cambia el contenido de la escena [18]. Al encontrarse el mapeo en el área covisible, el sistema navega por entornos amplios sin aumentar los tiempos de computación porque utiliza las mismas funciones para ejecutar todas las tareas SLAM haciendo que el sistema sea más eficiente y rápido [19].

En 2017, Raúl Mur-Artal y Juan Tardós propusieron ORB-SLAM2[20]. Éste trabajo presenta algunas contribuciones como ser el primer sistema de código abierto para cámaras monoculares, estéreo y RGB-D; y, según los resultados, el mejoramiento del sistema ganando en precisión basándose en el punto más cercano iterativo (ICP) [21].

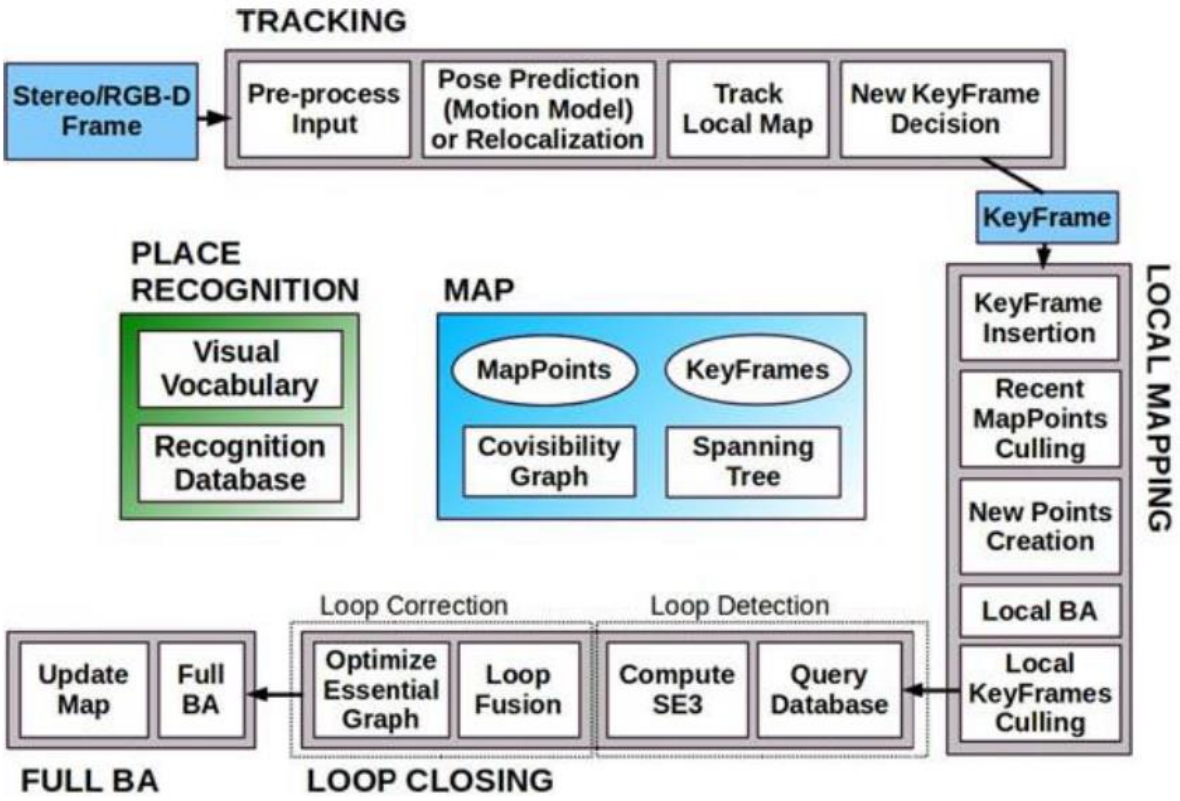


Figura 1. Hilos y módulos del sistema ORB-SLAM2
Imagen tomada del artículo "ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras"

La *Figura 1* muestra los módulos que componen los 3 hilos principales de ORB-SLAM2. Módulos que se explicarán a continuación.

- **SEGUIMIENTO**
  El tracking se encarga de localizar la cámara con cada fotograma y decidir cuándo insertar un nuevo fotograma clave. Primero se realiza una función inicial que encuentra una coincidencia con el cuadro anterior y la optimiza usando un ajuste de paquete. Una vez que hay una estimación inicial de la pose de la cámara y las coincidencias de características, se recupera un mapa visible local

utilizando el gráfico de covisibilidad de fotogramas clave que mantiene el sistema [20].

Una de las principales ventajas de utilizar cámaras RGB-D es que en el primer fotograma se puede disponer de información de profundidad y de rotación; esto porque los puntos clave se clasifican entre cercanos, los puntos son los puntos que su profundidad es inferior a 40 veces su línea de base estéreo/RGB-D [22], y ellos pueden triangularse a partir de un fotograma; mientras que los puntos lejanos, son los que se encuentran más alejados de la distancia mencionada y son ideales para brindar información sobre rotación. Ellos son triangulados cuando se apoyan en múltiples vistas. Entonces al iniciar el sistema se crea un fotograma clave con el primer fotograma, se fija su pose de origen e inicia el mapa a partir de todos los puntos clave estéreo, con lo que se solucionan también los problemas de relocalización [20].

Seguimiento del mapa local

Una vez que se tiene una estimación de la pose de la cámara y un conjunto inicial de coincidencias de características, se puede proyectar el mapa en el marco y buscar más correspondencias de puntos del mapa. Esto se realiza proyectando un solo mapa local que contiene un conjunto de fotogramas clave y un conjunto de fotogramas clave vecinos. Para encontrar un punto de mapa visto en el conjunto de fotogramas clave; primero, se calcula la proyección del punto de mapa en el cuadro actual, se descarta si se sale de los límites de la imagen; segundo, se obtiene el ángulo entre el rayo de visión actual y la dirección de visualización media del punto del mapa n, se descarta si ese ángulo es menor que el coseno de 60; tercero, se determina la distancia desde el punto de mapa hasta el centro de la cámara, se descarta si el punto está por fuera de la región de invarianza; cuarto, se halla la escala en el marco; y quinto, se compara el descriptor representativo del punto del mapa con las características ORB aún no coincidentes, asociándose el punto del mapa con la mejor coincidencia [18].

Decisión de nuevo fotograma clave

Para decidir si se introduce un nuevo fotograma clave, se deben cumplir las siguientes 4 condiciones; primero, deben haber pasado más de 20 fotogramas desde la última relocalilzación global; segundo, la asignación local está inactiva o han pasado más de 20 fotogramas desde la última inserción de un fotograma clave; tercero, el marco actual encuentra por lo menos 50 puntos y; cuarto, el cuadro actual rastrea menos del 90% de los puntos que rastrea el fotograma de referencia [18].

- **MAPEO LOCAL**
  Se describen los pasos realizados para ubicar cada fotograma clave a medida que las observaciones van aumentando y el mapa va creciendo.

  Inserción de fotogramas clave

  Se actualiza el gráfico de covisibilidad actualizando los bordes resultantes de los puntos de mapa compartidos con otros fotogramas clave, se refresca el enlace del árbol de expansión con el fotograma clave y la mayoría de sus puntos en común para finalmente calcularle una representación de bolsas de palabras. Con eso se facilitará la triangulación de nuevos puntos entregados por nuevos fotogramas clave [18].

  Eliminación de puntos de mapa recientes

  Para que los puntos característicos sobrevivan en el mapa, se debe garantizar que los puntos de mapa se puedan rastrear y que no estén triangulados incorrectamente. Para ello deben cumplir dos condiciones; primero, el seguimiento debe encontrar el punto en más del 25% de los cuadros en los que se cree que el punto debe ser visible; y segundo, si ha pasado más de un fotograma desde su creación, debe estar presente en por lo menos 3 fotogramas clave. Si pasan esta prueba, sólo se pueden eliminar si en algún momento se detecta en menos de 3 fotogramas clave, lo que puede llegar a suceder cuando se eliminan fotogramas clave redundantes [18].

  Creación de nuevos puntos de mapa

  Los nuevos puntos de mapa se encuentran triangulando entre fotogramas clave conectados las características puntuales ORB. Se buscan coincidencias con otros puntos no coincidentes en otros fotogramas claves, se triangulan los pares ORB y se comprueba bajo criterios de profundidad positiva, paralelaje, error de reproyección y consistencia de escala [18].

  Ajuste de paquete local

  El ajuste de paquete local se encarga de optimizar el fotograma clave actual apoyándose en todos los fotogramas clave conectados a él y todos los puntos de mapa vistos por esos fotogramas clave. Permanecen fijos los fotogramas clave que ven los puntos pero no están conectados al fotograma actual, aunque se incluyan en la optimización. Las observaciones con valores atípicos se descartan a la mitad y al final de la optimización [18].

Selección local de fotogramas clave

El mapeo local intenta detectar fotogramas clave y eliminarlos. Esto beneficia al sistema dado que la complejidad del ajuste de paquete local aumenta a medida que crece la cantidad de fotogramas clave; también se garantiza que la cantidad de fotogramas clave no crecerá sin límites, solo en caso que cambie el contenido de la escena. Se descartan fotogramas clave en los que el 90% de los puntos se hayan visto en mínimo otros tres fotogramas clave [18].

- **CIERRE DE BUCLE Y AJUSTE DE PAQUETE COMPLETO**

  El proceso de cierre de bucle se realiza en dos pasos; primero, se detecta y se valida un bucle; y segundo, se corrige el bucle. La información de profundidad hace que la escala sea observable, entonces hace que el cierre de bucle se base en transformaciones de cuerpo rígido. El sistema puede crear mapas y detectar bucles al mismo tiempo, pero si se detecta un nuevo bucle mientras se están optimizando fotogramas clave, se aborta esa optimización y se procede a cerrar el bucle [20].

2.2. SLAM BASADO EN PUNTOS.

Las aplicaciones de visión artificial se basan fundamentalmente en la detección de puntos característicos, incluyendo la computación de flujo óptico [23]–[25], la estructura de movimiento [26], [27] y la observación de cámaras estéreo para la reconstrucción 3D de escenas [28].

ORB-SLAM, es un sistema monocular de mapeo y localización simultáneo que utiliza características de imágenes binarias ORB, que a su vez se fundamenta en encontrar puntos clave en tiempo real basándose en el detector de esquinas FAST [9], [29],[30]. Los ORB tienen un buen rendimiento frente al problema de invariancia de rotación y escala, eso hace que presente buenos resultados independiente del punto de observación de la imagen [18].

Las esquinas son puntos de la imagen que muestran fuertes cambios de intensidad bidimensional y, por tanto, se distinguen bien de los puntos vecinos. Los detectores de esquinas se han utilizado ampliamente como detectores de puntos característicos porque las esquinas corresponden a ubicaciones de la imagen con un alto contenido de información, y pueden emparejarse entre imágenes de forma fiable. Algunas aplicaciones toman estas características como entrada para realizar tareas de visión artificial de alto nivel. Una óptima detección de esquinas, debe satisfacer esencial mente 3 criterios; consistencia, precisión y velocidad; la consistencia es fundamental ya que garantiza que los puntos detectados son insensibles al ruido y que están ubicados en la misma posición en todas las imágenes de la misma escena, la precisión se encarga de que el punto detectado se ubique lo más cercano posible a su posición real y, la

velocidad hace que todos estos requerimientos se cumplan en tiempo real. Si no cumple cualquiera de estos 3 criterios mencionados anteriormente, el algoritmo sería inútil [30].

FAST utiliza una función de respuesta de córner (CRF) que da un valor numérico dependiendo de la fuerza de esquina que tenga cada pixel sobre la intensidad de la imagen en la vecindad local. En él, las esquinas son máximos locales de la CRF. La función de respuesta de córner utiliza la misma notación que el algoritmo SUSAN; él considera un pixel y a su alrededor una ventana circular de píxeles que tienen un brillo similar al pixel del núcleo, eso se denomina la zona USAN [31]. Las esquinas se calculan basándose en área y el centro de gravedad de USAN.

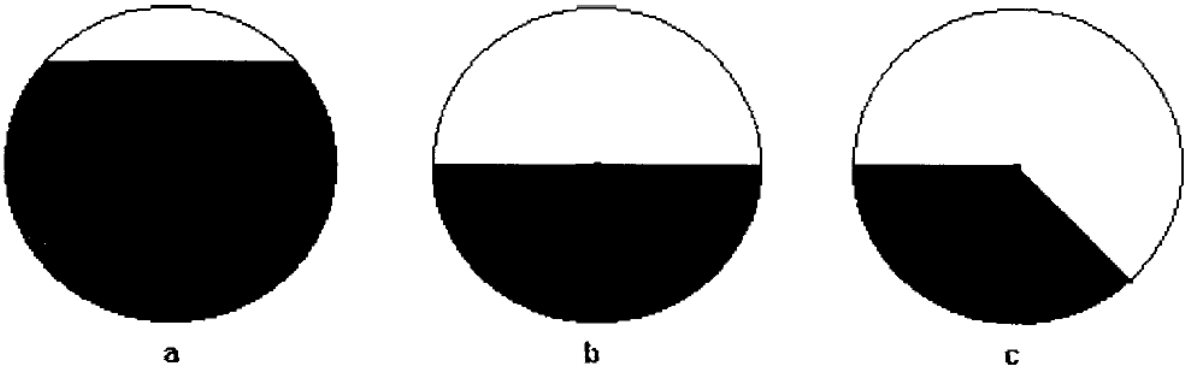


Figura 2. Casos representativos de USAN.
Imagen tomada del artículo *"Fast Corner Detection"*

Como se puede ver en la *Figura 2*, hay tres casos representativos de USAN. En el primero, el núcleo se encuentra dentro de un área uniforme; en el segundo, el núcleo es un punto en un borde y; en el tercero, el núcleo es un punto de esquina. La función de respuesta de córner debe diferenciar el punto de esquina con un punto de borde o dentro de un área uniforme [30].

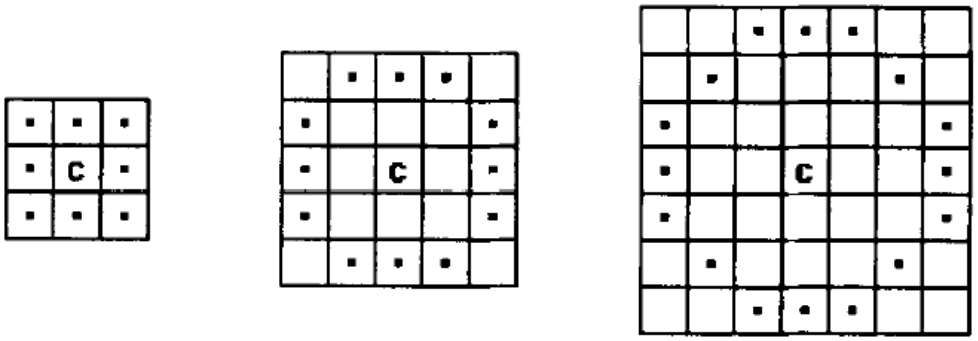


Figura 3. Círculos digitales con diámetro de 3 pixeles, 5 pixeles y 7 pixeles.
Imagen tomada del artículo *"Fast Corner Detection"*

La *Figura 3,* muestra 3 círculos digitales con diámetros diferentes que tienen el mismo centro *"c"* como candidato a esquina. Considerar una línea arbitraria l que cruce al núcleo y dos puntos opuestos de ventana circular *P* y *P'*.

$$R_N = \min\left(\left(f_p - f_N\right)^2 + \left(f_{p'} - f_N\right)^2\right)$$

N es el núcleo, y las funciones *f* se refieren a la intensidad en los puntos *P, P'* y *N.* Retomando los casos de la imagen USAN, en el primer caso, *P* y *P'* pertenecen a USAN, por lo que la respuesta es cercana a 0; en el segundo caso, dentro de todas las líneas tangentes que crucen a todos los puntos pasando por el núcleo, hay una recta que cruza dos puntos opuestos *P* y *P'* que pertenecen a USAN, por lo que la respuesta para esa línea vuelve a ser cercana a 0; finalmente, cuando el núcleo central es una esquina, no hay una recta que cruce dos puntos *P* y *P'* que pertenezcan a USAN, por lo que la respuesta del CRF será mayor que 0 [30].

El algoritmo SLAM utiliza ORB debido a que las esquinas FAST son rápidas de calcular y combinar, además de que son características binarias que tienen una buena invariancia al punto de observación; lo que mejora notablemente la precisión del ajuste de paquete. A pesar de que la implementación de ORB en el algoritmo entrega un buen desempeño, las exigencias del sistema hace que las técnicas no se limiten únicamente a esas funciones [18].

Como bien se sabe, los puntos de mapa ORB son puntos que contienen información adecuada para que el sistema SLAM sea capaz de trabajar con ellos. Básicamente, cada punto de mapa ORB debe almacenar por lo menos 4 entes. Primero; su ubicación en el sistema de coordenadas 3D; segundo, la dirección de visualización, que es el vector medio que une el centro óptico de la cámara con el punto de los fotogramas clave que lo observan; tercero, que el descriptor ORB asociado presente una distancia de haming mínima respecto a los demás descriptores asociados en los fotogramas clave que observan el punto; y cuarto, el máximo y el mínimo de distancias a las que se pueda observar el punto [18].

Para que se conserven esos puntos del mapa, que estar presente mínimo durante los tres primeros fotogramas clave después de su creación; eso garantiza que se puedan rastrear y que su triangulación no sea incorrecta. Además de eso, un punto de mapa debe cumplir con que el seguimiento lo encuentre en más del 25% de los cuadros en los que se pronostica que sea visible. Los nuevos puntos de mapa se crean triangulando las características ORB a partir de fotogramas clave conectados; cada punto ORB sin emparejamiento debe encontrar una coincidencia con otro punto sin emparejamiento en otro fotograma clave; así un punto de mapa podría observarse a partir de dos fotogramas clave a pesar de que podría coincidir con otros, por esa razón se proyecta en el resto de fotogramas clave conectados y se buscan las mejores correspondencias [18].

2.2. SLAM BASADO EN PUNTOS Y LÍNEAS

Como ya se mencionó, los sistemas basados en ORB-SLAM constan de tres hilos; el seguimiento que se encarga de localizar la pose de la cámara en cada fotograma, el mapeo local que es el que mantiene los mapas que incluyen los fotogramas clave, los puntos ORB y las líneas de mapa; y finalmente, el cierre de bucle que se encarga de buscar constantemente el bucle y corregirlo[11].

Dado que las coincidencias de líneas presentan una menor precisión a la hora de definir la posición inicial del sistema en el mapa, ella se define únicamente con los descriptores de puntos. Después de establecer la posición inicial, se inicia la búsqueda de coincidencias de puntos y líneas, con lo que la posición del sistema se optimiza. Los segmentos de línea en la etapa de mapeo local se obtienen con el detector EDLines [11], [32].

CARACTERÍSTICAS DE LÍNEA EN VISUAL SLAM.

ED Drawing (ED) es un algoritmo de detección de líneas de borde que obtiene de la imagen un conjunto de segmentos de borde, que son cadenas limpias y contiguas de píxeles que tienen una mayor probabilidad de encontrarse en un borde, se denominan como puntos de anclaje, y dibuja bordes de un pixel de ancho entre ellos con una conectividad real. Eso lo diferencia de otros algoritmos, pues la salida de otros detectores de líneas son entidades independientes y discontinuas que requieren de procesamientos adicionales para encontrar los límites de los objetos; cuestiones que pueden ser imposibles o dar paso a imprecisiones [32], [33].

Llevando la imagen a escala de grises, ED se ejecuta en cuatro pasos consecutivos; primero, la imagen pasa por un filtro gaussiano para suprimir el ruido y suavizar la imagen; segundo, se determina la magnitud y la dirección del gradiente de cada pixel de la imagen resultante del paso 1; tercero, se calculan los puntos de anclaje de los bordes; y cuarto, se vinculan los puntos de anclaje de los bordes mediante un trazado inteligente [32], [33].

1. ***Suavizado de imágenes y reducción del ruido.***

Figura 4. Imagen original (izquierda) e imagen suavizada con kernel gaussiano(derecha).

Para suavizar y reducir el ruido en la imagen difuminando cada pixel con los pixeles vecinos, simplemente se aplica a la imagen un kernel gaussiano estándar de 5x5 con σ = 1 [33]. Se obtiene un resultado como el de la *Figura 4.*

2. ***Determinación de las áreas y las direcciones de los bordes.***

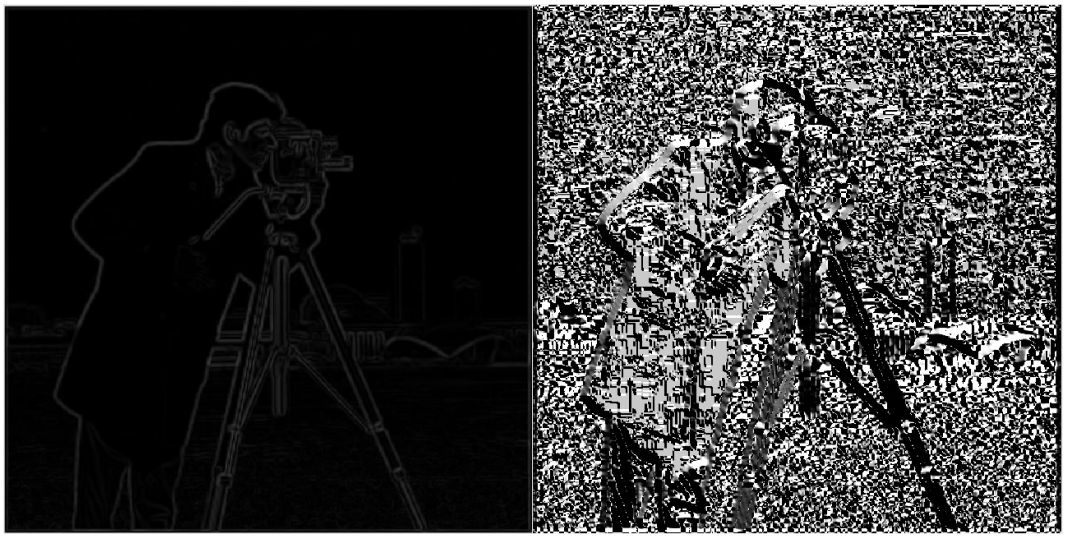


Figura 5. Magnitud (izquierda) y dirección (derecha)del gradiente.

Se definen como área de borde las zonas en las que los pixeles tienen una magnitud de degradado mayor a cierto umbral; para ello se aplica una función

de derivación o un filtro pasa alto a cada pixel de la imagen suavizada; esta tarea la cumple bien el operador de Sobel[34], pero también sirven bien otros operadores de gradiente como Prewitt y Scharr [33], [35]. El operador Sobel entrega gradientes verticales (Gx) y horizontales (Gy), la magnitud del gradiente se obtiene sumando sus valores absolutos ($G = |Gx| + |Gy|$). A esa magnitud del gradiente G se le aplica un umbral que determine que todos los pixeles con G mayores que el umbral pertenecen a un área de borde[35]. Obteniendo una imagen similar a la de la izquierda de la *Figura 5.*

Con la misma información que entrega el operador Sobel, se determinan las direcciones de los bordes. En este caso, se analiza si el borde es vertical o horizontal y se hace de la siguiente forma: Si $Gx > Gy$, en la zona de ese pixel está pasando un borde vertical; y si el borde que atraviesa el pixel es horizontal, entonces, $Gy > Gx$ [33], [35]. Resultando una imagen similar a la de la derecha de la *Figura 5.*

3. Extracción de puntos de anclaje.

Figura 6. Extracción de los puntos de anclaje partiendo de la magnitud del gradiente.

La *Figura 6* muestra el resultado de la extracción de los puntos de anclaje, parte fundamental para preparar el proceso de dibujo de los bordes; se hace un barrido de la imagen escaneándola horizontal y verticalmente; donde se pasan sólo los máximos y son determinados como puntos de anclaje.

Es importante mencionar que el usuario puede escoger qué cantidad de detalle desea; es decir, si sólo desea la detección de los bordes principales o aristas

largas entre puntos de anclaje, puede definir un 'ratio de detalle' grande. 'Ratio de detalle' que lo que define es el intervalo de exploración dentro de la imagen; entre más alto menos detalle y entre más bajo más detalle [33], [35].

4. Vinculación de puntos de anclaje mediante dibujo de borde.

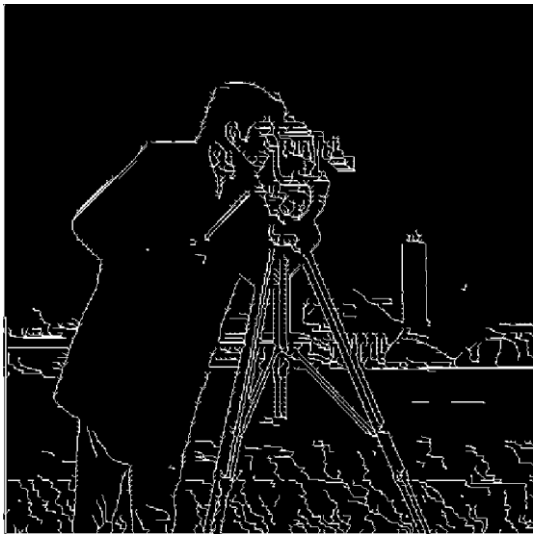


Figura 7. Dibujo de borde mediante los puntos de anclaje.

Finalmente, para obtener un resultado como el de la *Figura 7*. En este paso lo que se quiere es dibujar los bordes reales iniciando desde un punto de anclaje hasta llegar a otro. Para lograrlo, se utiliza el mapa de gradientes y de direcciones obtenidas en el segundo paso. Para hacer el enrutamiento, se observan los pixeles vecinos en la dirección del enlace y se escoge para el siguiente salto el que tiene mayor longitud de gradiente, esto se hace de esta forma hasta que se llega al límite de la imagen de magnitud de gradiente o hasta que se encuentra un borde previamente calculado. Con esto se garantiza que los bordes sean lisos, contiguos y de un pixel de ancho [33], [35].

Figura 8. Enrutamiento inteligente horizontal (izquierda), enrutamiento inteligente vertical (derecha).

Imagen tomada de *"Real-time Edge Segment Detection with Edge Drawing Algorithm"*

Se considera que el sistema se encuentra posicionado en un pixel (i,j) que es un punto de anclaje y se está moviendo hacia arriba porque la arista que pasa por ese pixel es vertical, entonces el siguiente paso es observar los pixeles (i-1,j-1), (i-1,j) e (i-1, j+1) y se da el salto al que tenga el valor máximo de gradiente al norte. Del mismo modo, el sistema puede recorrerse hacia abajo en una arista vertical; en ese caso, posicionado en el pixel (i,j) el sistema va al pixel que tenga el mayor valor de gradiente al sur entre los pixeles (i+1,j-1), (i+1,j) e (i+1,j+1). En caso de que el borde sea horizontal, se puede trazar un camino a la derecha o a la izquierda partiendo desde el punto de anclaje (i,j); si el recorrido se hace hacia la izquierda, el sistema iría al pixel con el valor máximo de gradiente entre los pixeles (i-1,j-1), (i,j-1) e (i+1,j-1); si por el contrario el recorrido se hace hacia la derecha, el sistema pasaría al valor máximo de gradiente entre los pixeles (i-1,j+1), (i,j+1) e (i+1,j+1). Este procedimiento de enlace hace que el dibujo pase por encima de todas las aristas reales conectando todos los puntos de anclaje. Este procedimiento se conoce como enrutamiento inteligente y se explica gráficamente en la *Figura 8.*

Ahora, es necesario generar una coincidencia de segmentos de línea 3D-2D, lo que se realiza en tres pasos: proyectar los segmentos de línea en la imagen, determinar coincidencias candidatas e identificar la mejor coincidencia. En el sistema, la línea del mapa se representa como $L = (L_P, L_F)$, donde L_P es la línea 3D que contiene el segmento de línea que se parametriza con la coordenada de línea de Plucker y la representación ortonormal [36]. L_F es un conjunto de fotogramas clave en los que se puede observar la línea [11].

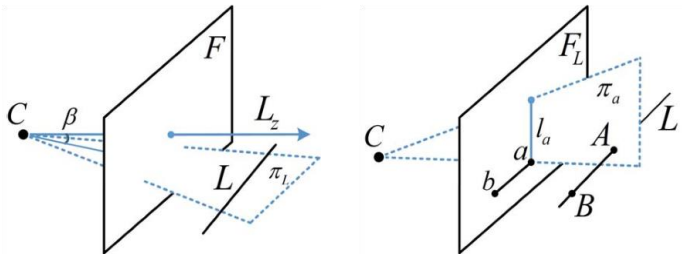


Figura 9. Ángulo de observación del segmento de línea 3D L (izquierda). Y el cálculo de los puntos finales A, B de L (derecha).
Imagen tomada de *"Real-Time Monocular Visual SLAM by Combining Points and Lines"*

La *Figura 9* ilustra la definición de las variables que se mencionan en el proceso de la coincidencia de los segmentos de línea. Primero se encuentra el fotograma clave F_L que tiene el ángulo de observación más cercano de L y se compara con el fotograma actual F. Entonces, el punto final A de L está determinado por la intersección de la línea y el plano π_a que contiene el centro de la cámara A y la línea vertical l_a de $ab. ab$ es el

segmento de línea coincidente de L en F_L. El otro punto final B se determina con el mismo método. Por último, los puntos finales A y B se proyectan en F y se puede obtener el segmento de línea de proyección l_L de L [11].

Los segmentos de línea en el marco actual se dividen en 36 conjuntos de acuerdo con sus direcciones. Y las entidades de puntos se procesan con el mismo método en ORB-SLAM. Se seleccionan los grupos de segmentos de línea en los que las líneas tienen una dirección similar a l_L. Todos los segmentos de línea de estos grupos se agregan al conjunto de líneas de coincidencia de candidatos S_L de L [11].

2.3. INTEGRACIÓN DEL SISTEMA DE PUNTOS Y LÍNEAS

El procesamiento paralelo es un método importante para crear SLAM visual en tiempo real. Este sistema de SLAM visual posteriores adopta un algoritmo de procesamiento paralelo que, como ya se vio, realiza tareas de seguimiento, mapeo y cierre de bucles [37]. Aquí se complementa el marco de procesamiento paralelo normal con un procesamiento paralelo de nivel de característica; lo que acelera aún más el sistema SLAM [11].

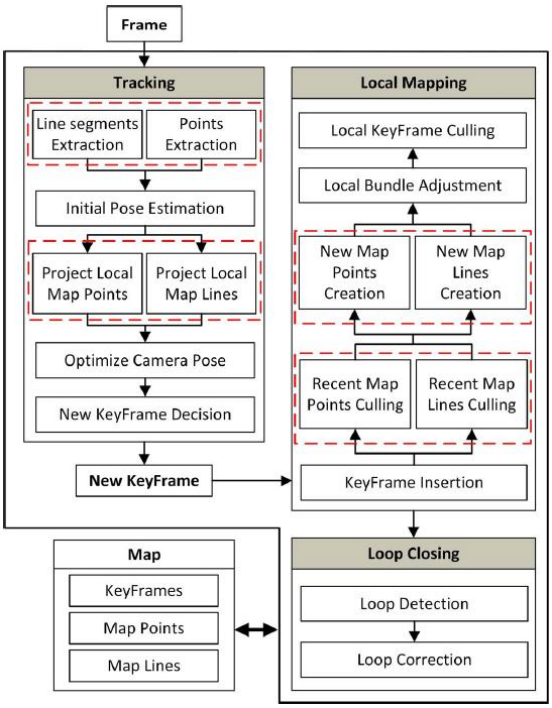
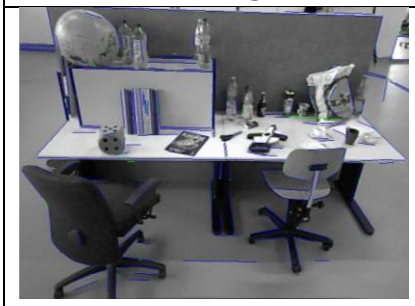


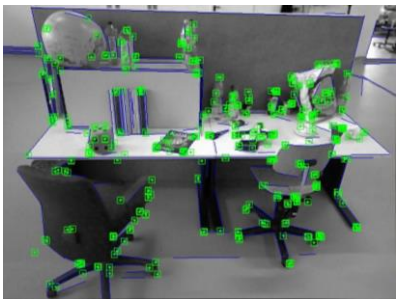
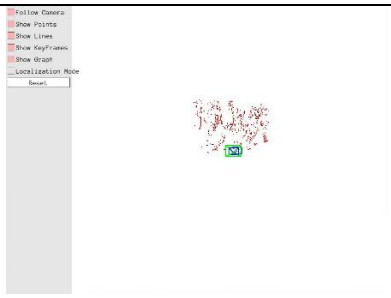
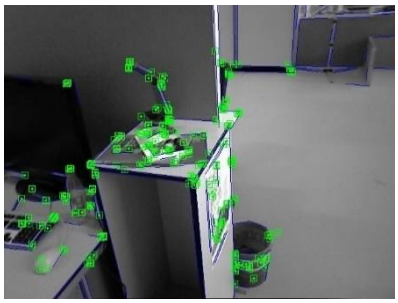
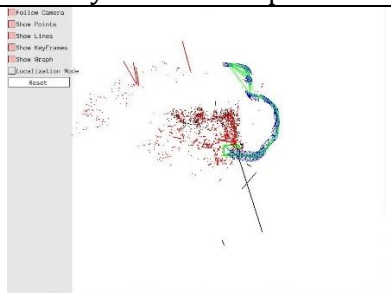
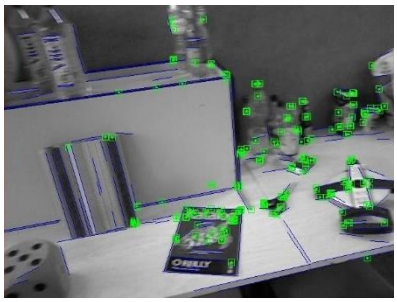
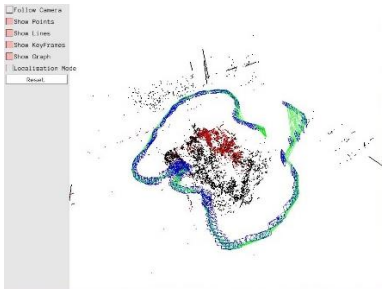
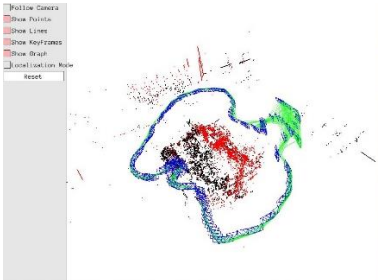
Figura 10. Descripción general del sistema SLAM.
Imagen tomada de *"Real-Time Monocular Visual SLAM by Combining Points and Lines"*

En la *figura 10* los módulos de procesamiento paralelo de nivel de característica se presentan con cuadros de puntos rojos; en el hilo de seguimiento, los segmentos de línea y los puntos ORB se extraen en dos hilos paralelos. Esto se hace así porque además de que para el primer fotograma clave el sistema utiliza únicamente la extracción de puntos, también se debe tener en cuenta la diferencia de velocidad de entre el detector de línea y el detector de punto. Después, las proyecciones de punto y de línea también se realizan en dos subprocesos; y en el hilo de mapeo local, las líneas y los puntos de mapa se procesan por separado en la selección de características para facilitar la selección de un nuevo fotograma clave.

De la misma forma como se hace con las características puntuales, todas las líneas del mapa local se proyectan en la imagen para encontrar coincidencias. Luego, si la imagen observada contiene suficiente información sobre su entorno se enmarca como un fotograma clave y sus líneas se triangulan para ser agregadas al mapa. Aquí también se descartan las líneas observadas en menos de tres fotogramas clave después de su primera observación o cuando la línea no se observa en por lo menos el 25% de los fotogramas desde los que se esperaba que se observara. Debido a que el costo computacional aumenta con el tratamiento de líneas, el cierre de bucle sólo se realiza con la detección de características puntuales[37].

- **SEGUIMIENTO DE LA TRAYECTORIA *'freiburg3_long_office_household'* PARA EVIDENCIAR LAS ETAPAS DEL ALGORITMO DE PUNTOS Y LÍNEAS.**

DETECCIÓN DE PUNTOS Y LÍNEAS	TRACKING	DESCRIPCIÓN
 **Figura 11.** Primera observación de líneas y puntos.	 **Figura 12.** Búsqueda de la pose inicial del sistema.	La *figura 11* y la *figura 12* muestran que el sistema está inicializándose. La primera inserción de un fotograma clave se da en el momento en el que el sistema logre determinar su posición inicial.

 **Figura 13.** Extracción de los primeros puntos y líneas.	 **Figura 14.** Almacenamiento de los primeros fotogramas clave y de los primeros puntos y líneas de mapa.	Una vez se determina la posición inicial del sistema, se insertan los primeros fotogramas clave.
 **Figura 15.** Extracción de nuevos puntos y líneas de la escena.	 **Figura 16.** Evidencia del seguimiento de la trayectoria y del recorrido de la cámara en el espacio.	Inicia la reconstrucción del mapa y se almacenan las primeras posiciones de la cámara en el espacio.
 **Figura 17.** Acercamiento del sistema y reconocimiento de puntos ya observados.	 **Figura 18.** Construcción de la trayectoria, observación de nuevos puntos de mapa y reconocimiento de puntos ya observados.	Se continúa con la reconstrucción del mapa y con el seguimiento de la posición de la cámara en el espacio. En azul se pueden observar los fotogramas clave, en verde el seguimiento del recorrido del sistema y en negro algunas proyecciones de los puntos y de las líneas observadas por el sistema.
 **Figura 19.** Regreso del sistema a un punto anterior del recorrido.	 **Figura 20.** Cierre de bucle en el seguimiento de la trayectoria.	Se cierra la trayectoria y se evidencia que el sistema regresa a un punto inicial, lo que se define como cierre de bucle.

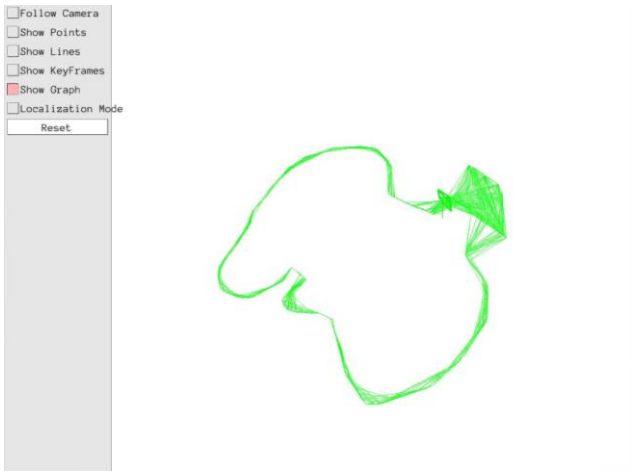
 **Figura 21.** Trayectoria final del sistema para la secuencia de imágenes.	Seguimiento del recorrido del sistema sobre el espacio. Trayectoria que ya es objeto de comparación con la trayectoria real del sistema y que se evaluará en la sección 5.1 de este documento.

Tabla 1. Seguimiento de la ejecución de una trayectoria con nuestra integración de líneas en el sistema basado en ORB-SLAM.

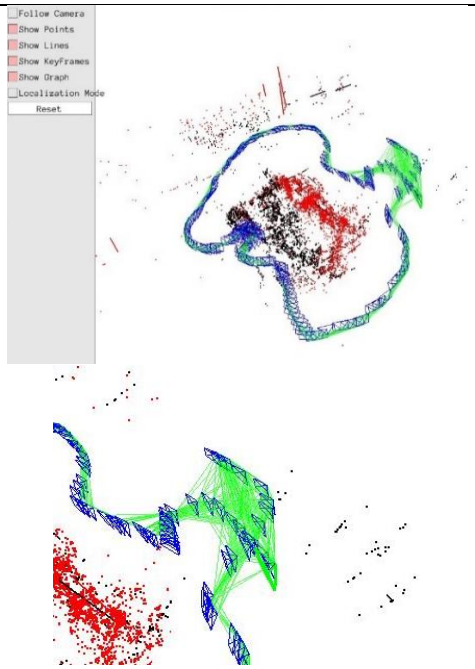
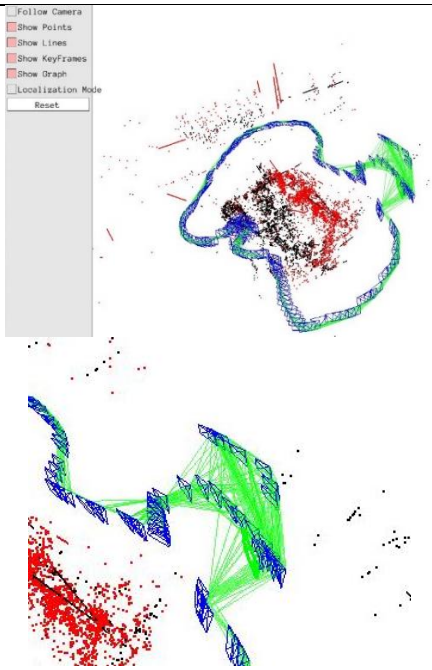
CIERRE DE BUCLE	
 **Figura 22.** Imagen previa a la detección del cierre de bucle.	 **Figura 23.** Depuración de fotogramas clave con el cierre de bucle detectado.
Cuando el sistema regresa a un punto anterior o un punto inicial, se presenta un cierre de bucle que el algoritmo debe detectar. En estas dos imágenes podemos evidenciar que el sistema lo reconoce correctamente pues se puede observar la eliminación de algunos fotogramas clave de acuerdo con lo mencionado anteriormente. También se puede evidenciar que el sistema puede encontrar bucles y seguir mapeando lo que observa.	

Tabla 2. Cierre de bucle de la ejecución de una trayectoria con nuestra integración de líneas en el sistema basado en ORB-SLAM.

3. SISTEMAS DE PROCESAMIENTO PARALELO

En la presente sección inicialmente se hace una descripción de las plataformas de procesamiento paralelo que se pusieron a prueba en el presente trabajo experimental, se da a conocer su funcionamiento y su estructura física. En la NVIDIA Jetson Nano por ser una tarjeta que ha estado sujeta a más experimentación, se hace una pequeña revisión bibliográfica para enseñar sus desempeños en algoritmos SLAM.

Aquí también se enseña el paso a paso de instalación de PL-SLAM que se siguió para conseguir la ejecución del algoritmo en el sistema de computación de NVIDIA Geforce GTX 1650 Ti y en la tarjeta de computación paralela NVIDIA Jetson Nano. Considerando que en ambos casos se construyó el sistema en Ubuntu 18.04 con el fin de garantizar que los procesos de instalación de librerías, dependencias y repositorios fueran realizados de la misma forma en ambas plataformas.

Finalmente se relata cronológicamente las experiencias obtenidas con su experimentación en la NVIDIA Jetson Nano.

3.1. NVIDIA JETSON NANO

Las capacidades computacionales han venido mostrando un importante aumento con el paso del tiempo, pues van creciendo a la par con las mejoras de las prestaciones de servicio de las redes de telecomunicaciones; lo que ha permitido que algunos desarrolladores ofrezcan servicios informáticos de forma remota[38]. Actualmente, existen tres entornos de computación distribuida escalable que procesan y distribuyen los datos; la computación *Cloud,* la computación *Fog* y la computación *Dew;* el primero, con rendimientos altos en potencia de cálculo; el segundo, con un rendimiento más enfocado a la fuente de datos; y el tercero, que procesa toda la información directamente en los dispositivos de uso [39].

El objetivo principal de *Dew Computing* es utilizar toda la potencia de los dispositivos evitando grandes impactos por pérdidas de conexiones a la red y a su vez ofreciendo una mejor seguridad y privacidad de los datos que procesan. Es por esto que, en 2019, Nvidia lanza el dispositivo llamado Jetson Nano, dispositivo que se direcciona al trabajo de tareas de Machine Learning. Este dispositivo presenta un bajo consumo energético y también buena compatibilidad con aplicaciones de Inteligencia Artificial; lo que añadido a su precio asequible facilita a los desarrolladores integrarlos en sus productos[40].

NVIDIA Jetson Nano permite el funcionamiento en paralelo de múltiples redes neuronales, cuenta JetPack SDK que es un entorno de desarrollo muy completo y con librerías ya desarrolladas para aplicaciones como aprendizaje profundo, visión por computador y otros más[41]. La placa también cuenta con una arquitectura homogénea CPU-GPU que permite que el sistema operativo sea arrancado por la CPU y programar la GPU para ser acelerado en función de algunas tareas complejas de aprendizaje automático [42].

Jetson Nano cuenta físicamente con una CPU Quad-Core ARM de 4 núcleos Cortex-A57 64-bit a 1.42 GHz, con una GPU NVIDIA Maxwell w/128 CUDA cores a 921 MHz, con una memoria de 4GB LPDDR4 a 1600 MHz, con conexión a red vía Ethernet y con adaptación para una tarjeta Wifi M.2 Key E, con un puerto HDMI 2.0 y un módulo *Embedded DisplayPort (Edp)* 1.4, puertos E/S digitales que permiten acomodar una gama de interfaces físicas: UART, USB, SPI con frecuencias de muestreo de 10 a 1000 fps y una potencia bajo carga de 5W-10W [8], [41], [43].

Aunque NVIDIA no tiene mucha experiencia en ORB-SLAM basada en Jetson Nano [44] algunos usuarios de la tarjeta han intentado implementar sistemas SLAM en la placa. Un estudio de 2019[45] demostró que la NVIDIA Jetson Nano no logra el objetivo SLAM en tiempo real (ejecutando ORB-SLAM2) sólo con el funcionamiento de su CPU; para que ella alcance los requerimientos en tiempo real es necesario habilitar el paralelismo GPU para que el sistema sobrepase los 10 fps, llegando hasta los 13.2 fps. En 2020 otros estudios lograron alcanzar el objetivo SLAM; por ejemplo *Zotov* en [43] logró que la Jetson Nano recopilara datos a 15 fps con una cámara de una resolución de 640x480; contemporáneamente, *Silveira et al.,* lograron resultados en tiempo real entre 11.83 y 16.54 fps[46].

3.2. GEFORCE GTX 1650 Ti

El constante avance de las tecnologías dedicadas a fines específicos ha ayudado a la mejora notable del poder computacional; un claro ejemplo de eso se puede observar con las supercomputadoras paralelas dedicadas a la Unidad de Procesamiento Gráfico (GPU). La aparición de las Unidades de Procesamiento Gráfico (GPU) y su rápida adopción como herramienta hardware, no solo para usos de aplicaciones relacionadas con videojuegos o 3D interactivas, sino también para computación general, hizo que se convirtieran una parte integral de los sistemas actuales de computación, pues el considerable incremento del rendimiento sumado a su bajo costo permitió que se incorporaran rápidamente en el mercado. Estas unidades con el paso del tiempo dejaron de ser sólo un procesador gráfico de gran potencia y pasaron a convertirse en co-procesadores de desarrollo y programación; lo que abrió un gran abanico de posibilidades para brindar ayudas a aplicaciones con necesidades altamente paralelizables y con gran requerimiento de recursos computacionales. Actualmente las GPU son una alternativa segura a los sistemas tradicionales de computación paralela[47], [48].

Existen diversas alternativas para el procesamiento en GPU, y una de las más utilizadas son las ofrecidas por NVIDIA, pues en ellas existe un entorno de desarrollo con un kit de programación en C y un control de hilos proporcionados por un driver que provee una interfaz GPU-CPU[49]. Ese entorno de desarrollo llamado CUDA (Compute Unified Device Architecture) se diseñó para simplificar el trabajo de sincronización de los hilos y la comunicación con la GPU[50].

El módulo GPU de NVIDIA que se usó en este trabajo fue una GeForce GTX 1650Ti, ella está diseñada con el rendimiento gráfico que le brinda la arquitectura NVIDIA Turing, que funciona como un potenciador para juegos y sistemas populares actuales. Cuenta con una frecuencia de base modificada de 1665 MHz y una velocidad de memoria de hasta 12GB/s, contiene núcleos CUDA que proporciona buen rendimiento a ejecuciones simultáneas de diferentes tareas y tiene 4GB de memoria RAM. Es una tarjeta de bajo consumo energético, pues está por debajo de los 75W, energía que le puede brindar la misma computadora y que hace que no requiera de alimentación extra; además, no presenta altas temperaturas cuando su rendimiento está al 100%, su temperatura no excede los 60 grados, comparado con otros sistemas que llegan a alcanzar los 90 grados y pesa 680g y mide 16 x 11,1 x 3,4 cm[51].

3.3. EXPERIENCIAS CON LA INTEGRACIÓN DEL SISTEMA DE PUNTOS Y LÍNEAS EN TARJETA DE PROCESAMIENTO PARALELO NVIDIA JETSON NANO

Inicialmente se decidió trabajar con el repositorio de SLAM adaptado y recomendado por NVIDIA para la Jetson Nano, obteniendo inicialmente que la placa era incapaz de compilar Pangolin de la forma que se recomendaba. Pues primero, no se especificaban las dependencias necesarias para que la biblioteca se pudiera instalar, además que tampoco se enseñaba la necesidad de compilar con MAKE que sirve para trabajar con proyectos de alta complejidad computacional. Finalmente, se encontró otro repositorio que indicaba los pasos que faltaban y que no permitían que Pangolin funcionara correctamente.

Luego de que se pudiera sobrepasar esa dificultad, el repositorio indicaba la instalación de OpenCV 4.1.0; instalación que se hace completa y sin percances. La dificultad se encontró cuando se iba a construir el algoritmo; la compilación mostraba un error con el archivo de cabecera ORBExtractor.h, pues en él no se encontraba un archivo llamado opencv/cv.h necesario para la instalación. Investigando sobre el error, se logró encontrar que el archivo se encontraba obsoleto a partir de OpenCV 4, pero que se podía reemplazar dentro del ORBExtractor.h por otro archivo de cabecera llamado opencv2/opencv.hpp.

Más adelante, en la construcción del mismo algoritmo, se menciona que es 'CvMat' no se encuentra declarado en algunos ficheros como System.h, PnPSolver.h y Sim3Solver.h; error al que se le dio solución agregando en esos ficheros las librerías de OpenCV:

```
#include <opencv2/imgproc/types_c.h>
#include <opencv2/opencv.hpp>
usando namespace cv;
```

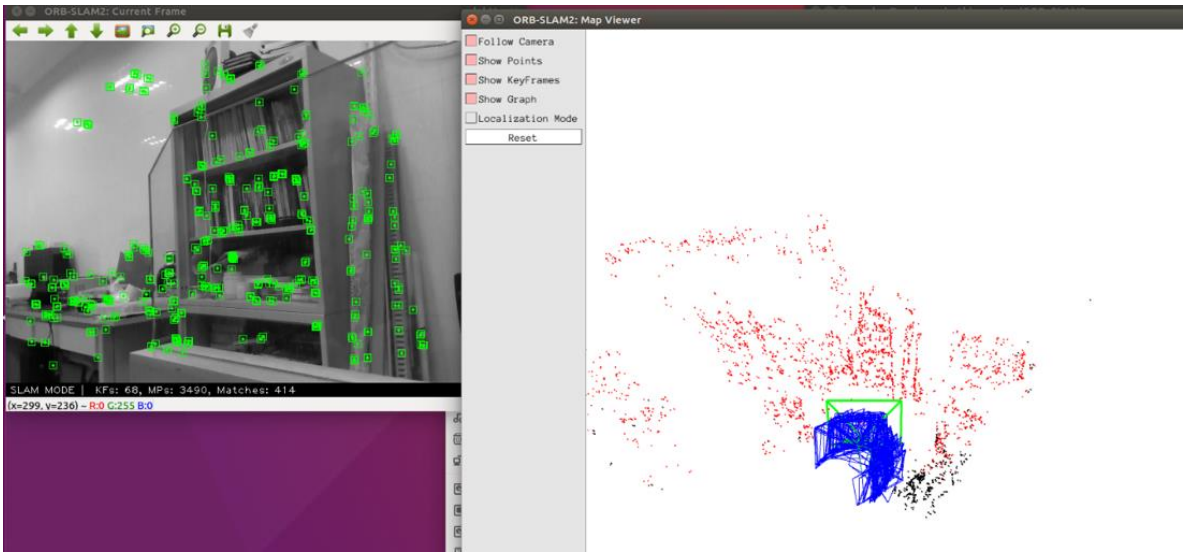


Figura 24. ORB-SLAM2 corriendo en la tarjeta NVIDIA Jetson Nano

Finalmente se logró ejecutar el algoritmo ORB-SLAM2 en la placa NVIDIA Jetson NANO. La *figura 24* es una ejecución de una secuencia de imágenes obtenida desde el dataset de KITTI. Es pertinente destacar que, a pesar de que NVIDIA recomienda observar estos repositorios para la instalación de ORB-SLAM en la tarjeta NVIDIA Jetson Nano, en una discusión que contenía nuestras dudas dentro del repositorio en cuestión, el autor del mismo mencionó que los pasos descritos en él sólo habían sido probados con las tarjetas NVIDA Jetson TX1 y TX2. La solución a todos estos problemas en la NVIDIA Jetson Nano se compartieron en el hilo #23 del repositorio en Github, "https://github.com/thien94/ORB_SLAM2_CUDA/issues/23", eso con el fin de facilitar la solución de algunos problemas de compilación a otros investigadores o usuarios de la tarjeta NVIDIA Jetson Nano.

Al corroborar que el sistema era capaz de ejecutar el algoritmo ORB-SLAM2, se procedió a ejecutar el algoritmo de puntos y líneas.

En esta ocasión los problemas se generaron con la instalación de OpenCV. El repositorio menciona que se requiere al menos la versión 2.4.3 de OpenCV y que ha sido probado con las versiones 2.4.11 y 3.2.

En el momento de hacer la instalación de las dependencias necesarias para la compilación de OpenCV 3.2.0 se presentaron problemas con la librería 'libjasper-dev'. Al parecer esa librería se encuentra obsoleta para Ubuntu 18.04 y ella impedía que la compilación de OpenCV se llevara a cabo exitosamente. Se encontraron soluciones en la investigación, pero algunas de ellas llevaron a que el sistema operativo fallara por completo. Hasta que se llegó a la forma de instalación que se menciona en la parte 2 del subcapítulo 3.3. de esta misma sección.

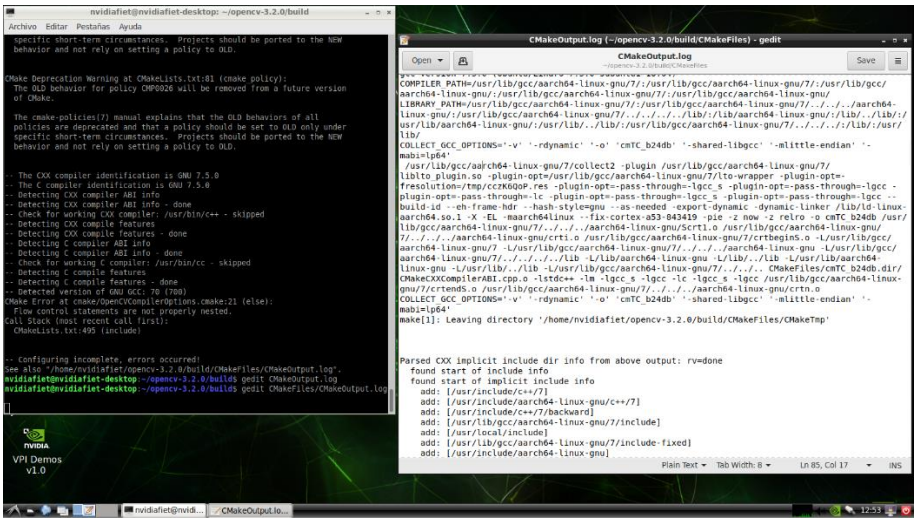


Figura 25. Error de instalación OpenCV 3.2.0.

Aun así, la instalación de esa versión de OpenCV siguió presentando problemas y nunca se logró instalar en la tarjeta a pesar de que se intentó solucionar los errores En todos los intentos de compilación se presentaba el error que se puede observar en la *Figura 25*. Es importante tener en cuenta este punto, ya que a raíz de esto se decidió intentar compilar el sistema en una máquina virtual con un sistema operativo de Ubuntu 18.04, para diagnosticar si el motivo del problema era la versión de ese sistema operativo. Encontrando que no, pues en el algoritmo fue compilado y ejecutado exitosamente en dicho equipo con el mismo sistema operativo.

Lograr ejecutar el algoritmo en otro equipo con el mismo sistema operativo siguiendo el mismo paso a paso de instalación con el que se pretendía instalar el algoritmo en la Jetson Nano desencadenó enfocar la explicación de todas las dificultades en otros aspectos. Lo que confluyó a los fallos que se van a presentar en la *figura 26.*

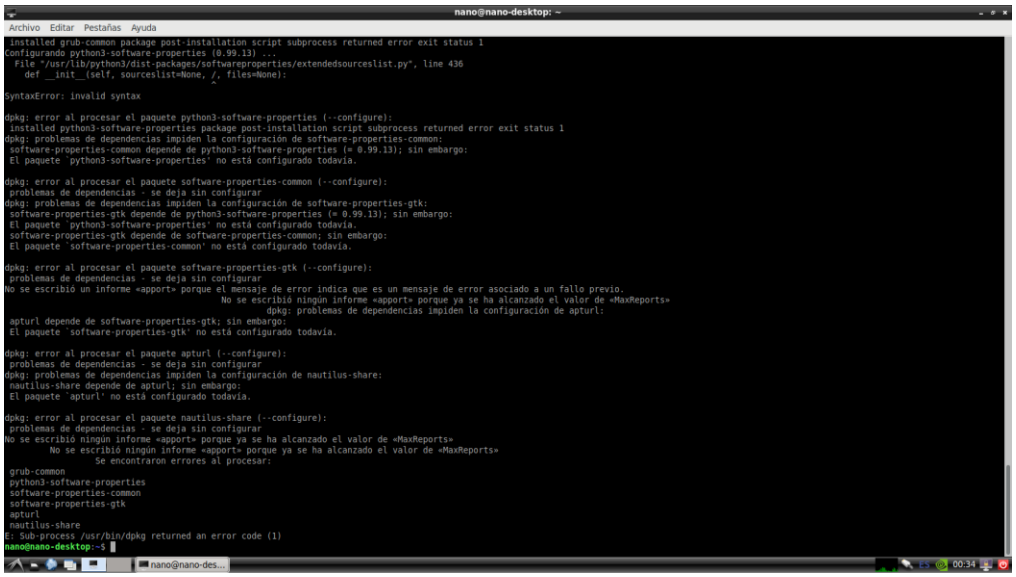


Figura 26. Error de configuración de dependencias.

Como se puede ver en la *Figura 26,* la respuesta del compilador en las soluciones probadas fue mencionar que el sistema tenía incorrectamente instaladas muchas dependencias, pero era imposible modificarlas porque ellas son propias del sistema operativo.

- **DIFERENCIAS HARDWARE ENTRE LA MV, LA JETSON NANO Y EL SISTEMA DE LA LITERATURA**

Característica	Jetson Nano	Máquina Virtual	Literatura*
Tamaño de memoria	1971 MiB	8171 MiB	30517.6 MiB
Número de núcleos	4	1	4
Modelo del Procesador	ARMv8 Processor rev 1 (v8l)	Intel® Core™ i5-10300H CPU @ 2.50GHz	Intel® Core™ i7-4790 a 3.60GHz
# Bits	64	64	64
CPU'S	Cantidad: 4 Capacidad: 1.479GHz	Cantidad: 1 Capacidad: 2.5GHz	Cantidad: 4 Capacidad: 4.0GHz
Gráficos del procesador	Failed to detect video card. Drivers: NVIDIA Resolución: 1024x768 @60Hz	VMware SVGA II adapter Drivers: vmware Resolución: 800x600 @60Hz	Intel ® HD Graphics 4600 Drivers: Intel HD. Resolución: 1920x1200 @60Hz

Tabla 3. Diferencias Hardware entre la máquina virtual, la Jetson Nano y el sistema de la literatura.

*__Literatura:__ Equipo usado en PL-SLAM: Real-time Monocular Visual SLAM with Points and Lines.

La *Tabla 3,* presenta las diferencias hardware entre dos sistemas que lograron ejecutar el algoritmo de puntos y líneas (Máquina virtual, y el equipo usado en PL-SLAM: Real-time Monocular Visual SLAM with Points and Lines), y el sistema en el que no se logró ejecutar el algoritmo (Jetson Nano) Encontrando que las mayores diferencias están en los tamaños de memoria, en la capacidad de las CPU's, en que la Jetson Nano es el único sistema con una tarjeta gráfica y en el modelo del procesador.

Se considera que las diferencias en el modelo del procesador es lo que da explicación a los problemas de configuración de las dependencias mostradas en la *figura 26* pues la Jetson Nano es la única placa que funciona con una arquitectura ARM.

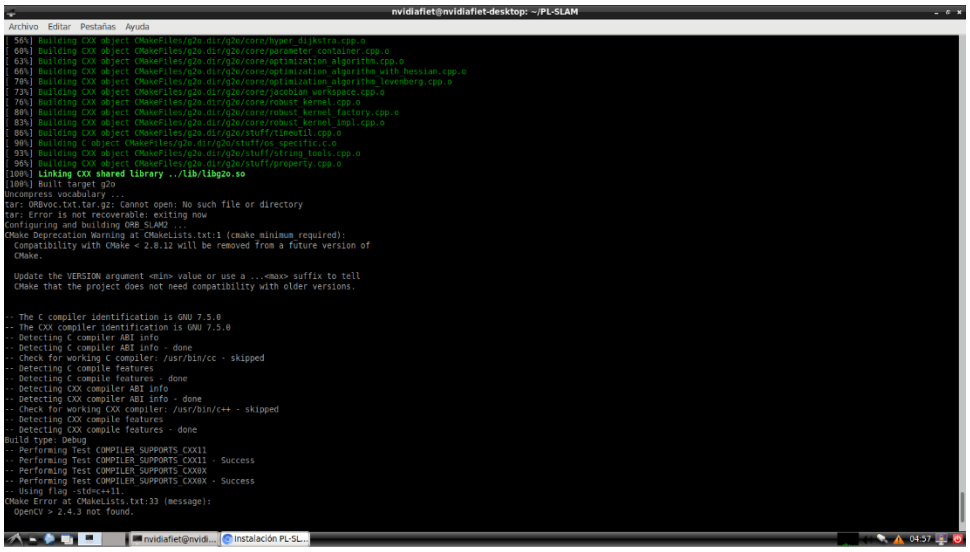


Figura 27. El sistema no encuentra la versión 2.4.3 de OpenCV

Siguiendo con las recomendaciones que se presentaban en el repositorio de instalación; se intentó compilar el sistema con la versión 2.4.11 de OpenCV. Versión que se pudo instalar correctamente. La dificultad se encontró en el momento de compilar el código de puntos y líneas, en el que se presentó la dificultad de que el sistema no encontraba la versión que le era necesaria para funcionar, error que se presenta en la *figura 27.* Para solucionar este error se hicieron cambios en el código fuente presentado en el repositorio pero aún así no se logró superar. Al parecer, el código no era compatible con esa versión de OpenCV.

Es importante mencionar que la versión 2.4.3 de OpenCV ya se encuentra obsoleta y ya no se ofrece en ninguna de las bases de datos con las versiones de OpenCV.

3.4. HABILITACIÓN DE LA EJECUCIÓN BAJO EL PARALELISMO DE NVIDIA GEFORCE GTX 1650 Ti

Debido a los problemas de compilación presentados en el experimento con la tarjeta NVIDIA Jetson Nano se compiló el algoritmo en una máquina virtual de VirtualBox que debe garantizar su funcionamiento bajo el paralelismo de la GPU NVIDIA GeForce 1650 Ti. Lo que se llevó a cabo como se explicará a continuación.

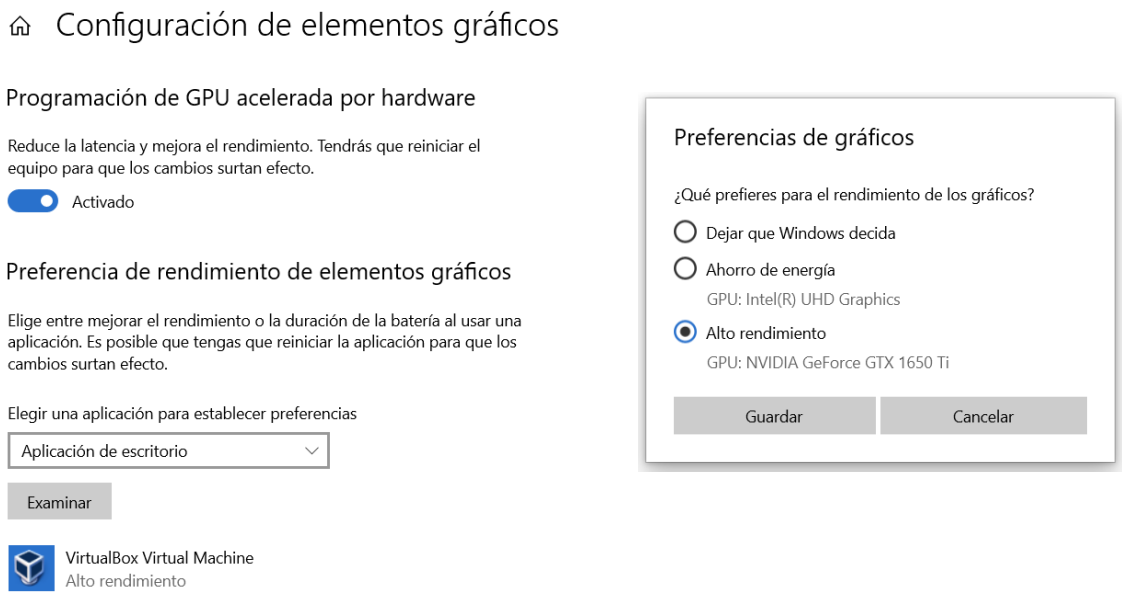


Figura 28. Habilitación del paralelismo de la GPU NVIDIA GeForce GTX 1650 Ti para la máquina virtual.

Para habilitar el paralelismo de la GPU NVIDIA GeForce GTX 1650Ti se debe ir a la configuración de elementos gráficos. Lo que se realiza dirigiéndose a *configuración* donde posteriormente se debe ingresar a *sistema;* ahí se debe permanecer en la pestaña de *pantalla* y se desplaza hasta la opción *configuración de elementos gráficos.*

Dentro de la configuración de elementos gráficos inicialmente se debe activar la programación de GPU acelerada por hardware; posteriormente se debe elegir una aplicación para establecer las preferencias, donde se debe escoger VirtualBox. Y finalmente cliqueando sobre el icono de VirtualBox se escogen las preferencias de gráficos de alto rendimiento ofrecida por la GPU NVIDIA GeForce GTX 1650 Ti, como se muestra en la *figura 28.*

3.5. INSTALACIÓN ALGORITMO DE PUNTOS Y LÍNEAS EN UBUNTU 18.04

PL-SLAM es una propuesta de solución al método de Visual-SLAM que permite procesar puntos y líneas simultáneamente abordando situaciones en las que los métodos basados únicamente en extracción de puntos son propensos a fallar. El sistema se basa en la arquitectura de ORB-SLAM2 y ha sido probado y comparado con algoritmos como ORB-SLAM, PTAM, LSD-SLAM y RGBD-SLAM, mostrando que PL-SLAM presenta mejores resultados en precisión de la trayectoria que ORB-SLAM en todas las secuencias en las que han sido comparados[37], [52]

El algoritmo PL-SLAM ha sido probado en Ubuntu 12.04, 14.04 y 16.04. Pero esta guía de instalación está hecha para Ubuntu 18.04.

1. INSTALACIÓN DE PANGOLIN.

Pangolin es un grupo de bibliotecas de utilidades livianas y portátiles hechas para crear prototipos de programas y algoritmos 3D, numéricos o basados en video. Es de mucho uso en el campo de la visión por computadora, ya que se considera un medio para eliminar texto de la plataforma y facilitar la visualización de datos [53].

Instalación de dependencias

- **Instalación de CMake**
  CMake es una herramienta que se utiliza para la creación de código fuente. CMake genera sistemas modernos de compilación y también proyectos para entornos de desarrollo integrado como Visual Studio y Xcode. CMake usa normalmente los lenguajes C y C++, pero también puede compilar código fuente de otros lenguajes[54].

```
sudo apt-get install cmake
sudo apt-get install checkinstall
```

- **Instalación de gcc y g++**
  GNU es un sistema operativo de software libre de tipo Unix, es decir que se trata de una colección de diversos programas, aplicaciones, bibliotecas, entornos de desarrollo y juegos que se usa generalmente en Linux [55]. GCC son originalmente los compiladores del sistema GNU en Linux, el primero para el lenguaje C y el segundo para C++ [56].

```
sudo apt-get install g++
```

```
sudo apt-get install gcc
sudo ldconfig
```

- **Otras dependencias**

```
sudo apt-get install libglew-dev libpython2.7-dev
sudo apt-get install git
```

- **Instalación de Pangolin en el sistema local**

```
git clone https://github.com/stevenlovegrove/Pangolin.git
cd Pangolin
./scripts/install_prerequisites.sh recommended
mkdir build
cd build
cmake ..
make -j4
```
(4 por el número de procesadores que tiene el sistema)
```
sudo make install
```

- **Correr ejemplo "HelloPangolin"**

```
cd examples
cd HelloPangolin
mkdir build
cd build
cmake ..
make
./HelloPangolin
```

2. INSTALACIÓN OPENCV 3.2.0

OpenCV es una biblioteca software de aprendizaje automático y visión artificial de código abierto. Se creó con el fin de que las aplicaciones de visión por computadora tuvieran una infraestructura común y para acelerar las percepciones de las máquinas en productos comerciales. La biblioteca está escrita en C y C++ y se puede ejecutar tanto en Linux como en Windows y en Mac OS X. Es importante porque se encuentra en constantes desarrollos en Python, Ruby, Matlab y otros lenguajes [57], [58].

```
sudo apt-get update
sudo apt-get upgrade
```

- **Instalación de dependencia "libjasper-dev"**

```
sudo apt-get install nano
cd /etc/apt/sources.list.d
```

Cree un archivo de texto de la siguiente forma:

```
sudo nano multistrap-main.list
```

Agregue las siguientes lineas:

```
deb http://ports.ubuntu.com/ubuntu-ports xenial-security main
deb http://ports.ubuntu.com/ubuntu-ports impish main
```

Guarde con ctrl+s o ctrl+o y salga de la edición del archivo con ctrl+x

Actualice.

```
cd..
sudo apt-get update
```

Actualice claves con la siguiente línea:

```
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys
<key>
```

Instale la librería libjasper-dev

```
sudo apt-get install libjasper-dev
```

Elimine el archivo multistrap-main.list

```
sudo rm -r multistrap-main.list
```

- **Instalación de otras dependencias**

```
sudo apt-get install build-essential libgtk2.0-dev libjpeg-dev libtiff5-dev libjasper-dev libopenexr-dev cmake python-dev python-numpy python-tk libtbb-dev libeigen3-dev yasm libfaac-dev libopencore-amrnb-dev libopencore-amrwb-dev libtheora-dev libvorbis-dev libxvidcore-dev libx264-dev libqt4-dev libqt4-opengl-dev sphinx-common texlive-latex-extra libv4l-dev libdc1394-22-dev libavcodec-dev libavformat-dev libswscale-dev default-jdk ant libvtk5-qt4-dev
```

```
sudo apt-get install qt5-default
```

- **Obtenga el código fuente de OpenCV 3.2.0**

```
wget https://github.com/opencv/opencv/archive/3.2.0.zip
```

```
unzip 3.2.0.zip
```

- **Obtenga el código fuente de Opencv-Contrib**

```
wget https://github.com/opencv/opencv_contrib/archive/3.2.0.zip
```

```
unzip 3.2.0.zip.1
```

- **Inicie la instalación de OpenCV 3.2.0**

```
cd opencv-3.2.0
```

```
mkdir build
```

```
cd build
```

Genere la configuración de compilación

```
cmake -D WITH_TBB=ON -D BUILD_NEW_PYTHON_SUPPORT=ON
-D   WITH_V4L=ON   -D   INSTALL_C_EXAMPLES=ON   -D
INSTALL_PYTHON_EXAMPLES=ON  -D  BUILD_EXAMPLES=ON  -
DCMAKE_BUILD_TYPE=RELEASE -
DENABLE_PRECOMPILED_HEADERS=OFF -D
CMAKE_INSTALL_PREFIX=/usr/local                    -
DOPENCV_EXTRA_MODULES_PATH=../../opencv_contrib-
3.2.0/modules ..
```

Compile teniendo en cuenta el número de procesadores del sistema (-j#)

```
make -j4
```

Instale el paquete

```
sudo make install
```

Registre las bibliotecas y los módulos

```
sudo   /bin/bash   -c   'echo   "/usr/local/lib"   >
/etc/ld.so.conf.d/opencv.conf'

sudo ldconfig
```

Desinstale la versión antigua de OpenCV

```
sudo apt-get autoremove libopencv-dev
```

Verifique la nueva versión instalada ejecutando

```
pkg-config --modversion opencv
```

3. INSTALACIÓN DE OTRAS LIBRERÍAS

- **Eigen3**
  Eigen es una librería con plantillas de C++ dedicada al procesamiento de operaciones de álgebra lineal: vectores, matrices, solucionadores numéricos y algoritmos relacionados [59].

```
sudo apt-get install libeigen3-dev
```

- **DBoW2**
  DBoW2 es una biblioteca de C++ de código abierto que se utiliza para ordenar información de las imágenes en una representación de bolsa de palabras. También implementa una base de datos de imágenes organizadas que permite hacer consultas rápidas y comparaciones de característica. DBoW2 es una versión mejorada de la biblioteca DBoW [60].

```
git clone https://github.com/dorian3d/DBoW2.git
cd DBoW2
mkdir build
cd build
cmake .. -DCMAKE_BUILD_TYPE=Release
make
```

- **Ceres**
  Ceres Solver es una biblioteca de C++ de código abierto que se utiliza especialmente para modelar y resolver problemas de optimización con un alto grado de complejidad. Se puede utilizar para resolver problemas de mínimos cuadrados no lineales con restricciones de límites y problemas generales de optmización sin restricciones [61].

```
sudo apt-get install cmake
sudo apt-get install libgoogle-glog-dev libgflags-dev
sudo apt-get install libatlas-base-dev
sudo apt-get install libeigen3-dev
sudo apt-get install libsuitesparse-dev
```

```
git clone https://ceres-solver.googlesource.com/ceres-solver
          cd ceres solver
          mkdir ceres-bin
          cd ceres-bin
          cmake ../
          make -j4
          make test
          sed -i 's/++11/++14/g' CMakeLists.txt
          sudo make install
```

- **CMake 3.20.0**

```
          sudo apt remove cmake
          sudo apt-get install build-essential libssl-dev
wget https://github.com/Kitware/CMake/releases/download/v3.20.0/cmake-3.20.0.tar.gz
          tar -zxvf cmake-3.20.0.tar.gz
          cd cmake-3.20.0
          ./bootstrap
          make
          sudo make install
          PATH=$PATH:/opt/cmake/bin
          . ~/.bashrc
          cmake --version
```

- **g2o**

  g2o es una librería de código abierto de C++ propuesta para optimizar algunas funciones de error no lineal basada en gráficos. La implementación de g2o actualmente brinda soluciones a las variantes de los sistema SLAM, pues se

diseñó para ser extensible fácilmente a una amplia gama de problemas y se implementa con pocas líneas de código [62].

```
git clone https://github.com/RainerKuemmerle/g2o.git
     cd g2o
     mkdir build
     cd build
     sudo cmake ..
     sudo make
     sudo make install
```

4. INSTALACIÓN Y COMPILACIÓN DEL ALGORITMO DE PUNTOS Y LÍNEAS.

Clone el repositorio

```
git clone https://github.com/HarborC/PL-SLAM
```

Es necesario añadir el vocabulario ORBvoc.txt, para eso descargue:

https://github.com/raulmur/ORB_SLAM2/raw/master/Vocabulary/ORBvoc.txt.tar.gz

Descomprima y llévelo a la carpeta 'Vocabulary' dentro de la carpeta PL-SLAM

```
cd  PL-SLAM
chmod +x build.sh
./build.sh
```

Descargue una secuencia de la base de datos de TUM RGB-D y reemplace **"PATH_TO_SEQUENCE_FOLDER"** por el nombre de la carpeta que contiene la secuencia dentro de la carpeta PL-SLAM.

```
./Examples/Monocular/mono_tum Vocabulary/ORBvoc.txt
Examples/Monocular/TUMX.yaml PATH_TO_SEQUENCE_FOLDER
```

4. ESTABLECIMIENTO DE LAS PRUEBAS CON LA BASE DE DATOS DE TUM.

El conjunto de datos RGB-D del dataset de TUM contiene secuencias en interiores procedentes de sensores (cámaras) RGB-D. Esas secuencias están agrupadas en varias categorías, con ellas se ponen a prueba los algoritmos dedicados a la reconstrucción de objetos y los métodos SLAM. El dataset ofrece secuencias en distintas condiciones de textura, iluminación, estructura y el procesamiento de objetos dinámicos [20]. A continuación, se describen las trayectorias con las que se evaluó el sistema mencionado anteriormente.

4.1. DESCRIPCIÓN DE LAS TRAYECTORIAS.

Las descripciones que se realizan a continuación, se reposan en el sitio web de TUM RGB-D.

1. **Secuencia 'freiburg1_xyz'**

Figura 29. Imagen de la secuencia **'freiburg1_xyz'.**

La *figura 29* presenta una imagen de la secuencia **'freiburg1_xyz'**; en ella el Kinect se apuntó a un escritorio típico en un entorno de oficina con objetos que brindan la posibilidad de observar una gran cantidad de información de bordes y de esquinas. Esta secuencia contiene solo movimientos de traslación a lo largo de los ejes principales del Kinect, mientras que la orientación se mantuvo (en su mayoría) fija. Esta secuencia es muy adecuada para fines de depuración, ya que es muy simple [63]. Tiene una duración de 30.09 segundos.

2. **Secuencia 'freiburg2_xyz'**

Figura 30. Imagen de la secuencia **'freiburg2_xyz'.**

La *figura 30* presenta una imagen del conjunto de imágenes contenida en la secuencia **'freiburg2_xyz'.** Esta secuencia dura 122,74 segundos y contiene datos muy limpios para depurar traducciones. El Kinect se movió a lo largo de los ejes principales en las direcciones *x, y* y *z* muy lentamente. El movimiento lento de la cámara básicamente asegura que no haya (casi) desenfoque de movimiento ni efectos de obturador rodante en los datos[63].

3. **Secuencia 'freiburg2_desk_with_person'**

Figura 31. Imagen de la secuencia **'freiburg2_desk_with_person'.**

La *figura 31* presenta la secuencia **'freiburg2_desk_with_person'** que es una escena típica de oficina con una persona sentada en un escritorio. Durante la grabación la persona se movía e interactuaba con algunos de los objetos (pantalla, teléfono,..). Esta secuencia dura 142,08 segundos y está destinada a comprobar la solidez de un sistema SLAM frente a objetos y personas dinámicos, pero también puede utilizarse para diferenciar mapas y encontrar cambios en la escena[63].

4. **Secuencia 'freiburg3_long_office_household'**

Figura 32. Imagen de la secuencia **'freiburg3_long_office_household'.**

La *figura 32* presenta la secuencia **'freiburg3_long_office_household'** en ella el sensor Asus Xtion se mueve a través de una escena de hogar y oficina con mucha textura y estructura. Esta secuencia dura 87,09 segundos y en el final de la trayectoria se superpone con el comienzo (el robot regresa a la posición inicial), de modo que hay un gran cierre de bucle[63].

5. **Secuencia 'freiburg3_nostructure_texture_near_withloop'**

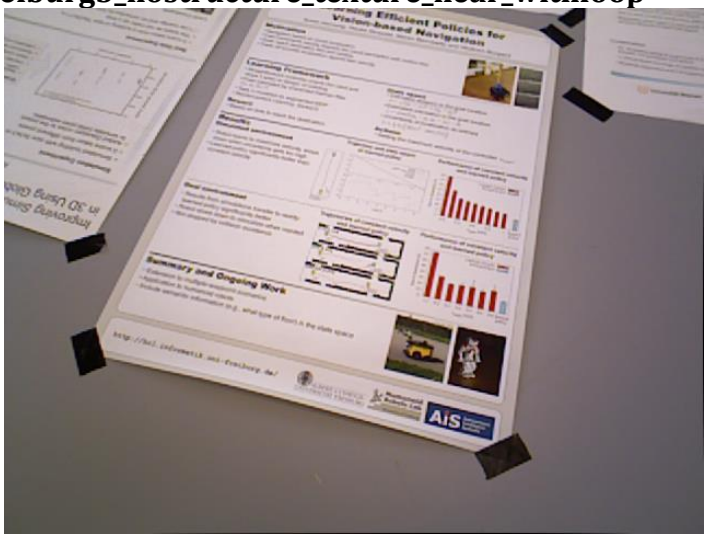


Figura 33. Imagen de la secuencia **'freiburg3_nostructure_texture_near_withloop'.**

La *figura 33* presenta la secuencia de 56,48 segundos de duración llamada **'freiburg3_nostructure_texture_near_withloop',** en la que el sensor Asus Xtion se mueve a un metro de altura en un círculo sobre una superficie plana texturizada. La textura es muy discriminatoria ya que consta de varios carteles de conferencias. El comienzo y el final de la trayectoria se superponen, de modo que hay un cierre de bucle[63].

6. **Secuencia 'freiburg3_structure_texture_near'**

Figura 34. Imagen de la secuencia **'freiburg3_structure_texture_near'.**

La *figura 34* es una imagen extraída de la secuencia de 36,91 segundos que se llama **'freiburg3_structure_texture_near'.** En ella, el Asus Xtion se movió a medio metro de altura a lo largo de una estructura en zig-zag construida con paneles de madera. El objeto está completamente envuelto en una lámina de plástico de colores con una textura fuerte. El suelo frente al objeto ha sido cubierto con varios carteles que también tienen una textura fuerte[63].

7. **Secuencia 'freiburg3_sitting_xyz'**

Figura 35. Imagen de la secuencia **'freiburg3_sitting_xyz'.**

En la *figura 35* se puede ver una imagen de la secuencia **'freiburg3_sitting_xyz'.** En dicha secuencia dos personas se sientan en un escritorio, hablan y gesticulan un poco. El sensor Asus Xtion se ha movido manualmente en tres direcciones *x, y* y *z,* manteniendo la misma orientación. Esta secuencia dura 42,50 segundos y está destinada a evaluar la solidez de los algoritmos de odometría y SLAM visual para objetos dinámicos que se mueven lentamente[63].

8. **Secuencia 'freiburg3_sitting_halfsphere'**

Figura 36. Imagen de la secuencia **'freiburg3_sitting_halfsphere'.**

En la *figura 36,* se presenta un extracto de la secuencia **'freiburg3_sitting_halfsphere',** secuencia en la que dos personas se sientan en un escritorio, hablan y gesticulan un poco. El sensor Asus Xtion se ha movido sobre una pequeña media esfera de aproximadamente un metro de diámetro. Esta secuencia tiene una duración de 37,15 segundos y está destinada a evaluar la solidez de los algoritmos de odometría y SLAM visual para objetos dinámicos que se mueven lentamente[63].

9. **Secuencia 'freiburg3_walking_xyz'**

Figura 37. Imagen de la secuencia **'freiburg3_walking_xyz'.**

En la *figura 37* se presenta una imagen de la secuencia **'freiburg3_walking_xyz',** secuencia en la que dos personas caminan alrededor de una escena de oficina. El sensor Asus Xtion se ha movido manualmente en tres direcciones *x, y* y *z* manteniendo la misma orientación. Esta secuencia tiene una duración de 28,83 segundos y está destinada a evaluar la solidez de los algoritmos de odometría y SLAM visual para objetos dinámicos que se mueven rápidamente en gran parte de la escena visible[63].

10. Secuencia 'freiburg3_walking_halfsphere'

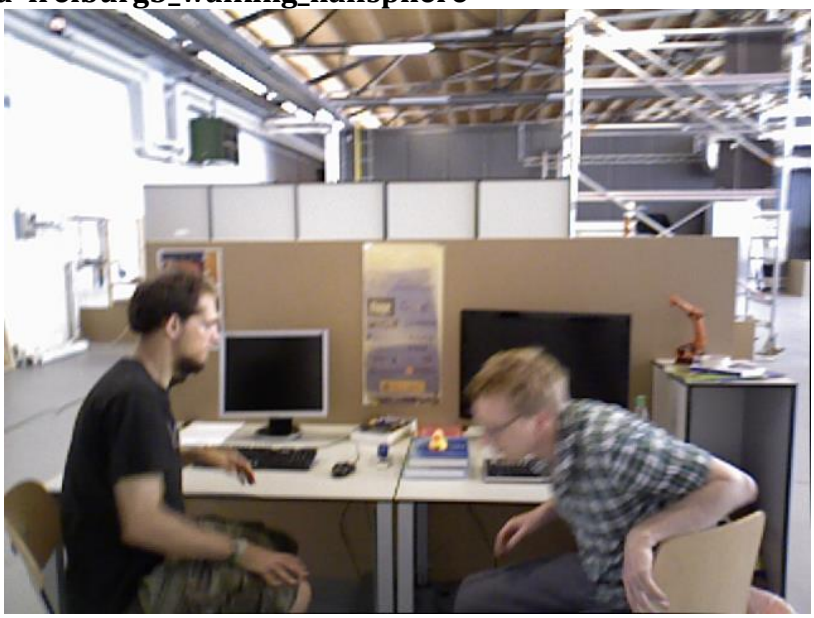


Figura 38. Imagen de la secuencia **'freiburg3_walking_halfsphere'.**

La *figura 38* presenta una imagen de la secuencia **'freiburg3_walking_halfsphere',** en esa secuencia dos personas caminan por una escena de oficina. El sensor Asus Xtion se ha movido sobre una pequeña media esfera de aproximadamente un metro de diámetro. Esta secuencia tiene una duración de 35,81 segundos y está destinada a evaluar la solidez de los algoritmos de odometría y SLAM visual para objetos dinámicos que se mueven rápidamente en gran parte de la escena visible[63].

4.2. INDICADORES DE DESEMPEÑO.

INDICADORES DE DESEMPEÑO PARA LA TRAYECTORIA.

- **Raíz Cuadrada del Error Cuadrático Medio (RMSE):** La raíz cuadrada del error cuadrático medio determina la cantidad de error que hay entre dos conjuntos de datos. Eso lo logra comparando un valor predicho con un valor observado[64].

$$RMSE = \frac{1}{n}\sqrt{\sum_{i=1}^{n}(real_i - estimado_i)^2}$$

- **Error Medio:** El error medio es la medida de la diferencia entre dos variables continuas, es una herramienta usada para cuantificar la precisión de una técnica. Es sensible a errores en los datos; un dato erróneo puede cambiar significativamente el valor del error medio[64].

$$MAE = \frac{1}{n}\sum_{i=1}^{n}(real_i - estimado_i)$$

- **Mediana del error:** La mediana del error define el valor que ocupa el lugar central de todos los candidatos cuando ellos están ordenados de menor a mayor. El propósito de la mediana es reflejar la tendencia central de la muestra de manera que no sea influida por valores extremos[65].

$$\begin{cases} x_{(\frac{n+1}{2})}, & si\ n\ es\ impar \\ \frac{1}{2}(x_{(\frac{n}{2})} + x_{(\frac{n}{2}+1)}), & si\ n\ es\ par \end{cases}$$

- **Error mínimo:** El error mínimo es el valor estimado más cercano al valor real, es la posición en la que el sistema estuvo más cerca del valor real para la misma marca de tiempo del recorrido.

- **Error máximo:** El error máximo es el valor estimado más lejano al valor real, es la posición en la que el robot estuvo más lejos del valor real para la misma marca de tiempo del recorrido.

5. EVALUACIÓN DE RESULTADOS

A continuación, se presentan los resultados del sistema para cada una de las trayectorias mencionadas en el capítulo anterior. Se debe tener en cuenta que las secuencias han sido probadas en una tarjeta GeForce GTX 1650 Ti de la familia de NVIDIA.

5.1. ANÁLISIS MATEMÁTICOS DE LOS DATOS.

PL-SLAM se corrió en 10 trayectorias del dataset de TUM RGB-D, esas trayectorias se probaron 10 veces y en cada una de esas ejecuciones se extrajeron los datos iniciales que fueron; la media del tiempo de seguimiento, la mediana del tiempo de seguimiento y el KeyFrameTrajectory. Éste último, es una matriz que contiene los argumentos posicionales de la ejecución según las marcas de tiempo.

Para obtener los datos como la Raíz del Error Cuadrático Medio, el error medio, la mediana del error, el error estándar, el error mínimo y el error máximo; los resultados de la trayectoria fueron procesados por la herramienta de evaluación automática online ofrecida por TUM. Finalmente, a cada uno de los datos mencionados anteriormente se les obtiene la media con el fin de facilitar la comparación del rendimiento del sistema con otros algoritmos o con ejecuciones en otras tarjetas de procesamiento paralelo.

Los datos de precisión se presentan en metros y los datos de tiempo en unidades de segundo.

- Análisis de la ejecución de la trayectoria 'freiburg3_long_office_household'.

EVALUACIÓN DE LAS PRUEBAS DE LA TRAYECTORIA 'freiburg3_long_office_household'								
Número de Prueba	RMSE	Error Medio	Mediana de Error	Error Estándar	Error Mínimo	Error Máximo	Median Tracking Time	Mean Tracking Time
1	0.046693	0.042619	0.041975	0.019075	0.0088346	0.098965	0.180186	0.183960
2	0.054165	0.051669	0.050648	0.016256	0.013339	0.095922	0.191986	0.195748
3	0.039585	0.034073	0.028986	0.20148	0.005536	0.172074	0.232539	0.240558
4	0.037076	0.031318	0.027147	0.019844	0.004794	0.131987	0.191895	0.190537
5	0.075886	0.071986	0.071325	0.024015	0.023192	0.125767	0.198739	0.203818
6	0.084918	0.079503	0.072500	0.029839	0.039894	0.193884	0.190646	0.196704
7	0.062864	0.046171	0.032738	0.042662	0.006846	0.265523	0.193748	0.192627
8	0.095450	0.051440	0.036551	0.080403	0.008867	0.643758	0.191526	0.191689
9	0.042757	0.039340	0.036337	0.016748	0.005397	0.081093	0.191448	0.189877
10	0.051333	0.044853	0.042907	0.024965	0.011291	0.154583	0.192407	0.1993551
Media	0.0590727	0.0492972	0.0441114	0.0475287	0.01279906	0.1963556	0.195512	0.19848731

Tabla 4. Análisis matemático de los datos para la secuencia **'freiburg3_long_office_household'.**

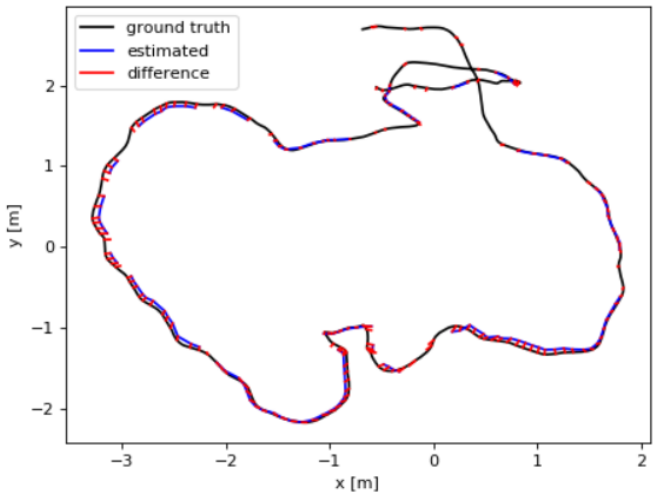


Figura 39. Trayectoria real vs trayectoria estimada de la secuencia **'freiburg3_long_office_household'.**

- **ANÁLISIS DE LA PRECISIÓN DE LA PRUEBA BAJO LOS INDICADORES DE DESEMPEÑO.**

  Como lo muestra la *tabla 4,* la trayectoria **'freiburg3_long_office_household'** se probó en 10 ocasiones. Cada una de esas pruebas fue sujeta a observación y evaluación bajo los parámetros de desempeño ya explicados en el capítulo anterior. Después de las 10 ejecuciones, se procede a encontrar la media de todos los datos extraídos para cada una de las ejecuciones.

  Teniendo en cuenta que los datos presentados están en metros, podemos especificar que:

  - El promedio de la raíz cuadrada del error cuadrático medio para esta trayectoria es de 5.9cm.
  - El promedio del error medio para esta trayectoria es de 4.9cm.
  - El promedio de la mediana del error para esta trayectoria es de 4.4cm.
  - El promedio del error estándar para esta trayectoria es de 4.7cm.
  - El promedio del error mínimo para esta trayectoria es de 1.2cm.
  - El promedio del error máximo para esta trayectoria es de19.63cm.

  El algoritmo, por cada prueba que se realiza entrega una media y una mediana de los tiempos de seguimiento. Encontrando que:

  - El promedio de la mediana del tiempo de seguimiento es de 0.195s.
  - El promedio de la media del tiempo de seguimiento es de 0.198s.

- **ANÁLISIS GRÁFICO DEL RESULTADO.**
  La *figura 39* compara la trayectoria real de (en negro, *'Ground truth'*) con la trayectoria estimada (en azul, *'estimated'*) para una de las 10 pruebas que se hicieron de la trayectoria **'freiburg3_long_office_household'**. Enseñando como resultado el seguimiento del recorrido del sistema para cada momento y evidenciando la diferencia gráfica (en rojo, '*difference'*) de la estimación de la posición del sistema con la posición real del sistema para cada momento.

- Análisis de la ejecución de la trayectoria 'freiburg1_xyz'.

EVALUACIÓN DE LAS PRUEBAS DE LA TRAYECTORIA 'freiburg1_xyz'								
Número de Prueba	RMSE	Error Medio	Mediana de Error	Error Estándar	Error Mínimo	Error Máximo	Median Tracking Time	Mean Tracking Time
1	0.008249	0.007143	0.006858	0.004126	0.001768	0.020934	0.170898	0.164019
2	0.009186	0.008024	0.007871	0.004472	0.001161	0.020994	0.176884	0.172241
3	0.014198	0.012852	0.012603	0.006035	0.004365	0.027702	0.178969	0.172317
4	0.010402	0.008800	0.007977	0.005546	0.001090	0.023573	0.178122	0.172703
5	0.008653	0.007104	0.006229	0.004940	0.001512	0.019186	0.18463	0.179505
6	0.008451	0.006798	0.005432	0.005021	0.000596	0.025008	0.184175	0.181436
7	0.008138	0.007095	0.005965	0.003985	0.00116	0.014961	0.178933	0.181825
8	0.008006	0.007318	0.007240	0.003247	0.000597	0.016420	0.16918	0.162862
9	0.009403	0.008258	0.007467	0.004497	0.001577	0.020465	0.17989	0.172495
10	0.009902	0.008141	0.006655	0.005637	0.001682	0.022869	0.185928	0.192
Media	0.0094588	0.0081533	0.0074297	0.0047506	0.0015508	0.0212112	0.1787609	0.1751403

Tabla 5. Análisis matemático de los datos para la secuencia **'freiburg1_xyz'**

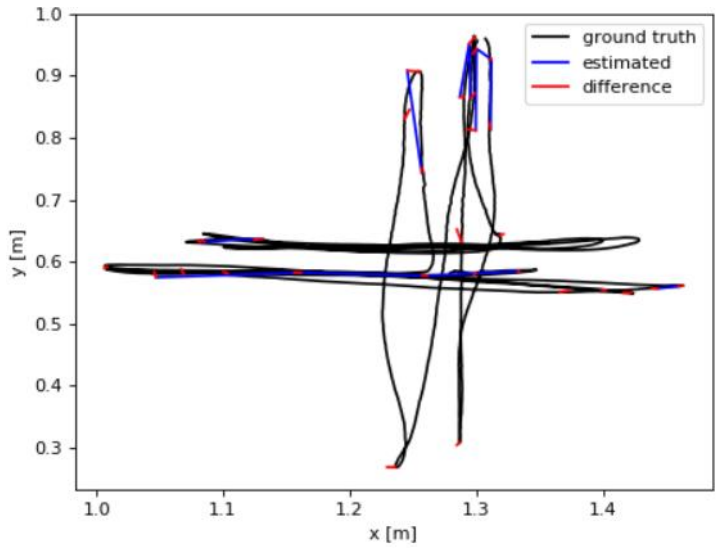


Figura 40. Trayectoria real vs trayectoria estimada de la secuencia **'freiburg1_xyz'**

- Evaluación de la trayectoria 'freiburg2_xyz'.

EVALUACIÓN DE LAS PRUEBAS DE LA TRAYECTORIA 'freiburg2_xyz'								
Número de Prueba	RMSE	Error Medio	Mediana de Error	Error Estándar	Error Mínimo	Error Máximo	Median Tracking Time	Mean Tracking Time
1	0.003278	0.003017	0.002979	0.001284	0.001046	0.005887	0.326025	0.335758
2	0.002956	0.002691	0.002538	0.001223	0.000268	0.005059	0.210323	0.217216
3	0.002800	0.002570	0.002395	0.001113	0.000788	0.004879	0.168667	0.165238
4	0.002673	0.002439	0.002275	0.001093	0.000507	0.004806	0.203114	0.199612
5	0.003187	0.002841	0.003027	0.001444	0.000662	0.005898	0.186081	0.185771
6	0.003399	0.003111	0.003126	0.001369	0.000634	0.006034	0.271198	0.279755
7	0.002411	0.002140	0.001863	0.001111	0.000480	0.004651	0.206223	0.211979
8	0.002841	0.002564	0.001993	0.001223	0.000491	0.005628	0.184212	0.183357
9	0.002855	0.002685	0.002529	0.000972	0.000719	0.005063	0.171208	0.17132
10	0.002502	0.002300	0.002206	0.000984	0.000912	0.004469	0.293944	0.296411
Media	0.0028902	0.0026358	0.0024931	0.0011816	0.0006507	0.0052374	0.2220995	0.2246417

Tabla 6. Análisis matemático de los datos para la secuencia **'freiburg2_xyz'.**

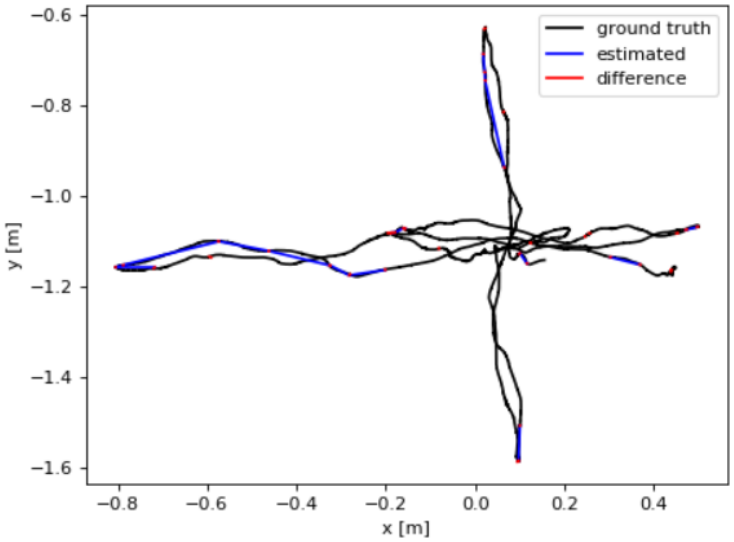


Figura 41. Trayectoria real vs trayectoria estimada de la secuencia **'freiburg2_xyz'.**

- Análisis de la ejecución de la trayectoria 'freiburg2_desk_with_person'.

EVALUACIÓN DE LAS PRUEBAS DE LA TRAYECTORIA 'freiburg2_desk_with_person'								
Número de Prueba	RMSE	Error Medio	Mediana de Error	Error Estándar	Error Mínimo	Error Máximo	Median Tracking Time	Mean Tracking Time
1	0.009473	0.008400	0.007186	0.004380	0.002722	0.021294	0.219988	0.228794
2	0.040702	0.039123	0.037710	0.011230	0.016745	0.068760	0.202755	0.205531
3	0.025912	0.024778	0.024604	0.007584	0.008170	0.066229	0.199217	0.199509
4	0.065753	0.063895	0.068576	0.015518	0.032130	0.096213	0.194492	0.204435
5	0.010214	0.008829	0.007771	0.005136	0.001267	0.022365	0.191927	0.201154
6	0.012970	0.012022	0.011133	0.004869	0.004680	0.026557	0.194246	0.20520
7	0.038783	0.037193	0.038657	0.010993	0.012197	0.056996	0.208535	0.218234
8	0.018226	0.016645	0.015091	0.007426	0.002536	0.033405	0.184346	0.189698
9	0.036242	0.033922	0.031669	0.012759	0.007988	0.067427	0.204268	0.220964
10	0.019587	0.017795	0.015647	0.008184	0.004613	0.036754	0.198395	0.204860
Media	0.0277862	0.0262602	0.0258044	0.0088079	0.0093048	0.04960	0.1998169	0.2078379

Tabla 7. Análisis matemático de los datos para la secuencia **'freiburg2_desk_with_person'.**

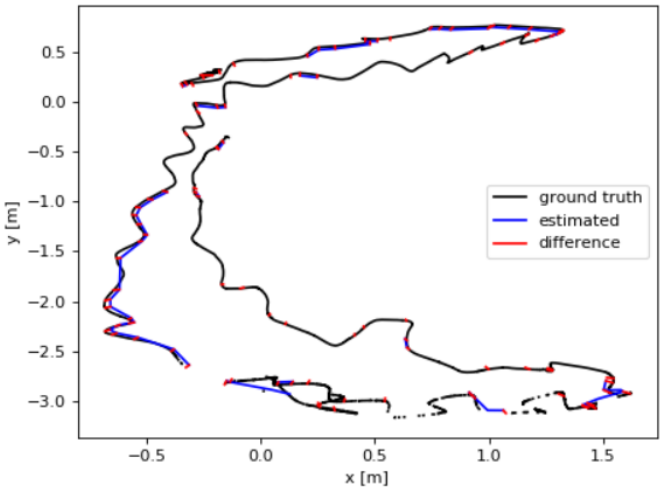


Figura 42. Trayectoria real vs trayectoria estimada de la secuencia **'freiburg2_desk_with_person'.**

- Análisis de la ejecución de la trayectoria 'freiburg3_nostructure_texture_near'.

EVALUACIÓN DE LAS PRUEBAS DE LA TRAYECTORIA 'freiburg3_nostructure_texture_near'								
Número de Prueba	RMSE	Error Medio	Mediana de Error	Error Estándar	Error Mínimo	Error Máximo	Median Tracking Time	Mean Tracking Time
1	0.146598	0.142504	0.141467	0.034405	0.0623227	0.218032	0.272837	0.280113
2	0.149628	0.142567	0.136815	0.045421	0.074406	0.264064	0.234681	0.245626
3	0.146222	0.139123	0.121845	0.045008	0.076374	0.271467	0.180557	0.196586
4	0.141076	0.134383	0.128697	0.042940	0.068370	0.264378	0.217043	0.23221
5	0.130860	0.127477	0.123132	0.029561	0.075777	0.195866	0.195671	0.204883
6	0.134913	0.117200	0.097031	0.0668226	0.017606	0.262077	0.183936	0.191137
7	0.210946	0.201060	0.198503	0.063823	0.103589	0.349300	0.201512	0.212944
8	0.142078	0.132387	0.115286	0.051574	0.061681	0.278849	0.176295	0.187578
9	0.162681	0.253560	0.147614	0.053708	0.037825	0.272716	0.178501	0.182827
10	0.147659	0.128816	0.113429	0.072177	0.031227	0.314443	0.211626	0.218735
Media	0.1512661	0.1519077	0.1323819	0.05054396	0.06091777	0.2691192	0.2052659	0.2152639

Tabla 8. Análisis matemático de los datos para la secuencia **'freiburg3_nostructure_texture_near'.**

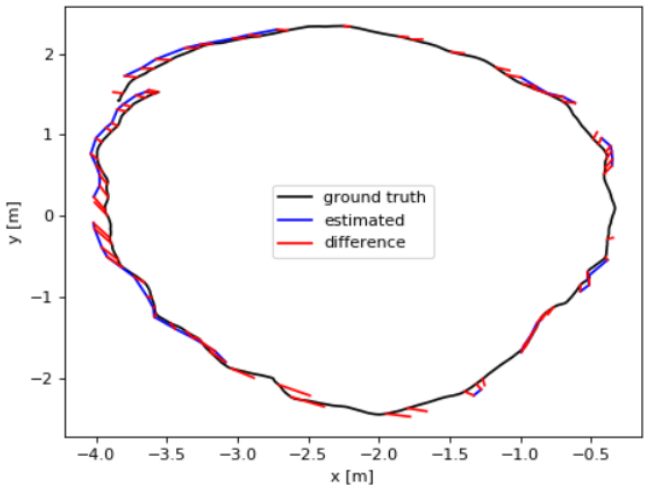


Figura 43. Trayectoria real vs trayectoria estimada de la secuencia **'freiburg3_nostructure_texture_near'.**

- Análisis de la ejecución de la trayectoria 'freiburg3_structure_texture_near'.

EVALUACIÓN DE LAS PRUEBAS DE LA TRAYECTORIA 'freiburg3_structure_texture_near'								
Número de Prueba	RMSE	Error Medio	Mediana de Error	Error Estándar	Error Mínimo	Error Máximo	Median Tracking Time	Mean Tracking Time
1	0.024119	0.022053	0.020600	0.009766	0.007227	0.051356	0.174142	0.176383
2	0.025662	0.023954	0.021621	0.009204	0.009817	0.056196	0.175763	0.192685
3	0.024517	0.022467	0.023381	0.009812	0.007477	0.053203	0.1818	0.191146
4	0.027179	0.02675	0.021249	0.013348	0.007295	0.062924	0.161004	0.164016
5	0.019409	0.017615	0.016736	0.008149	0.001373	0.039403	0.178666	0.18509
6	0.039813	0.033005	0.026462	0.022265	0.004765	0.108094	0.189884	0.199461
7	0.025439	0.023531	0.023109	0.009667	0.008358	0.054746	0.217106	0.228497
8	0.029296	0.026919	0.024694	0.011560	0.008501	0.052477	0.183095	0.195631
9	0.029921	0.026966	0.024386	0.012965	0.009522	0.075396	0.177511	0.188778
10	0.022113	0.021208	0.021213	0.006263	0.010262	0.034619	0.205949	0.218768
Media	0.0267468	0.0244468	0.0223451	0.0112999	0.0074597	0.0588414	0.184492	0.1940455

Tabla 9. Análisis matemático de los datos para la secuencia **'freiburg3_structure_texture_near'.**

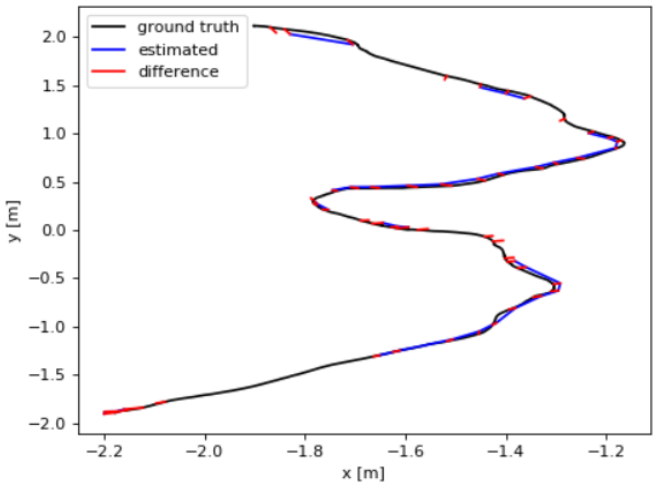


Figura 44. Trayectoria real vs trayectoria estimada de la secuencia **'freiburg3_structure_texture_near'.**

- Análisis de la ejecución de la trayectoria 'freiburg3_sitting_xyz'.

EVALUACIÓN DE LAS PRUEBAS DE LA TRAYECTORIA 'freiburg3_sitting_xyz'								
Número de Prueba	RMSE	Error Medio	Mediana de Error	Error Estándar	Error Mínimo	Error Máximo	Median Tracking Time	Mean Tracking Time
1	0.016522	0.014474	0.013836	0.007968	0.004456	0.044077	0.25134	0.25888
2	0.021258	0.019813	0.020413	0.007703	0.000695	0.038201	0.18535	0.171344
3	0.019926	0.017708	0.016414	0.009135	0.005149	0.047856	0.197071	0.188465
4	0.021965	0.020093	0.019841	0.008873	0.005503	0.040407	0.209572	0.212575
5	0.015318	0.014623	0.015517	0.004561	0.004766	0.022606	0.278107	0.258967
6	0.019038	0.017643	0.018693	0.007155	0.005751	0.027753	0.257644	0.246604
7	0.023931	0.022210	0.022149	0.008911	0.005478	0.042104	0.198564	0.187447
8	0.026061	0.025057	0.025600	0.007165	0.005683	0.035023	0.192706	0.177478
9	0.017758	0.016273	0.015335	0.007110	0.004650	0.031664	0.208736	0.197136
10	0.017366	0.016172	0.014723	0.006328	0.004605	0.029092	0.186943	0.175122
Media	0.0199143	0.0184066	0.0182521	0.0074909	0.0046736	0.0358783	0.2166033	0.2074018

Tabla 10. Análisis matemático de los datos para la secuencia **'freiburg3_sitting_xyz'.**

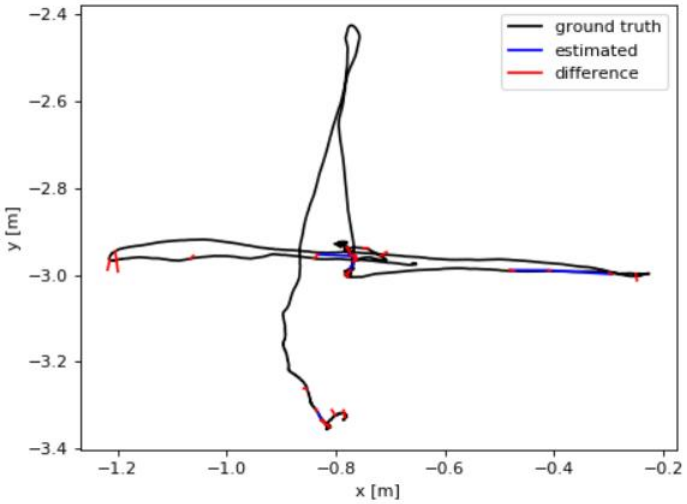


Figura 45. Trayectoria real vs trayectoria estimada de la secuencia **'freiburg3_sitting_xyz'.**

- Análisis de la ejecución de la trayectoria 'freiburg3_sitting_halfsphere'.

EVALUACIÓN DE LAS PRUEBAS DE LA TRAYECTORIA 'freiburg3_sitting_halfsphere'								
Número de Prueba	RMSE	Error Medio	Mediana de Error	Error Estándar	Error Mínimo	Error Máximo	Median Tracking Time	Mean Tracking Time
1	0.028049	0.024095	0.019604	0.014360	0.003480	0.061268	0.208951	0.209075
2	0.016616	0.015119	0.014551	0.006893	0.002879	0.040368	0.174604	0.174248
3	0.022248	0.020171	0.019371	0.009385	0.003672	0.041984	0.195706	0.188642
4	0.020479	0.018723	0.017186	0.008297	0.004421	0.051873	0.211839	0.216743
5	0.022775	0.018949	0.015386	0.012635	0.006030	0.083583	0.190874	0.196413
6	0.022872	0.019909	0.016576	0.011260	0.002741	0.050768	0.207287	0.205946
7	0.033211	0.029758	0.031469	0.014744	0.007608	0.081518	0.206067	0.203944
8	0.052580	0.04297	0.036898	0.028328	0.009258	0.129303	0.199659	0.18878
9	0.020249	0.017324	0.015399	0.010484	0.002182	0.069184	0.186551	0.1894355
10	0.026806	0.0124616	0.021728	0.010611	0.007750	0.060453	0.207414	0.214009
Media	0.0265885	0.02194796	0.02194796	0.0126997	0.0050021	0.0670302	0.1988952	0.19872355

Tabla 11. Análisis matemático de los datos para la secuencia **'freiburg3_sitting_halfsphere'.**

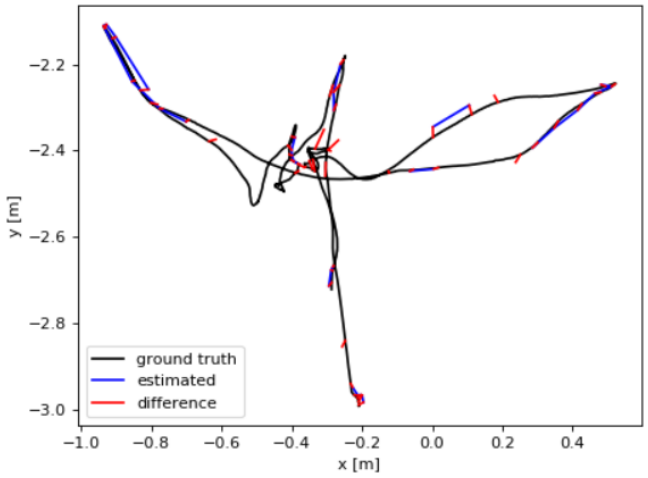


Figura 46. Trayectoria real vs trayectoria estimada de la secuencia **'freiburg3_sitting_halfsphere'.**

- Análisis de la ejecución de la trayectoria 'freiburg3_walking_xyz'.

EVALUACIÓN DE LAS PRUEBAS DE LA TRAYECTORIA 'freiburg3_walking_xyz'								
Número de Prueba	RMSE	Error Medio	Mediana de Error	Error Estándar	Error Mínimo	Error Máximo	Median Tracking Time	Mean Tracking Time
1	0.022945	0.020519	0.018487	0.010269	0.005905	0.045158	0.196837	0.192161
2	0.024714	0.021443	0.016392	0.012288	0.007977	0.046373	0.119235	0.126744
3	0.014950	0.013698	0.013157	0.005988	0.003682	0.024591	0.141796	0.160989
4	0.014736	0.013304	0.012046	0.006335	0.001731	0.026654	0.122522	0.140312
5	0.010932	0.009584	0.007607	0.005260	0.003695	0.021588	0.126892	0.143855
6	0.021159	0.020012	0.020434	0.006872	0.003029	0.033194	0.119012	0.116853
7	0.011999	0.011324	0.010945	0.003968	0.004122	0.017843	0.141584	0.170304
8	0.019020	0.016501	0.017473	0.009460	0.002715	0.046167	0.130326	0.150388
9	0.013044	0.011634	0.010226	0.005899	0.003958	0.024938	0.125507	0.140841
10	0.053610	0.035088	0.024830	0.040533	0.013919	0.189251	0.128529	0.148332
Media	0.0207109	0.0173107	0.0151597	0.0106872	0.0050733	0.0475757	0.135224	0.1490779

Tabla 12. Análisis matemático de los datos para la secuencia **'freiburg3_walking_xyz'.**

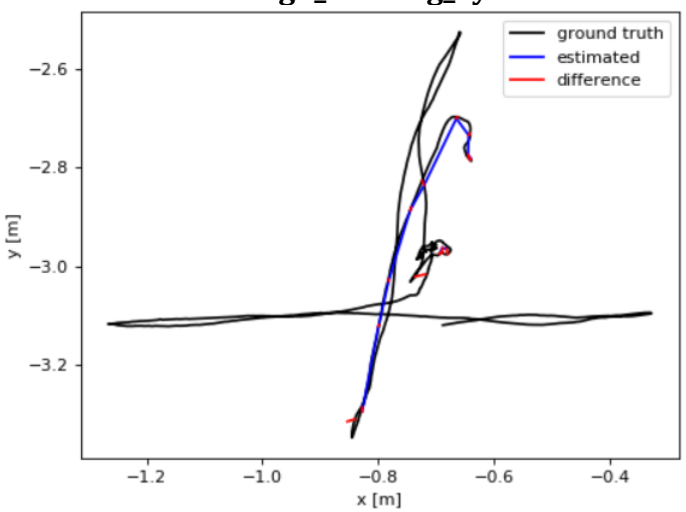


Figura 47. Trayectoria real vs trayectoria estimada de la secuencia **'freiburg3_walking_xyz'.**

- Análisis de la ejecución de la trayectoria 'freiburg3_walking_halfsphere'.

EVALUACIÓN DE LAS PRUEBAS DE LA TRAYECTORIA 'freiburg3_walking_halfsphere'								
Número de Prueba	RMSE	Error Medio	Mediana de Error	Error Estándar	Error Mínimo	Error Máximo	Median Tracking Time	Mean Tracking Time
1	0.030361	0.026932	0.024688	0.014016	0.003556	0.058673	0.193282	0.190087
2	0.050287	0.044673	0.045825	0.0230457	0.010806	0.0891192	0.200419	0.205076
3	0.045015	0.041071	0.043379	0.018428	0.009349	0.086237	0.180658	0.179032
4	0.050125	0.037927	0.031400	0.032772	0.013417	0.209148	0.243486	0.235552
5	0.039038	0.035196	0.033776	0.016888	0.007826	0.074695	0.185717	0.177588
6	0.470752	0.410061	0.401344	0.231209	0.087474	0.930042	0.198444	0.1990454
7	0.032739	0.027818	0.023879	0.017263	0.004980	0.114625	0.20413	0.20334
8	0.039661	0.036379	0.034964	0.015798	0.010175	0.070264	0.185532	0.186621
9	0.029637	0.027413	0.026330	0.011265	0.008678	0.063839	0.193184	0.190308
10	0.032021	0.028256	0.023882	0.015065	0.005381	0.077725	0.187144	0.193356
Media	0.0819636	0.0715726	0.0689467	0.03957497	0.0161642	0.17743672	0.1971996	0.19600054

Tabla 13. Análisis matemático de los datos para la secuencia **'freiburg3_walking_halfsphere'.**

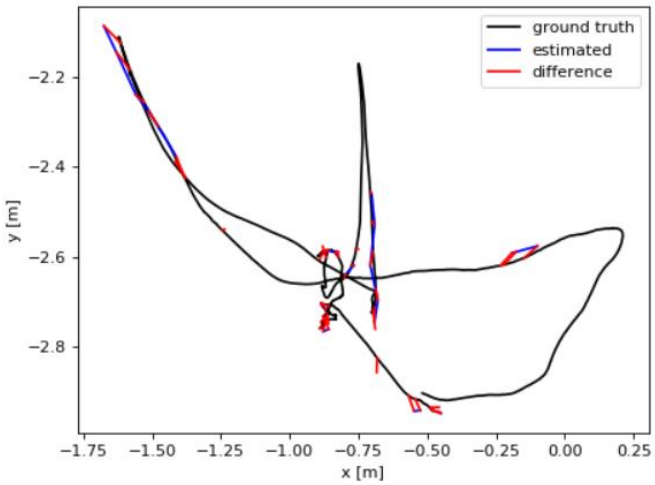


Figura .48 Trayectoria real vs trayectoria estimada de la secuencia **'freiburg3_walking_halfsphere'.**

PROMEDIO DE LOS INDICADORES DE DESEMPEÑO PARA TODAS LAS TRAYECTORIAS OBJETO DE ANÁLISIS.

Secuencia TUM RGB-D	PROMEDIO DE LOS INDICADORES DE DESEMPEÑO							
	RMSE	Error Medio	Mediana del Error	Error Estándar	Error Mínimo	Error Máximo	Median Tracking Time	Mean Tracking Time
f1_xyz	0.0094588	0.0081533	0.0074297	0.0047506	0.0015508	0.0212112	0.1787609	0.1751403
f2_xyz	0.0028902	0.0026358	0.0024931	0.0011816	0.0006507	0.0052374	0.2220995	0.2246417
f3_long_office	0.0590727	0.0492972	0.0441114	0.0475287	0.01279906	0.1963556	0.195512	0.19848731
f3_nstr_tex_near	0.1512661	0.1519077	0.1323819	0.05054396	0.06091777	0.2691192	0.2052659	0.2152639
f3_str_tex_near	0.0267468	0.0244468	0.0223451	0.0112999	0.0074597	0.0588414	0.184492	0.1940455
f2_desk_person	0.0277862	0.0262602	0.0258044	0.0088079	0.0093048	0.04960	0.1998169	0.2078379
f3_sit_xyz	0.0199143	0.0184066	0.0182521	0.0074909	0.0046736	0.0358783	0.2166033	0.2074018
f3_sit_halfsph	0.0265885	0.02194796	0.02194796	0.0126997	0.0050021	0.0670302	0.1988952	0.19872355
f3_walk_xyz	0.0207109	0.0173107	0.0151597	0.0106872	0.0050733	0.0475757	0.135224	0.1490779
f3_walk_halfsphe	0.0819636	0.0715726	0.0689467	0.03957497	0.0161642	0.17743672	0.1971996	0.19600054

TABLA 14. Promedio de los indicadores de desempeño para todas las trayectorias objeto de análisis.

5.2. COMPARACIÓN DE LA PRECISIÓN CON OTROS SISTEMAS SLAM Y DETERMINACIÓN DE LA VELOCIDAD DEL SISTEMA

Con fines de observar si el rendimiento del sistema es aceptable en lo que concierne a la velocidad y a la precisión, se decidió inicialmente comparar su precisión con otros cuatro sistemas y posteriormente determinar su velocidad en materia de tasa de fotogramas por segundo.

- **Comparación de la precisión con otros algoritmos SLAM.**
  Los datos sobre la precisión de ORB-SLAM, PTAM, LSD-SLAM y RGBD-SLAM para cada trayectoria fueron extraídos de [37].

Secuencia TUM RGB-D	Raíz del Error Cuadrático Medio (cm)				
	PL-SLAM *Nuestro*	ORB-SLAM	PTAM	LSD-SLAM	RGBD-SLAM
f1_xyz	**0.94588**	1.38	1.15	9.00	1.34
f2_xyz	0.28902	0.54	**0.2**	2.15	2.61
f3_long_office	5.90727	**4.05**	-	38.53	-
f3_nstr_tex_near	15.12661	2.88	**2.74**	7.54	-
f3_str_tex_near	2.67468	1.5451	**1.04**	-	-
f2_desk_person	**2.77862**	5.95	-	31.73	6.97
f3_sit_xyz	1.99143	**0.08**	0.83	7.73	-
f3_sit_halfsph	2.65885	**1.48**	-	5.87	-
f3_walk_xyz	2.07109	**1.64**	-	12.44	-
f3_walk_halfsphe	8.19636	**2.09**	-	-	-

Tabla 15. Comparación de la precisión de nuestro sistema con otros 4 sistemas SLAM.

- **Determinación de la velocidad del sistema**

Secuencia TUM RGB-D	**Median Tracking Time (s)**	**Mean Tracking Time (s)**
f1_xyz	0.1787609	0.1751403
f2_xyz	0.2220995	0.2246417
f3_long_office	0.195512	0.19848731
f3_nstr_tex_near	0.2052659	0.2152639
f3_str_tex_near	0.184492	0.1940455
f2_desk_person	0.1998169	0.2078379
f3_sit_xyz	0.2166033	0.2074018
f3_sit_halfsph	0.1988952	0.19872355
f3_walk_xyz	0.135224	0.1490779
f3_walk_halfsphe	0.1971996	0.19600054
Promedio	**0.19338693**	**0.19666204**

Tabla 16. Determinación de los promedios de la mediana del tiempo de seguimiento y de la media del tiempo de seguimiento.

El sistema de puntos y líneas bajo las condiciones actuales procesa a una velocidad de **5.17098fps.**

6. CONCLUSIONES

Nuestro sistema de puntos y líneas, logró ejecutar todas las secuencias en las que fue probado, lo que no lograron los algoritmos PTAM, LSD-SLAM y RGBD-SLAM.

En general, nuestro sistema presentó una buena precisión; superando en 3 de las 10 trayectorias en análisis al sistema ORB-SLAM; logrando ejecutar más secuencias que PTAM y superando la precisión de dicho algoritmo en 1 de las 5 secuencias que este logró ejecutar; alcanzando también un mejor rendimiento en materia de precisión que LSD-SLAM en 6 de las 8 secuencias que se pueden analizar; y finalmente, superando en precisión a RGBD-SLAM en 2 de las 3 trayectorias que pueden ser objeto de comparación con éste algoritmo.

El sistema bajo las condiciones actuales presenta una tasa de fotogramas de **5.17098fps,** tasa que no cumple con las condiciones de Visual SLAM en tiempo real, y que comparado con la tasa de fotogramas de otras ejecuciones de sistemas de puntos y líneas (20 fps en [37] y 30 fps en [11]) resulta muy bajo.

Como se mencionó en el capítulo 3, el algoritmo PL-SLAM fue probado en versiones de Ubuntu 12.04, 14.04 y 16.04. Pero la tarjeta NVIDIA Jetson Nano, por lo menos hasta la fecha de publicación de esta monografía, sólo es compatible con la versión 18.04 de Ubuntu[66]. Al darse esa situación, muchas librerías y dependencias utilizadas en las anteriores versiones se han visto obsoletas o innecesarias para la versión compatible con la Jetson Nano. Para corroborar si era posible la instalación del sistema en la versión 18.04 de Ubuntu, se ejecutó el algoritmo en una versión de Ubuntu 18.04 diferente a la que está adaptada para la NVIDIA Jetson Nano, encontrando que sí era posible su ejecución como lo demuestran los resultados presentados en este documento, pero también encontrando como respuesta que el algoritmo en cuestión no se puede compilar en la NVIDIA Jetson Nano debido a sus diferencias en la arquitectura del sistema, pues este algoritmo no está adaptado para equipos con sistemas operativos con arquitecturas ARM.

BIBLIOGRAFÍA

[1] P. López Torres, "Análisis De Algoritmos Para Localización Y Mapeado Simultáneo De Objetos," Universidad de Sevilla, 2016. [Online]. Available: https://idus.us.es/xmlui/bitstream/handle/11441/54393/TFM_PatriciaLópez Torres.pdf?sequence=1

[2] M. W. M. Gamini Dissanayake, P. Newman, S. Clark, H. F. Durrant-Whyte, and M. Csorba, "A Solution To The Simultaneous Localization And Map Building (SLAM) Problem," *IEEE Trans. Robot. Autom.*, vol. 17, no. 3, pp. 229–241, 2001, doi: 10.1109/70.938381.

[3] C. Hertzberg, R. Wagner, O. Birbach, T. Hammer, and U. Frese, "Experiences in building a visual SLAM system from open source components," in *Proceedings - IEEE International Conference on Robotics and Automation*, 2011, pp. 2644–2651. doi: 10.1109/ICRA.2011.5980140.

[4] G. A. Acosta Amaya, "SLAM Monocular en Tiempo Real," Universidad Nacional de Colombia, 2019. [Online]. Available: https://idus.us.es/bitstream/handle/11441/93944/TFG-2243-MACEDA GARCIA.pdf?sequence=1&isAllowed=y

[5] M. Himmelsbach, "LIDAR-based 3D Object Perception," *Proc. 1st Int. Work. Cogn. Tech. Syst.*, no. October 2008, 2008, [Online]. Available: http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:LIDAR-based+3D+Object+Perception#0

[6] H. N. E. The and R. Socie-, "Сравнение современных лазерных алгоритмов SLAM," 2001.

[7] D. P. Perez, E. S. Gomez, and M. M. Quintas, "Simultaneous localization and structure reconstruction of mobile robots with external cameras," *IEEE Int. Symp. Ind. Electron.*, vol. III, no. 1, pp. 1321–1326, 2005, doi: 10.1109/ISIE.2005.1529116.

[8] T. Peng, D. Zhang, D. L. Nirmal Hettiarachchi, and J. Loomis, "An Evaluation Of Embedded GPU Systems For Visual SLAM Algorithms," *Electron. Imaging*, pp. 1–6, 2020, doi: 10.2352/issn.2470-1173.2020.6.iriacv-325.

[9] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: An Efficient Alternative to SIFT or SURF," 2011. doi: 10.1088/1742-5468/2008/02/L02001.

[10] R. Mur-Artal and J. D. Tardós, "ORB-SLAM: Tracking and Mapping Recognizable Features," *Work. Multi VIew Geom. Robot. - RSS 2014*, no. July, 2014, [Online]. Available: http://vindelman.technion.ac.il/events/mvigro/MurArtal14rss_ws.pdf

[11] X. Wei, J. Huang, and X. Ma, “Real-Time Monocular Visual SLAM By Combining Points And Lines,” *Proc. - IEEE Int. Conf. Multimed. Expo*, vol. 2019-July, pp. 103–108, 2019, doi: 10.1109/ICME.2019.00026.

[12] J. Kurzak, “Preliminary results of autotuning GEMM kernels for the NVIDIA Kepler architecture-GeForce GTX 680,” vol. 100, no. April 2010, pp. 1–12, 2014, [Online]. Available: http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Preliminary+Results+of+Autotuning+GEMM+Kernels+for+the+NVIDIA+Kepler+Architecture#0%5Cnhttp://epubs.siam.org/doi/abs/10.1137/120863691%5Cnhttp://escholarship.org/uc/item/03f1h34s.pdf

[13] L. Jian, C. Wang, Y. Liu, S. Liang, W. Yi, and Y. Shi, “Parallel data mining techniques on Graphics Processing Unit with Compute Unified Device Architecture (CUDA),” *J. Supercomput.*, vol. 64, no. 3, pp. 942–967, 2013, doi: 10.1007/s11227-011-0672-7.

[14] M. W. M. G. Dissanayake, P. Newman, H. F. Durrant-Whyte, S. Clark, and M. Csorba, “An experimental and theoretical investigation into simultaneous localisation and map building,” *Exp. Robot. VI*, pp. 265–274, 2006, doi: 10.1007/bfb0119405.

[15] G. Dissanayake, H. Durrant-whyte, and T. Bailey, “A Computationally Efficient Solution to the Simultaneous Localization and Map Building (SLAM) Problem,” *Proc. 2000 IEEE*, no. April 2000, 2006.

[16] D. Bourque, “CUDA-Accelerated Visual SLAM For UAVs,” Worcester Polytechnic Institute, 2017. [Online]. Available: https://web.wpi.edu/Pubs/ETD/Available/etd-060117-025609/unrestricted/dbourque.pdf

[17] P. Ardón, K. Kushibar, and S. Peng, “A Hybrid SLAM And Object Recognition System For Pepper Robot,” 2019, [Online]. Available: http://arxiv.org/abs/1903.00675

[18] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, “ORB-SLAM: A Versatile And Accurate Monocular SLAM System,” *IEEE Trans. Robot.*, vol. 31, no. 5, pp. 1147–1163, 2015, doi: 10.1109/TRO.2015.2463671.

[19] P. Lopez Torres, “Análisis De Algoritmos Para Localización Y Mapeado Simultáneo De Objetos,” Universidad de Sevilla, 2016. [Online]. Available: https://idus.us.es/xmlui/handle/11441/54393

[20] R. Mur-Artal and J. D. Tardos, “ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras,” *IEEE Trans. Robot.*, vol. 6, no. 2, pp. 16–25, 2017, doi: 10.1017/S1042771600010292.

[21] J. Zhang, Y. Yao, and B. Deng, “Fast and Robust Iterative Closest Point,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 8828, no. c, pp. 1–16, 2021, doi: 10.1109/TPAMI.2021.3054619.

[22] L. M. Paz, P. Piniés, J. D. Tardós, and J. Neira, “Large Scale 6 DOF SLAM With Stereo In Hand,” *IEEE Trans. Robot.*, vol. 53, no. 7, pp. 3–7, 2008.

[23] S. T. Barnard and W. B. Thompson, “Disparity Analysis of Images,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-2, no. 4, pp. 333–340, 1980, doi: 10.1109/TPAMI.1980.4767032.

[24] L. Dreschler and H. H. Nagel, “Volumetric Model and 3D Trajectory Of A Moving Car Derived From Monocular TV-Frame Sequence Of A Street Scene,” *Comput. Graph. Image Process.*, vol. 20, no. 3, pp. 199–228, 1982, doi: 10.1016/0146-664X(82)90081-8.

[25] H. H. Nagel, “On The Estimation Of Optical Flow: Relations Between Different Approaches And Some New Results,” *Artif. Intell.*, vol. 33, no. 3, pp. 299–324, 1987, doi: 10.1016/0004-3702(87)90041-5.

[26] C. Harris, “Structure From Motion Under Orthographic Projection,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 427 LNCS, pp. 118–123, 1990, doi: 10.1007/BFb0014857.

[27] C. Tomasi and T. Kanade, “Shape And Motion From Image Streams: A Factorization Method,” *Int. J. Comput. Vis.*, vol. 9, no. 7597, pp. 137–154, 1992, [Online]. Available: http://ecommons.cornell.edu/handle/1813/7110

[28] R. J. Blissett, “Retrieving 3D Information From Video For Robot Control And Surveillance Blissett,” *Electron. Commun. Eng. J.*, vol. 2, no. 4, pp. 155–164, 1990, doi: 10.1049/ecej:19900037.

[29] P. Rosa, O. Silveira, J. De Melo, L. Moreira, and L. Rodrigues, “Development of Embedded Algorithm for Visual Simultaneous Localization and Mapping,” pp. 160–163, 2019, doi: 10.5753/sibgrapi.est.2019.8319.

[30] M. Trajković and M. Hedley, “Fast Corner Detection,” *Image Vis. Comput.*, vol. 16, no. 2, pp. 75–87, 1998, doi: 10.1016/s0262-8856(97)00056-5.

[31] S. M. Smith and J. M. Brady, “SUSAN - A new approach to low level image processing,” *Int. J. Comput. Vis.*, vol. 23, no. 1, pp. 45–78, 1997, doi: 10.1023/A:1007963824710.

[32] C. Akinlar and C. Topal, “EDLINES : REAL-TIME LINE SEGMENT DETECTION BY EDGE DRAWING (ED),” *Ieee Int. Conf. Image Process.*, no. 2, pp. 2897–2900, 2011.

[33] C. Topal, C. Akinlar, and Y. Genç, “Edge drawing: A heuristic approach to robust real-time edge detection,” *Proc. - Int. Conf. Pattern Recognit.*, pp. 2424–2427, 2010, doi: 10.1109/ICPR.2010.593.

[34] I. Sobel, “Camera models and machine perception,” 1970.

[35] C. Topal, O. Ozsen, and C. Akinlar, “Real-Time Edge Segment Detection With Edge Drawing Algorithm,” *ISPA 2011 - 7th Int. Symp. Image Signal Process. Anal.*,

no. January, pp. 313–318, 2011.

[36] A. Bartoli *et al.*, “Structure-From-Motion Using Lines : Representation , Triangulation and Bundle Adjustment To cite this version : HAL Id : hal-00092589 Structure-From-Motion Using Lines : Representation , Triangulation and Bundle Adjustment,” 2006.

[37] A. Pumarola *et al.*, “PL-SLAM: Real-Time Monocular Visual SLAM with Points and Lines Albert,” *Proc. - IEEE Int. Conf. Robot. Autom.*, vol. 8, no. 2, pp. 1–17, 2020, [Online]. Available: http://arxiv.org/abs/2009.09972

[38] M. Kotlar, D. Bojic, M. Punt, and V. Milutinovic, “A Survey of Deep Neural Networks: Deployment Location and Underlying Hardware,” *2018 14th Symp. Neural Networks Appl. NEUREL 2018*, pp. 1–6, 2018, doi: 10.1109/NEUREL.2018.8587006.

[39] K. Skala and Z. Sojat, “Cloud, Fog and Dew Computing: A Distributed Hierarchy”.

[40] S. Valladares, M. Toscano, R. Tufiño, P. Morillo, and D. Vallejo-Huanga, “Performance Evaluation of the Nvidia Jetson Nano Through a Real-Time Machine Learning Application,” no. January, pp. 343–349, 2021, doi: 10.1007/978-3-030-68017-6_51.

[41] A. A. Suzen, B. Duman, and B. Sen, “Benchmark Analysis of Jetson TX2, Jetson Nano and Raspberry PI using Deep-CNN,” *HORA 2020 - 2nd Int. Congr. Human-Computer Interact. Optim. Robot. Appl. Proc.*, pp. 3–7, 2020, doi: 10.1109/HORA49412.2020.9152915.

[42] S. Mittal, “A Survey On Optimized Implementation Of Deep Learning Models On The NVIDIA Jetson Platform,” *J. Syst. Archit.*, vol. 97, pp. 428–442, 2019, doi: 10.1016/j.sysarc.2019.01.011.

[43] S. Zotov, “Stand-Alone Orientation System Based On Visual SLAM,” *2020 IEEE/ION Position, Locat. Navig. Symp. PLANS 2020*, pp. 226–233, 2020, doi: 10.1109/PLANS46316.2020.9109974.

[44] “ORB-SLAM & Nvidia - Jetson & Embedded Systems / Jetson AGX Xavier - NVIDIA Developer Forums.” https://forums.developer.nvidia.com/t/orb-slam-nvidia/159642 (accessed May 09, 2022).

[45] T. Peng, D. Zhang, R. Liu, V. K. Asari, and J. S. Loomis, “Evaluating the Power Efficiency of Visual SLAM on Embedded GPU Systems,” *Proc. IEEE Natl. Aerosp. Electron. Conf. NAECON*, vol. 2019-July, pp. 117–121, 2019, doi: 10.1109/NAECON46414.2019.9058059.

[46] O. C. B. Silveira, J. G. O. C. De Melo, L. A. S. Moreira, J. B. N. G. Pinto, L. R. L. Rodrigues, and P. F. F. Rosa, “Evaluating a Visual Simultaneous Localization and Mapping Solution on Embedded Platforms,” *IEEE Int. Symp. Ind. Electron.*, vol. 2020-June, pp. 530–535, 2020, doi: 10.1109/ISIE45063.2020.9152370.

[47] C. Perez and F. Piccoli, “ESTIMACIÓN DE LOS PARÁMETROS DE RENDIMIENTO

DE UNA GPU," vol. XXIX, pp. 15–18, 2010.

[48] M. F. Piccoli, *Computación de Alto Desempeño en GPU*. 2011.

[49] M. Joselli *et al.*, "Automatic dynamic task distribution between CPU and GPU for real-time systems," *Proc. - 2008 IEEE 11th Int. Conf. Comput. Sci. Eng. CSE 2008*, pp. 48–55, 2008, doi: 10.1109/CSE.2008.38.

[50] W. N. Chen and H. M. Hang, "H.264/AVC MOTION ESTIMATION IMPLEMENTATION ON COMPUTE UNIFIED DEVICE ARCHITECTURE (CUDA)," *2008 IEEE Int. Conf. Multimed. Expo, ICME 2008 - Proc.*, pp. 697–700, 2008, doi: 10.1109/ICME.2008.4607530.

[51] "GeForce GTX 16 Series Graphics Cards | NVIDIA." https://www.nvidia.com/es-la/geforce/graphics-cards/gtx-1650/ (accessed May 10, 2022).

[52] C. JIAGANG, "PL-SLAM," 2020. https://github.com/HarborC/PL-SLAM (accessed May 09, 2022).

[53] S. Lovegrove, "Pangolín," 2018. https://github.com/stevenlovegrove/Pangolin (accessed May 05, 2022).

[54] https://cmake.org/, "Documentación de referencia de CMake — Documentación de CMake." https://cmake.org/cmake/help/latest/ (accessed May 05, 2022).

[55] Https://www.gnu.org/, "El sistema operativo GNU y el movimiento del software libre." https://www.gnu.org/home.es.html (accessed May 05, 2022).

[56] https://gcc.gnu.org/, "GCC, la colección de compiladores GNU - Proyecto GNU." https://gcc.gnu.org/ (accessed May 05, 2022).

[57] "Acerca de - OpenCV." https://opencv.org/about/ (accessed May 05, 2022).

[58] G. Bradski and A. Kaehler, *Learning OpenCV: Computer Vision with the OpenCV Library*. United States of America: O'Reilly Media, Inc., 2008.

[59] "Eigen." https://eigen.tuxfamily.org/index.php?title=Main_Page (accessed May 05, 2022).

[60] D. Galvez López and J. D. Tardós, "Bags of Binary Words for Fast Place Recognition in Image Sequences," *IEEE Trans. Robot.*, vol. 28, no. 3, pp. 592–606, 2012, doi: 10.1109/TRO.2011.2179580.

[61] "Ceres Solver: una biblioteca de optimización no lineal a gran escala." http://ceres-solver.org/ (accessed May 05, 2022).

[62] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "g2o: A General Framework For Graph Optimization," *Proc. - IEEE Int. Conf. Robot. Autom.*, pp. 3607–3613, 2011, doi: 10.1109/ICRA.2011.5979949.

[63] "Computer Vision Group - Dataset Download." https://vision.in.tum.de/data/datasets/rgbd-dataset/download (accessed May

06, 2022).

[64] R. Walpole, R. Myrers, S. Myers, and K. Ye, *PROBABILIDAD Y ESTADÍSTICA PARA INGENIERÍA Y CIENCIAS*, vol. 59.

[65] L. O. Rodriguez, *Probabilidad Y Estadística Básica Para Ingenieros*. 1999. [Online]. Available: https://books.google.com.pe/books?id=9DWw696jLbMC&printsec=frontcover&dq=Probabilidad+y+estadística+para+ingenieros&hl=es-419&sa=X&ved=0ahUKEwjIpo3Zn-7hAhWHGrkGHb0nCZIQ6AEILjAB#v=onepage&q=Probabilidad y estadística para ingenieros&f=false%0Ahttp://onlineli

[66] "Jetson nano image with Ubuntu 16.04 - Jetson & Embedded Systems / Jetson Nano - NVIDIA Developer Forums." https://forums.developer.nvidia.com/t/jetson-nano-image-with-ubuntu-16-04/79152 (accessed May 10, 2022).