



Escuela
Politécnica
Superior

Reconstrucción del cuerpo humano utilizando datos RGB-D y GPUs



Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor:

Edgar Martínez Serrano

Tutor/es:

Marcelo Saval Calvo

Jorge Azorín López

Julio 2022

Reconstrucción del cuerpo humano utilizando datos RGB-D y GPUs

Implementación del proyecto en una computadora de bajo coste.

Autor

Edgar Martínez Serrano

Tutor/es

Marcelo Saval Calvo

Tecnología Informática y Computación

Jorge Azorín López

Tecnología Informática y Computación



Grado en Ingeniería Informática



Escuela
Politécnica
Superior



Universitat d'Alacant
Universidad de Alicante

ALICANTE, Julio 2022

Agradecimientos

En primer lugar me gustaría dar las gracias a Marcelo, mi tutor, que sin él nada de esto hubiera sido posible. Gracias por todos los conocimientos que me has impartido, por lo paciente que has sido conmigo tras varios periodos donde he estado ausente durante meses, gracias por haber estado siempre ahí y por sacar tiempo de donde no lo tienes para poder sacar adelante este trabajo.

También me gustaría agradecer a mi pareja, Sandra, por todo el ánimo y apoyo fundamental que me ha estado dando sobre todo estos últimos meses. Agradecer también a Javi, Miguel y Raúl, compañeros de universidad y trabajo, amigos que me han estado apoyando como los que más estas últimas semanas para que esto salga adelante.

Gracias a mis amigos de toda la vida, por seguir a mi lado después de meses sin interacción por mi parte. Me alegra estar escribiendo estas palabras sabiendo que mañana por fin voy a tener más tiempo libre para compartir con vosotros.

Por último, me gustaría agradecer a PCCOM, la empresa donde trabajo, por hacer posible que terminase a tiempo el trabajo siendo lo más flexibles posible con el horario para poder dedicarme al máximo en este proyecto. Gracias Sergio y Pablo, por la flexibilidad y por el apoyo recibido.

Preámbulo

Este TFG (Trabajo Final de Grado) consiste en desarrollar un sistema embebido de bajo coste y portable que, utilizando datos RGB-D (Red Green Blue – Depth) de color y profundidad, reconstruya un cuerpo humano completo. El sistema tomará capturas del cuerpo desde un punto de vista fijo mientras el sujeto gira frente a la cámara. Con los datos obtenidos, se aplicarán técnicas de registro para alinear las distintas vistas y así formar el modelo completo del sujeto. Estos métodos, se acelerarán posteriormente con la GPU (Graphics Processing Unit) del sistema embebido para permitir una ejecución en línea del sistema.

Índice general

1. Introducción	1
1.1. Motivación y Contexto	1
1.1.1. Análisis previo del sistema	1
1.2. Estado del arte	2
1.2.1. Hardware para adquisición de datos	3
1.2.2. Hardware para el registro y análisis de los datos	5
1.2.3. Situación social actual, necesidades y aplicaciones	8
1.2.4. Investigaciones y otros proyectos ya realizados	9
1.3. Objetivos	9
1.4. Desarrollo de la memoria	10
2. Base teórica	11
2.1. Métodos de registro	11
2.1.1. Métodos de registro rígidos, ICP	13
2.1.2. Métodos de registro no rígidos, CPD	16
2.2. Matriz de transformación 3D	16
2.3. Filtros para el pre-procesamiento y post-procesamiento de los datos	19
2.4. Aceleración por GPU, CUDA	21
3. Intel RealSense Depth Camera D435	25
4. Propuesta de solución	27
4.1. Adquisición de datos	27
4.2. Pre-procesado	28
4.3. Registro	29
4.3.1. Posible refinamiento con CPD	30
4.4. Post-procesado	31
4.5. Análisis	31
5. Solución del proyecto en Jetson Nano	33
5.1. Integración del Software. Dependencias	33
5.1.1. Sistema Operativo. Ubuntu 18.04	33
5.1.2. CMake. Compilador C++	33
5.1.3. SDK RealSense	34
5.1.4. Point Cloud Library (PCL)	34
5.1.5. CUDA. CUDA-ICP	35
5.1.6. JsonCpp	35
5.2. Desarrollo de la solución	36
5.2.1. Adquisición de datos	36

5.2.2. Pre-procesado	38
5.2.3. Registro	40
6. Experimentación	43
6.1. Herramientas para visualizar las nubes de puntos	43
6.2. Resultados entre un modelo y una escena	44
6.3. Resultados de la reconstrucción completa con un objeto	45
6.4. Resultados de la reconstrucción completa de un cuerpo humano	46
7. Conclusión	55
Anexos	57
A. Entorno experimental	59
B. Estructura del proyecto subido a GitHub	61
Bibliografía	63
Lista de Acrónimos y Abreviaturas	67

Índice de figuras

1.1. Esquema general de un sistema 3D	2
1.2. Sensor Intel RealSense D435.	4
1.3. Media de error en la comparación de sensores RGB-D.	5
1.4. Benchmarking Nvidia Jetson TX2, Nvidia Jetson Nano y Raspberry Pi 4 Model B.	7
2.1. Ejemplo de alineación de una escena con un modelo.	11
2.2. Diagrama tipos de registro.	12
2.3. Ejemplo de alineación de una escena usando RANSAC.	14
2.4. Diagrama del algoritmo ICP.	15
2.5. Ejemplo de captura de escena muy alejada al modelo.	15
2.6. Rotación de 15° de un modelo 3D en el eje X.	17
2.7. Ejemplo de matriz de transformación inicial.	19
2.8. Filtros de pre-procesamiento y post-procesamiento en un esquema general de un sistema 3D.	20
2.9. Comparación de arquitecturas CPU y GPU. Figura extraída de Stopper y Roth (2017).	22
2.10. Arquitectura GPU con tecnología CUDA. Figura extraída de Glaskowsky (2009). .	23
3.1. Sensores Intel RealSense D435.	25
3.2. Ejemplo de identificación de un punto común con tecnología estéreo activa. Figura extraída de Whyte (2022).	26
4.1. Diagrama de la propuesta e implementación de los módulos del sistema.	27
4.2. Ejemplo de filtro Statistical Outlier Removal aplicado.	29
4.3. Diagrama de reconstrucción 3D acumulando matrices de transformación para cada nueva iteración.	30
4.4. Ejemplo de modelo 3D en la fase de análisis.	31
5.1. Ejemplo de rotación en el pre-procesamiento de datos para dejar la nube de puntos alineada con el eje X.	39
6.1. Objeto de pruebas para reconstrucción 3D.	43
6.2. Captura de pantalla de Intel RealSense Viewer.	44
6.3. Captura de pantalla de MeshLab con 3 nubes de puntos.	44
6.4. Ejemplo de alineación de una escena con un modelo.	45
6.5. Resultados experimentación modelo y escena alineados.	47
6.6. Resultados de Reconstrucción 3D con objeto usando ICP.	48
6.7. Resultados de Reconstrucción 3D con objeto usando CUDA-ICP.	49
6.8. Resultados de Reconstrucción 3D de un cuerpo humano usando ICP.	51

6.9. Resultados de Reconstrucción 3D de un cuerpo humano usando CUDA-ICP.	52
A.1. Entorno experimental.	59
B.1. Estructura del proyecto en GitHub.	61

Lista de ecuaciones

2.1. Matrices de rotación básicas.	17
2.2. Vector y matriz de Traslación.	18
2.3. Matriz de Traslación aplicada a un punto.	18
2.4. Matriz de Transformación.	18

Índice de tablas

6.1. Resultados de Reconstrucción 3D con objeto.	50
6.2. Resultados de Reconstrucción 3D de un cuerpo humano.	53

Índice de Códigos

5.1.	Dependencia CMakeLists: RealSense Library	34
5.2.	Dependencia CMakeLists: Point Cloud Library	34
5.3.	Dependencia CMakeLists: CUDA	35
5.4.	Dependencia CMakeLists: CUDA-ICP	35
5.5.	Dependencia CMakeLists: JsonCpp	35
5.6.	Parámetros de la aplicación	36
5.7.	Inicialización del sensor RealSense D435 para la adquisición de datos	37
5.8.	Bucle para adquisición de los datos con el sensor RealSense D435	37
5.9.	Guardar las nubes de puntos en un fichero de datos	38
5.10.	Conversión de una nube de puntos de la librería RealSense a una nube de puntos de PCL	38
5.11.	Rotación en fase de pre-procesamiento de la nube de puntos	38
5.12.	Aplicación del Filtro PassThrough	39
5.13.	Aplicación del Statistical Outlier Removal	39
5.14.	Aplicación del Filtro Mediana	40
5.15.	Matriz de transformación inicial	40
5.16.	Preparación y llamada al algoritmo ICP de Point Cloud Library	40
5.17.	Resultado del algoritmo ICP. Matriz de Transformación y nube de puntos resultante	41
5.18.	Preparación y llamada al algoritmo de CUDA-ICP	41

1. Introducción

El primer capítulo introduce el tema principal del trabajo. Está organizado de la siguiente manera: la sección 1.1 habla del contexto, explicando la importancia de la investigación y la motivación de la misma; la sección 1.2 nos sitúa acerca del estado del arte actual sobre el tema en cuestión y la situación social actual, mostrando ejemplos de investigaciones y proyectos ya realizados; la sección 1.3 introduce el propósito del desarrollo con los objetivos y subobjetivos principales del proyecto; y por último, la sección 1.4 detalla la estructura que sigue el documento a continuación.

1.1. Motivación y Contexto

En tercero de carrera tuve que escoger mi primera asignatura optativa perteneciente a una de las especialidades del grado que marcaría el final de mis estudios. Toda mi vida me ha encantado la rama de IS (Ingeniería del Software), sin embargo, el área de ATC (Arquitectura y Tecnología de los Computadores) acabó atrayendo mi atención cada vez más. Finalmente realicé mi TFG en este área, donde abarqué temas de computación tales como la visión por computadores.

He realizado el proyecto e investigación en el DTIC (Departamento de Tecnología Informática y Computación) de ATC, que tiene experiencia en el campo de la visión por computador. Como he mencionado anteriormente, este campo me ha resultado cada vez más atractivo y he querido embarcarme en un proyecto donde poder explorar la rama de visión 3D (Tres Dimensiones).

Mi proyecto es parte de un proyecto de investigación mayor llamado Tech4Diet (2020) dirigido por mis tutores Marcelo Saval y Jorge Azorín. Tech4Diet es un proyecto que desarrolla un sistema para medir la evolución física del cuerpo humano haciendo uso de tecnologías de visión artificial. Para ello, el sistema utiliza 13 cámaras Intel D435 para capturar varias imágenes de un mismo cuerpo desde ángulos distintos para reconstruir el cuerpo en 3D. De esta forma se comparan diferentes muestras en 3D a lo largo del tiempo, obteniendo como resultado un análisis preciso de la evolución del cuerpo. El TFG pretende profundizar en estas técnicas de modelado 3D, reduciendo el número de cámaras a sólo una, en un sistema embebido económico que sea más asequible.

1.1.1. Análisis previo del sistema

En los últimos años los sistemas de visión artificial han evolucionado hacia una concepción mayormente multidimensional del entorno, principalmente la visión tridimensional. Debido a esto, se observa un crecimiento en el desarrollo de nuevas tecnologías para obtener y analizar escenas y objetos en 3D.

En los sistemas de visión 3D dedicados al modelado y análisis de escenas se pueden diferenciar tres fases: adquisición, registro y análisis, como se puede ver en la Figura 1.1

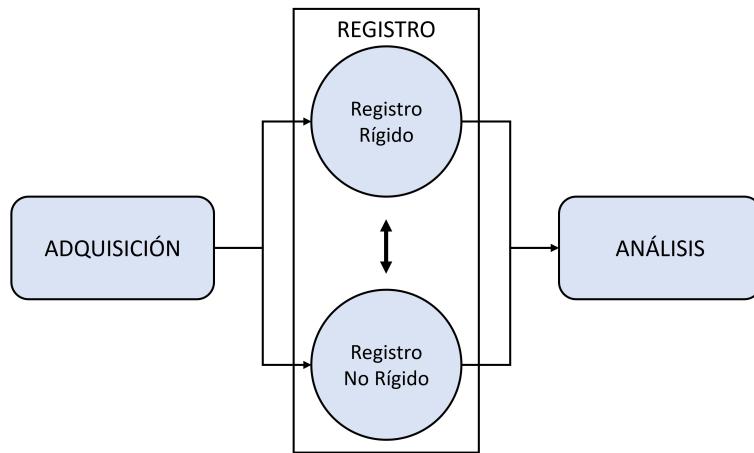


Figura 1.1: Esquema general de un sistema 3D.

La adquisición es la primera de las fases para obtener el análisis de una escena. En esta fase inicial es donde se toman los datos de entrada, por tanto, es la parte del proceso donde interviene el sensor utilizado. Esta es una parte crítica del sistema, debido a que el resultado de las fases posteriores depende por completo de los datos adquiridos.

Tras obtener los datos de entrada en la fase de adquisición, la fase del registro se encarga de realizar las transformaciones necesarias para obtener un modelo completo de la escena. Existen dos tipos principales de registro: registro rígido y registro no rígido. El registro rígido transforma todos los datos al unísono para llevarlos a un mismo sistema de coordenadas, el registro no rígido permite alinear los datos de forma independiente. Este último es útil para analizar cambios de forma o de color dependiendo del espacio que se esté analizando.

En la última fase, el análisis, se extrae la información necesaria de los datos obtenidos y procesados para cumplir el propósito del problema original. Este propósito será el que define los requerimientos para la fase de adquisición y de registro.

Por ejemplo, si la escena a analizar se trata de un objeto encima de una mesa, el proceso podría ser obtener los datos desde distintos puntos de vista como primer paso (fase de adquisición). Seguidamente registrar todas las vistas para que formen el modelo 3D completo (fase de registro). Finalmente, el análisis del modelo 3D nos permite obtener datos útiles de la reconstrucción, como la altura, el volumen, los colores, la forma, etc (fase de análisis).

Habitualmente se asume que los sensores están calibrados para mejorar la adquisición, y el registro de los datos 3D permita una mejor precisión y exactitud en la fase del análisis.

1.2. Estado del arte

Para abordar este proyecto construyendo un sistema embebido, portable y a la vez económico, se han explorado los diferentes dispositivos que existen en el mercado necesarios para abarcar las diferentes fases del sistema que pretendemos crear. Es decir, estudiaremos por una parte los diferentes sensores que hay en el mercado para la adquisición de los datos, y por otra parte el hardware necesario que se encargará del registro y del análisis de estos datos. Además, analizaremos la situación social actual para entender qué necesidades existen y la

utilidad que aporta un sistema como el que se pretende construir. Por último, se hará una pequeña investigación sobre proyectos similares para entender cuál es el estado actual de la tecnología y qué posibilidades ofrece.

1.2.1. Hardware para adquisición de datos

La fase de adquisición depende por completo del sensor utilizado para ello. Por tanto, es necesario analizar los distintos tipos de sensores para determinar qué sensor será utilizado para dicha finalidad.

Los sensores 3D pueden diferenciarse en las siguientes categorías: LIDAR (Light Detection And Ranging), ToF (Time of Flight), cámaras estereoscópicas y luz estructurada.

LIDAR: Los sensores LIDAR proyectan un láser en la escena y analizan la reflexión para determinar la distancia y obtener un mapa de profundidad.

ToF: Los sensores *Time of Flight* o “Tiempo de Vuelo” en español también proyectan un haz de luz, pero en este caso analizan el tiempo transcurrido entre la emisión y el retorno de dicho haz de luz para determinar la distancia.

A diferencia de los LIDAR, no utilizan un solo pulso de disparo, sino que emiten un cono de disparos, obteniendo información sobre todos los píxeles de la imagen. Los escenarios con objetos poco reflectantes pueden tener un impacto negativo en las mediciones realizadas con esta tecnología.

Luz estructurada: Los sensores de luz estructurada proyectan un patrón conocido en la escena y calculan la disparidad entre el patrón observado y el conocido. Dependiendo del tipo de cámara, puede utilizar patrones codificados temporal o espacialmente.

Las cámaras de luz estructurada con patrones codificados temporalmente utilizan más de un patrón para crear la imagen y diferenciar los elementos visuales. Sin embargo, este enfoque requiere sincronización entre el proyector de patrones y la cámara.

Por otro lado, las cámaras con patrones codificados espacialmente utilizan similitudes de vecindad espacial y no requieren patrones múltiples. No obstante, esta técnica puede presentar problemas en bordes o si hay oclusión del objeto en relación al proyector.

La iluminación del entorno, así como el motivo, pueden dificultar la identificación del patrón. Una vez identificado el patrón, se puede determinar la distancia entre el objeto y la cámara. Para ello, es necesario analizar las deformaciones que sufre el patrón sobre la superficie del objeto.

Cámaras estereoscópicas: Las cámaras estereoscópicas consisten en utilizar más de una cámara calibrada en diferentes posiciones para estimar la profundidad calculando la disparidad entre las imágenes obtenidas por cada cámara.

Se basan en una técnica que utiliza el desplazamiento espacial entre un par de imágenes para triangular y calcular la distancia real entre los objetos y la cámara. Muchos sistemas utilizan algoritmos de extracción de características y pueden presentar algunos problemas al intentar identificar puntos clave en escenarios con texturas deficientes.

A diferencia de los sensores mencionados anteriormente, estos son capaces de proporcionar información de color, pero es necesario calibrar ambos sensores cada vez que varíe su emplazamiento, por tanto, son sensores poco portables.

Los sensores RGB-D son sensores de propósito general y comúnmente conocidos por su “bajo coste”. Combinan las diferentes técnicas de los sensores anteriores con cámaras de color que permiten obtener de forma simultánea la profundidad y el color de la escena. A través de un sensor RGB-D podemos obtener un modelo 3D de un objeto a través de múltiples vistas rotando alrededor del mismo.

En la Figura 1.2 podemos ver un ejemplo de sensor RGB-D, la Intel RealSense D435. Este sensor utiliza cámaras estereoscópicas (“Right Imager” y “Left Imager”) acompañadas de un proyector IR (Radiación Infrarroja) que se encuentra en medio de estas. El proyector IR es una fuente de luz que sirve para iluminar la escena proyectando un patrón de puntos que ayuda a identificar puntos comunes entre las cámaras estereoscópicas. Estos tres componentes se encargan de capturar la profundidad y están acompañados de un módulo RGB (Red Green Blue) para poder capturar la información del color.



Figura 1.2: Sensor Intel RealSense D435.

Como hemos visto, existen diversas categorías de cámaras 3D con diferentes tecnologías para capturar objetos. En “Comparison of RGB-D sensors for 3D reconstruction” (da Silva Neto y cols., 2020) han realizado una comparación con 10 cámaras RGB-D de bajo coste distintas para concluir cuáles son las que presentan mejores resultados.

En la Figura 1.3 podemos ver como resultado la media de error obtenida para los 3 recorridos que se han hecho en cada cámara.

En general, el Intel D435 obtiene resultados aceptables en las rutas 1 y 3, que son las más similares al caso que abordaremos en este TFG.

La prueba donde más destaca el Intel RealSense D435 es en la ruta 3, Figura 1.3c, donde la cámara realiza acercamientos a los objetos. Esto indica que esta cámara trabaja mejor en escenarios donde la cámara se posiciona cerca del objeto o cuerpo a escanear, lo cual corresponde al objetivo de este TFG.

Además, la Intel RealSense D435 se trata del mismo sensor que se utiliza en el proyecto Tech4Diet (2020) del que se ha hablado en la sección 1.1 de Motivación y Contexto. Por tanto, este será el sensor a utilizar en este trabajo.

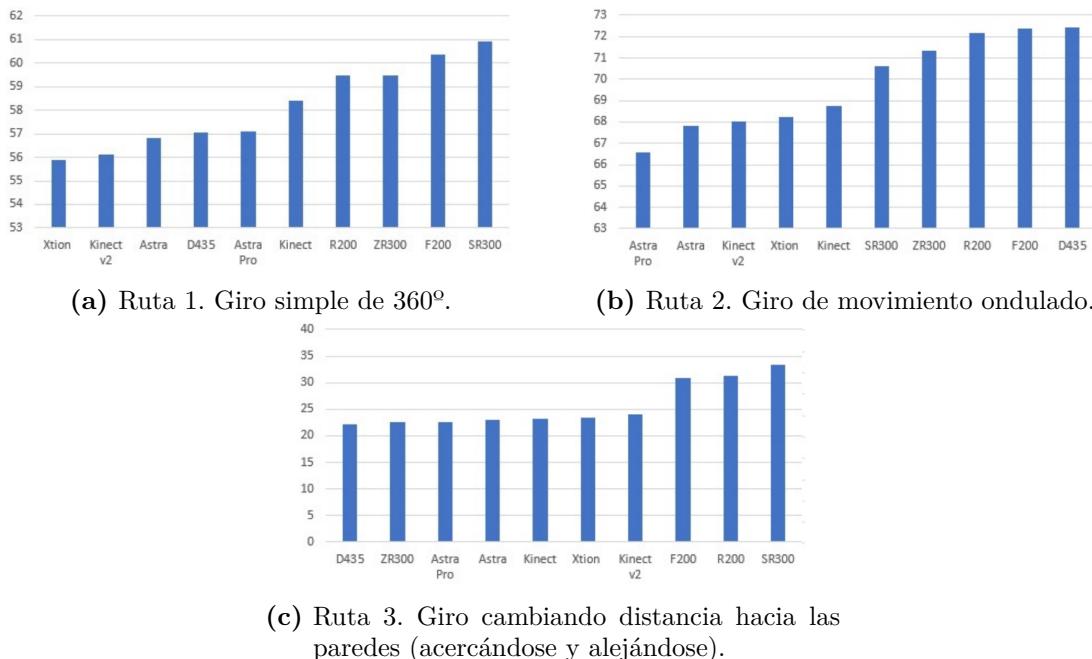


Figura 1.3: Media de error en la comparación de sensores RGB-D.

1.2.2. Hardware para el registro y análisis de los datos

Para cumplir con el apartado de registro y análisis de los datos necesitaremos hacer uso de un computador que nos permita: por una parte almacenar y realizar las modificaciones necesarias a los datos de entrada; y por otra parte analizar y extraer la información necesaria para el propósito del problema.

Uno de los requisitos que nos proponemos en este proyecto es la capacidad de realizar esto con un dispositivo que sea económico y a la vez portable. Para ello, se han explorado distintas opciones, como:

Raspberry Pi 4 Model B: se trata del último modelo de la serie Raspberry Pi. La Raspberry es un mini computador construido en un sistema embebido con un rendimiento comparable con el de un ordenador de sobremesa personal de gama baja.

Estos dispositivos se pueden utilizar como pequeños ordenadores debido a su gran potencial. También tiene la capacidad suficiente para usarse como servidor personal por ejemplo.

Comparado con la anterior generación (Raspberry Pi 3 Model B+) ofrece un aumento significativo en la velocidad del procesador, memoria y conectividad.

Cuenta con una CPU (Central Processing Unit) Quad-Core ARM Cortex-A72 de 64 bits a 1.5 GHz de velocidad. En cuanto a la GPU, cuenta con el chip Broadcom VideoCore VI, con una velocidad de 500 MHz. La capacidad de memoria RAM (Random Access Memory) es de 8 GB LPDDR4 y cuenta con todo tipo de conexiones (USB tipo C, USB Tipo B, microHDMI, conector para cámara...).

Con estas especificaciones consigue una potencia computacional de 13.5 GFlops. El precio de mercado de este dispositivo está alrededor de 35,00€¹.

Nvidia Jetson Nano: la Jetson Nano es la gama de entrada de la familia de Nvidia Jetson. Nvidia Jetson es una serie de computadores embebidos, parecidas a las Raspberries pero con GPUs más potentes. Esto nos permitirá realizar operaciones en paralelo en algoritmos para la visión por computador.

Cuenta con una CPU Quad-Core ARM Cortex-A57 de 64 bits a 1.42 GHz de velocidad. En cuanto a la GPU, el chip está basado en la arquitectura Maxwell de Nvidia, cuenta con 128 CUDA (Compute Unified Device Architecture) cores con una velocidad de 921 MHz. La capacidad de memoria RAM es de 4 GB LPDDR4 y cuenta con todo tipo de conexiones (USB tipo C, USB Tipo B, microHDMI, conector para cámara...).

Con estas especificaciones consigue una potencia computacional de 472 GFlops. El precio de mercado de este dispositivo está alrededor de 89,00€¹.

Nvidia Jetson TX2: se trata de un modelo superior a la Jetson Nano, con especificaciones superiores, pero también más caro.

Cuenta con una CPU Quad-Core ARM Cortex-A57 de 64 bits a 2.00 GHz de velocidad. En cuanto a la GPU, el chip está basado en la arquitectura Pascal de Nvidia, cuenta con 256 CUDA cores con una velocidad de 1300 MHz. La capacidad de memoria RAM es de 8 GB LPDDR4 y cuenta con todo tipo de conexiones (USB tipo C, USB Tipo B, microHDMI, conector para cámara...).

Con estas especificaciones consigue una potencia computacional de 1.3 TFlops. El precio de mercado de este dispositivo está alrededor de 399,00€¹.

Como podemos observar, la Nvidia Jetson Nano cuenta con un procesador ligeramente inferior al de la Raspberry Pi 4 Model B. Sin embargo, la GPU es mucho más potente, con 128 núcleos a 921 MHz que permite superar en potencia computacional a la Raspberry unas 35 veces.

Además, nuestro proyecto trabaja con la visión por computadores, lo cual quiere decir que se realizará un gran número de operaciones y cálculos fácilmente paralelizables. Para poder sacar mejor rendimiento y aprovechar la concurrencia de los cálculos, viene bien hacer uso de una GPU con muchos núcleos, por tanto será necesario una GPU más potente que el chip que incluye la Raspberry Pi 4 Model B.

Por otro lado, la Nvidia Jetson TX2 es claramente superior en especificaciones a la Nvidia Jetson Nano. Cuenta con un procesador un poco más potente y una GPU con el doble de núcleos a 1300 MHz de velocidad, superando en potencia computacional casi 3 veces a la Jetson Nano, pero con un precio muy superior.

“Benchmark Analysis of Jetson TX2, Jetson Nano and Raspberry PI using Deep-CNN” (Suzen y cols., 2020) se trata de una investigación donde miden el rendimiento de estos tres dispositivos para aplicaciones que usan deep learning. Si bien nuestro proyecto de investigación no se basa en el deep learning, el principio para determinar el rendimiento de estos dispositivos es el mismo, la concurrencia.

¹El precio indicado se trata del PVP (Precio de Venta al Público) oficial recomendado por los fabricantes. Debido a la carencia de chips durante el año 2022 estos precios no se corresponden con la realidad, pudiendo encontrar estos productos por precios mucho más elevados o incluso no quedar stock de los mismos.

En el estudio de Suzen y cols. (2020) utilizan 5 conjuntos de datos de varios tamaños. En la Figura 1.4 se ve que hay un error de memoria para la Nvidia Jetson Nano y la Raspberry Pi. No es un problema que deba preocuparnos ya que esto es un problema que surge cuando se requiere de un conjunto tan grande de datos para el deep learning. Para nuestro proyecto, basado en la visión por computador, no será un límite a tener en cuenta.

También podemos ver el resultado del estudio, con el porcentaje de aciertos, el tiempo que emplean, la memoria utilizada y el consumo. Para nuestro caso, el resultado de mayor interés a tener en cuenta es el tiempo, ya que es la forma de medir qué dispositivo tiene un mejor rendimiento.

	Acc (%)			Time(sec)			Memory (GB)			CPU(Power/W)			GPU(Power/W)		
Dataset	TX2	Nano	PI	TX2	Nano	PI	TX2	Nano	PI	TX2	Nano	PI	TX2	Nano	PI
Idle	-	-	-	-	-	-	1,9	1,5	1,4	0,675	0,47	0,30	2,6	0,76	-
5K	87,6	87,5	87,2	23	32	173	2,6	2,0	2,1	2,23	1,50	3,5	5,27	2,23	-
10K	93,8	93,9	91,6	32	58	372	3,1	2,75	2,6	2,78	2,32	3,6	5,32	3,25	-
20K	94,6	94,5	-	52	-	462	4,5	ERR	4,0	3,76	-	3,9	5,22	-	-
30K	96,4	-	-	122	-	-	5,2	ERR	ERR	4,25	-	-	5,74	-	-
45K	97,8	-	-	235	-	-	6,5	ERR	ERR	4,92	-	-	6,29	-	-

Figura 1.4: Benchmarking Nvidia Jetson TX2, Nvidia Jetson Nano y Raspberry Pi 4 Model B.

Como podemos ver, las Nvidia Jetson quedan muy por encima de la Raspberry Pi 4 Model B, con unos resultados muy parecidos entre ellas, siendo superior la Nvidia Jetson TX2 en todos los casos.

Aunque la Nvidia Jetson TX2 es la clara ganadora en rendimiento, para este proyecto la potencia no lo es todo. Entre nuestros requisitos para construir el escáner 3D, los dispositivos a utilizar deben ser portables y económicos. Los 3 computadores que hemos comparado son igual de portables, pero la Jetson TX2 tiene un precio demasiado elevado para el coste económico que estamos buscando.

Para poder comparar la calidad/precio de cada uno de los dispositivos, nos fijaremos en la potencia computacional para medir la calidad de los mismos. Por tanto, nos centraremos en el precio del GFlop en cada uno de los dispositivos. Podemos ver que en la Raspberry Pi 4 Model B tendría un coste de 2,60€/GFlop, la Nvidia Jetson Nano un coste de 0,19€/GFlop y la Nvidia Jetson TX2 un coste de 0,30€/GFlop.

Además, si nos basamos en los resultados de los benchmarks que se muestran en la Figura 1.4, la diferencia de rendimiento en tiempo entre la Nvidia Jetson Nano y la Nvidia Jetson TX2 no es tan grande como para compensar el coste económico que supone hacerse con una Nvidia Jetson TX2.

Por tanto, la opción más recomendable para este proyecto es, sin lugar a dudas, la Nvidia Jetson Nano, que nos ofrece un rendimiento aceptable, muy superior a una Raspberry PI y por un precio muy económico.

1.2.3. Situación social actual, necesidades y aplicaciones

Un escáner 3D tiene la capacidad de analizar un objeto o una escena para reunir datos sobre su forma y, ocasionalmente, su color. Con la información obtenida se puede pasar a construir modelos digitales tridimensionales.

Al principio, las utilidades y aplicaciones que proporcionaba esta tecnología era utilizada exclusivamente en el ámbito industrial. Hoy en día, gracias a los avances y a la popularidad de la tecnología, es utilizada en una amplia variedad de actividades, tales como arquitectura, ingeniería, medicina, arqueología y entretenimiento, entre otras. A continuación, comentaré algunos ejemplos de ellos. Veremos como son muchas las utilidades que un escáner 3D ofrece, y que existe la necesidad de herramientas así que sean portables y de bajo coste.

En el ámbito industrial e ingenieril tiene diferentes aplicaciones debido a su gran capacidad para capturar de forma rápida y precisa los datos requeridos. De otra forma, las mediciones tendrían que ser recolectadas por métodos manuales que pueden ser menos exactos, muy costosos y consumir mucho tiempo. Como por ejemplo se estudia en el artículo “Generación de prototipos por modelado, escaneado e impresión 3D” (Rayón Encinas y cols., 2015) de la UPV (Universidad Politécnica de Valencia) donde desarrollan una metodología docente enfocada a los alumnos de IDIDP (Ingeniería en Diseño Industrial y Desarrollo de Producto) para el uso del escáner 3D. En este informe dan a conocer la importancia y necesidad del escáner 3D, donde digitalizan un objeto en 3D y realizan modificaciones en el diseño para finalmente generar un prototipo real por impresión 3D, facilitando así el desarrollo de un producto.

En el ámbito de la salud más cercano a la temática del proyecto los requerimientos de escaneo 3D son diversos, como explican en el artículo científico “3D scanning applications in medical field: A literature-based review” (Haleem y Javaid, 2019) donde hablan de la importancia de esta tecnología y del crecimiento que ha tenido en el campo de la medicina en los últimos años. Por ejemplo, los profesionales de la salud pueden de forma muy rápida realizar un escaneo corporal completo, facilitando así la obtención y comparación de datos precisos para llevar a cabo investigaciones y monitorizar los cambios en las mediciones corporales que ocurren con el tiempo. Esto es posible gracias a investigaciones como las que se han hecho en Tech4Diet (2020), con artículos científicos como “RGB-D-Based Framework to Acquire, Visualize and Measure the Human Body for Dietetic Treatments” (Fuster-Guilló y cols., 2020).

Por otro lado, el escaneo 3D también trae la posibilidad de crear soluciones personalizadas para el cuidado de la salud. Por ejemplo, en “Reconstrucción digital del muñón de un amputado transfemoral a partir de datos obtenidos de escáner 3D” (Isaza y cols., 2011) muestran como el uso de un escáner 3D permite realizar reconstrucciones sobre un molde tomado de un muñón. “Diseño de prótesis inferiores impresas en 3D para personas amputadas en Camerún” (Martínez Belda, 2020) es otro ejemplo donde muestran el desarrollo del proceso de diseño de prótesis inferiores, haciendo uso de técnicas de digitalización e impresión 3D, debido a que es rápida, económica y permite la personalización de las prótesis para cada paciente.

En el ámbito de la medicina forense también se está experimentando un crecimiento en el uso de técnicas 3D. Ofrecen portabilidad, flexibilidad y precisión que facilita la tarea de recopilar datos forenses comparado con los métodos tradicionales. Por ejemplo, en la investigación “Geotecnologías láser y fotogramétricas aplicadas a la modelización 3D de escenarios complejos en infografía forense” (Blazquez, 2015) proponen el uso de escáneres 3D para la

identificación por superposición craneofacial. Para ello, utilizan una plataforma giratoria que automatiza la adquisición de las distintas vistas del cráneo para aumentar la rapidez y precisión de la obtención del modelo 3D. Un concepto muy parecido al que se desea realizar en este trabajo.

Por último, en el ámbito de la paleontología y arqueología gracias a las tecnologías 3D se ha conseguido la creación de réplicas digitales de gran precisión de diversos artefactos, desde la recreación de elementos encontrados durante excavaciones hasta la creación de museos online con cientos de exhibiciones. En “Restauración y conservación digital de fósiles mediante escaneado 3D y la reproducción con prototipado rápido” (Valverde-Bastidas y cols., 2020) se utiliza ingeniería inversa mediante el escaneado 3D para obtener una nube de puntos en tres dimensiones de un objeto material. Posteriormente tratan esa malla para luego realizar una reingeniería, un rediseño o directamente para volver a fabricar el objeto.

1.2.4. Investigaciones y otros proyectos ya realizados

Se han investigado proyectos cuyo hardware es parecido al que se pretende utilizar en este trabajo de investigación. En “Analysis of 3D perception based on depth sensors in order to perform 3D scene understanding” (Fioriti, 2021) pretenden realizar un análisis de las características de percepción 3D con la ayuda de sensores de profundidad. Para ello han utilizado una cámara RGB-D Intel RealSense D435 conectada inicialmente a una Raspberry Pi 3, utilizando el algoritmo de detección Yolo adaptado para usar una detección 3D. De esta forma, han implementado un sistema capaz de detectar la distancia que hay con el objeto. El algoritmo puede ser utilizado tanto con imágenes como con un vídeo del objeto. El problema actual que tienen es que no puede ser utilizado en tiempo real, pues para ello necesitan acelerar el proceso. Esto es debido seguramente a las bajas prestaciones del hardware, ya que utilizan una Raspberry Pi 3 cuya GPU es muy pobre comparada con la GPU de la Nvidia Jetson Nano que utilizaremos en este proyecto.

En “Development of augmented reality systems displaying three-dimensional dynamic motion in real time” (Aoki y cols., 2020) han utilizado también el sensor D435 conectado a un dispositivo Raspberry Pi 4. Con este hardware pretenden desarrollar un sistema de seguimiento de un movimiento tridimensional mostrando en tiempo real la trayectoria en un dispositivo móvil. Por tanto, han desarrollado una aplicación para poder seguir objetos 3D con el sensor y mostrar la trayectoria en realidad aumentada desde el dispositivo móvil. Esta aplicación tiene una limitación respecto a la velocidad del movimiento, ya que la frecuencia del muestreo ha sido limitada a 8 fps debido a las especificaciones del hardware. De nuevo, esto se podría solucionar utilizando un hardware más potente como la Nvidia Jetson Nano, que permitiría paralelizar gran parte del proceso aumentando la frecuencia del muestreo máxima y permitiendo así detectar movimientos más rápidos.

1.3. Objetivos

El objetivo general es proponer una arquitectura de visión por computador que sea capaz de modelar en 3D objetos deformables. Concretamente, para este TFG el objetivo principal es hacer un sistema embebido de bajo coste que permita modelar un cuerpo o individuo articulado, como lo es un cuerpo humano. Para ello, se plantean tres subobjetivos:

- El primer subobjetivo es realizar un análisis del estado del arte para evaluar la situación en la que nos encontramos. Además, se analizarán las tecnologías y herramientas necesarias para llevar a cabo el objetivo del proyecto.
- Hacer una propuesta de solución. Este subobjetivo consistirá en integrar el SDK del sensor RGB-D Intel RealSense D435 con el dispositivo Nvidia Jetson Nano, implementar una solución, validarla y documentarla.
- Por último, experimentar y estudiar los resultados obtenidos.

1.4. Desarrollo de la memoria

A continuación, los siguientes capítulos contendrán las bases teóricas y el desarrollo del tema principal del trabajo. Está organizado de la siguiente manera:

El capítulo 2 nos presenta la base teórica necesaria para comprender el desarrollo e investigación que se ha llevado a cabo a lo largo del proyecto.

El capítulo 3 habla del sensor Intel RealSense Depth Camera D435. En este capítulo se analiza más detalladamente el funcionamiento y las características de este sensor, que es el utilizado en este trabajo.

El capítulo 4 muestra la propuesta del trabajo, explicando detalladamente los pasos que se van a seguir para obtener una solución a la reconstrucción de un cuerpo humano utilizando el sensor Intel RealSense D435 y el sistema embebido Jetson Nano.

En el capítulo 5 explica cómo se ha llevado a cabo la propuesta del capítulo anterior, mostrando fragmentos de código y la solución final obtenida.

Seguidamente en el capítulo 6 se mostrarán los resultados que se obtienen con la solución del proyecto.

Finalmente, en el capítulo 7 nos encontraremos con la conclusión del trabajo elaborado en este proyecto.

2. Base teórica

La reconstrucción de cuerpos u objetos en 3D a partir de un sensor RGBD se consigue gracias al uso de métodos de registro que realizan cálculos de correspondencias entre dos vistas distintas. Es decir, dado un conjunto de datos de entrada, donde cada entrada son capturas de nubes de puntos de un mismo cuerpo realizadas desde distintos ángulos, el método de registro se encargará de encontrar una transformación que alinee estas nubes de puntos para finalmente reconstruir el cuerpo entero. Esta transformación no es más que una combinación de rotaciones y traslaciones en los ejes de coordenadas **X**, **Y**, **Z** sobre la nube de puntos.

2.1. Métodos de registro

Los métodos de registro tratan de encontrar una correspondencia entre la posición de los puntos de una nube de puntos y otra, emparejándolos para calcular de esta forma la transformación global que corresponda de mejor manera entre los dos conjuntos de nube de puntos. Estos métodos realizan cálculos en ocasiones de forma iterativa, para encontrar la mejor correspondencia entre los puntos y así estimar la transformación que consiga alinear ambas vistas.



(a) Captura del modelo.



(b) Captura de la escena.



(c) Modelo y escena superpuestos.



(d) Escena alineada con el modelo.

Figura 2.1: Ejemplo de alineación de una escena con un modelo.

Como podemos ver en la Figura 2.1, dado dos entradas de nubes de punto, el modelo y la escena, obtenemos su alineación a través de un método de registro. Normalmente a la nube de puntos de referencia con la que se tratará de alinear el resto de nubes de puntos recibe el nombre de modelo. Las nubes de puntos que se tratan de emparejar reciben el nombre de escena. Cuando aplicamos un método de registro sobre una escena (Figura 2.1b) y tomamos como referencia un modelo (Figura 2.1a) lo que conseguiremos es transformar la escena para que quede alineada y así forme parte del modelo (Figura 2.1d). De esta forma, gracias a la alineación de la escena sobre el modelo, conseguimos que el modelo quede más completo.

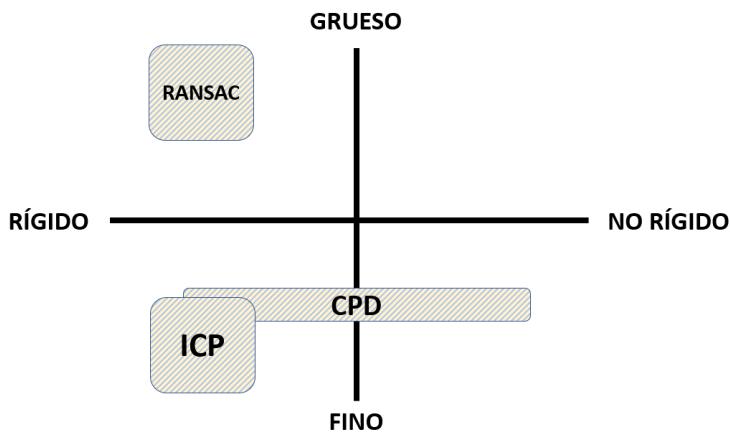


Figura 2.2: Diagrama tipos de registro.

En la Figura 2.2 podemos ver un diagrama que sitúa algunos de los métodos de registros de los que hablaremos en este TFG en función de sus características. Podemos ver que un método de registro se puede diferenciar por ser de grano grueso o grano fino, y por el tipo de registro que se está utilizando para alinear los datos: rígido o no rígido.

Los métodos de grano grueso suelen utilizarse como paso inicial en el registro debido a que suelen ser poco precisos, pero ofrecen una buena aproximación inicial para vistas muy separadas entre sí. Estos métodos suelen muestrear los datos y utilizan métodos basados en detección y descripción de características para intentar reducir la cantidad de puntos tanto de la escena como del modelo. Una característica (keypoint) tiene una posición y un descriptor que describe ese keypoint. Las características proceden de la captura en sí ya sea una imagen 2D (Dos Dimensiones) o una nube de puntos (imagen 3D). En este paso de detección de características se intenta detectar las partes distintivas de un objeto en los conjuntos de datos, como formas, regiones cerradas, contornos, líneas, etc. Una vez detectadas las partes del objeto se representa como un conjunto de valores normalmente llamados descriptores de características. Tras la detección y descripción de características, se buscan correspondencias entre las mismas y finalmente una estimación de la transformación.

El método basado en características más utilizado para encontrar la transformación entre correspondencias está basado en el paradigma RANSAC (Random Sample Consensus), publicado por primera vez por Fischler y Bolles (1981). RANSAC es un método iterativo para calcular los parámetros de un modelo matemático de un conjunto de datos observados que contiene valores atípicos. Es un algoritmo no determinista en el sentido de que produce un

resultado razonable solo con una cierta probabilidad, mayor a medida que se permiten más iteraciones. En cada iteración del algoritmo, se selecciona aleatoriamente un subconjunto de correspondencias a los que se les aplica una matriz de transformación. Los datos restantes se testean con el modelo aplicado de manera que si su error está por encima de un umbral se eliminan de los datos seleccionados, de esta forma se descartan los puntos con errores extremadamente altos (*outliers*). En la Figura 2.3 podemos ver un ejemplo de alineación con imágenes 2D utilizando RANSAC. En la subfigura 2.3d podemos ver un ejemplo de resultado de una iteración que ha tenido buena probabilidad con la correspondencia entre el subconjunto de puntos seleccionados (subfigura 2.3c). En la subfigura 2.3e podemos ver un ejemplo de una iteración que ha partido de una mala correspondencia entre los subconjuntos de los puntos seleccionados. Este proceso se repite varias veces, finalmente se utilizará la iteración que mejor solución presente como aproximación inicial para un método de registro de grano fino.

Los métodos de grano fino se usan normalmente cuando se necesitan resultados más precisos. Por lo general, estos métodos necesitan el resultado de un registro de grano grueso previo como dato inicial del que partir. Cuando se le aplica el resultado de un método de grano grueso a la escena, esta se encontrará más próxima al modelo, facilitando así la ejecución de un método de grano fino. Sin embargo, no siempre es necesario una alineación previa si las nubes de puntos que se intentan alinear ya están lo suficientemente cerca. Por ejemplo, cuando se usa una frecuencia de capturación en el sensor muy alta, y el movimiento entre dos capturas consecutivas es muy pequeño.

Como conclusión entre los métodos de grano grueso y fino, los métodos de registro fino se suele utilizar para refinar un registro en el que la escena y el modelo se encuentran muy próximos, en contraste con los métodos de grano grueso que se suelen utilizar para distancias mayores con la finalidad de aproximar la escena al modelo sin ser del todo preciso.

Por otra parte, los métodos de registro también se pueden diferenciar entre métodos de registro rígidos y métodos de registro no rígidos. Cuando hablamos de un objeto o cuerpo no rígido, nos referimos a un objeto o cuerpo articulado o deformable. El registro rígido se utiliza cuando la diferencia entre modelo y escena consiste en una matriz de transformación, es decir, la escena sobre el objeto se ha rotado o movido respecto al modelo sobre el objeto, pero en ambos casos el objeto es el mismo y mantiene la misma forma. Cuando hablamos de un registro no rígido es porque el objeto de la escena puede haber sido deformado, además de rotado o movido respecto al modelo.

2.1.1. Métodos de registro rígidos, ICP

El método de registro rígido más utilizado es el ICP (Iterative Closest Point) Besl y McKay (1992). Es un método de propósito general, independiente de la representación y cuyo cometido es encontrar la transformación entre una nube de puntos y otra, minimizando los errores entre los puntos emparejados de ambas nubes. Este método trata de realizar el registro de nubes de puntos de manera eficiente desde el punto de vista computacional. Si las nubes de puntos se encuentran lo suficientemente cerca, el algoritmo es capaz de proporcionar muy buenos resultados en la mayoría de casos.

En la Figura 2.4 podemos ver un esquema del funcionamiento del algoritmo ICP. Este algoritmo recibe como parámetros de entrada la nube de puntos de la escena (fuente) y la nube de puntos del modelo (destino), además de los parámetros de configuración del algoritmo. El

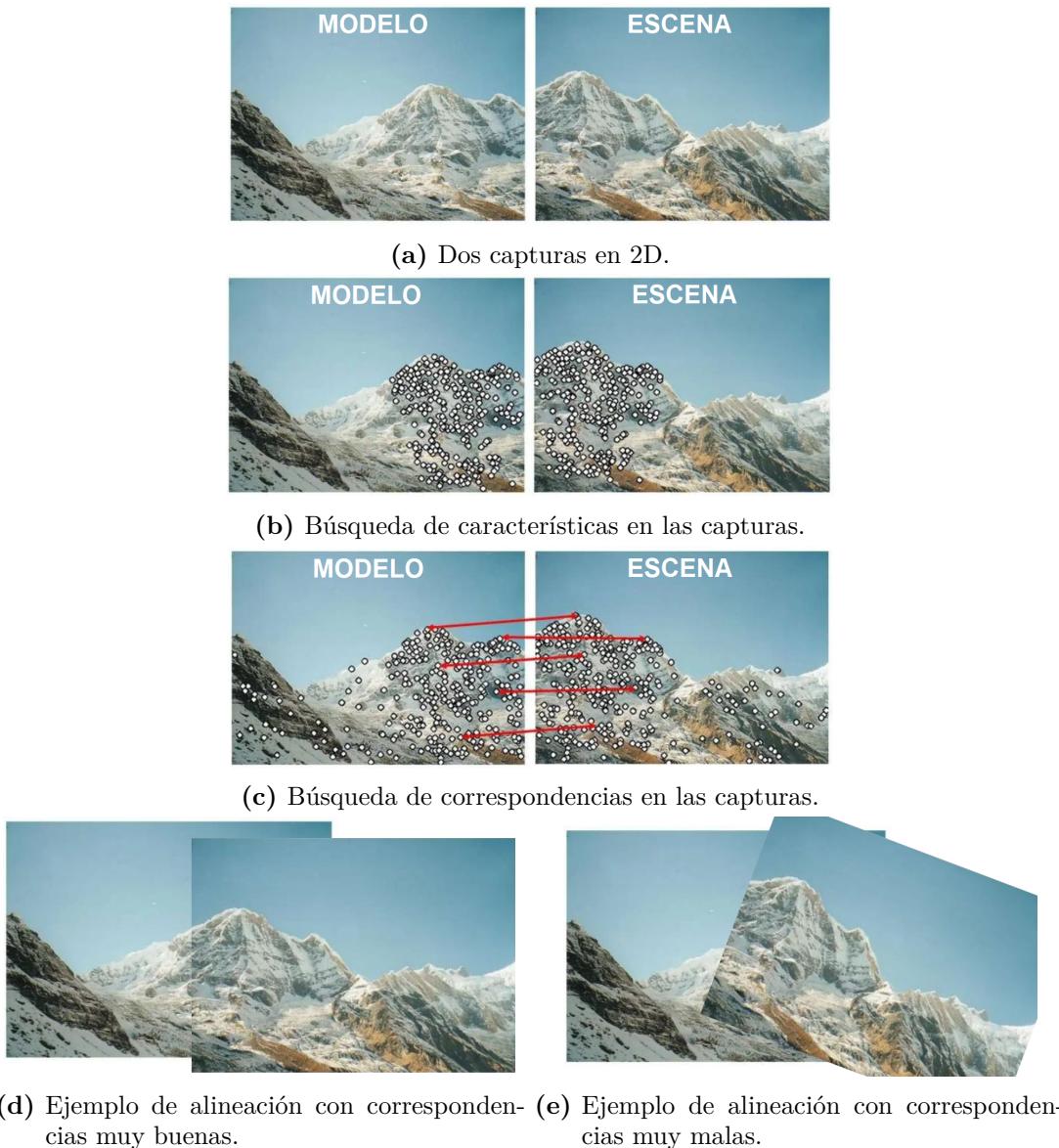


Figura 2.3: Ejemplo de alineación de una escena usando RANSAC.

algoritmo parte de la escena, que es la nube de puntos a alinear, y aplica diversas transformaciones con el objetivo de acercarse a la nube destino, el modelo. El proceso se realiza de forma iterativa, partiendo en cada nueva iteración de la transformación anterior, hasta que finalmente la nube de puntos converge con el destino. El resultado de aplicar el algoritmo ICP es la nube de puntos de la escena alineada, a la vez que una matriz de transformación final. Si esta matriz de transformación la aplicamos sobre la nube de puntos de la escena original, obtendremos una nube de puntos de escena alineada con la nube de puntos del modelo.

Además, al método ICP original se pueden aplicar diferentes variaciones con la finalidad de mejorar su robustez. Por ejemplo, se le puede añadir un parámetro para indicar el límite

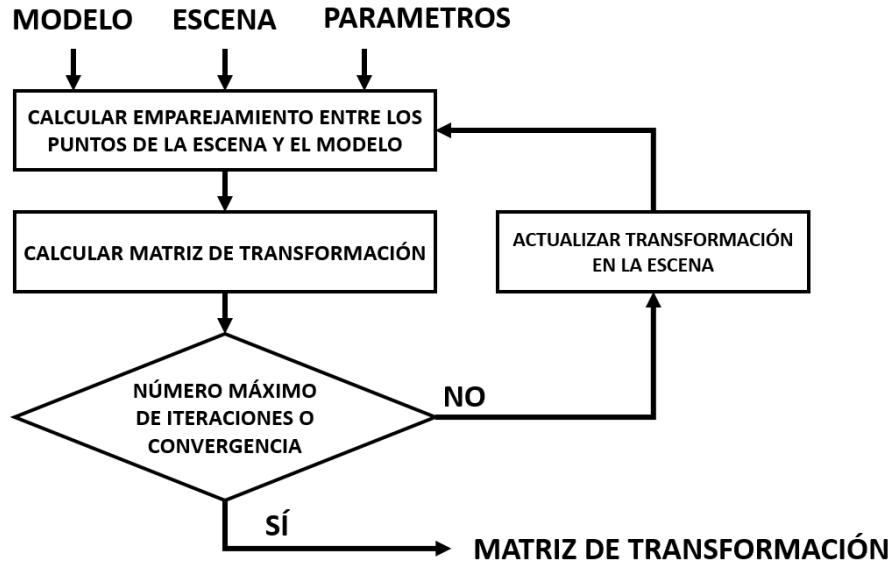
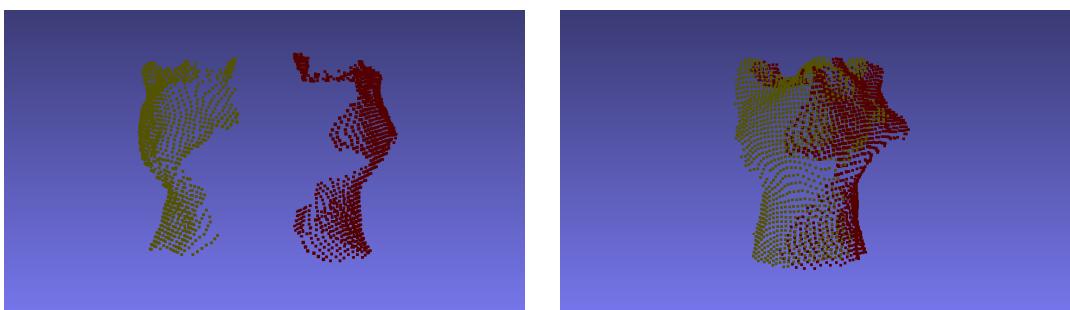


Figura 2.4: Diagrama del algoritmo ICP.

máximo de distancia para la correspondencia entre puntos, es decir, un valor que indique el radio de distancia desde cada punto en la nube de puntos de origen (escena) en el que la búsqueda de vecinos intentará encontrar un punto correspondiente en la nube de puntos de destino (modelo). Este parámetro puede ser muy importante ya que limitar esta distancia ayuda en el rendimiento (cuanto menor sea el radio más rápido será) y puede evitar que el resultado no converja si la diferencia entre ambas nubes de puntos es muy grande.



(a) Modelo y escena superpuestos sin alinear. (b) Modelo y escena superpuestos alineados.

Figura 2.5: Ejemplo de captura de escena muy alejada al modelo.

En la Figura 2.5 podemos ver un caso donde gran parte de la nube de puntos de la escena no corresponden con el modelo, por tanto son pocos la cantidad de puntos que deberían coincidir entre ambas nubes. En este caso, aplicar un umbral de distancia máxima entre correspondencias será muy útil para que el algoritmo ICP descarte los emparejamientos entre los puntos más lejanos. También son importantes parámetros como el número de iteraciones máximas o el valor umbral para el error medio de emparejamiento que definirá la cantidad

de iteraciones que el algoritmo ICP debe realizar para obtener la convergencia entre escena y modelo.

2.1.2. Métodos de registro no rígidos, CPD

Mientras que una transformación rígida solo permite la traslación, rotación y escalado de la escena, la transformación no rígida permite tanto transformar afines, articulares y sesgos anisotrópicos. Las aproximaciones simplistas de transformación no rígida verdadera, incluidos los modelos polinómicos y afines por partes, a menudo son inadecuadas para la alineación correcta y pueden producir correspondencias erróneas. También suelen tener una alta complejidad computacional. Además, debido al gran número de parámetros de transformación, los métodos de registro de nubes de puntos no rígidas tienden a ser sensibles al ruido y a los valores atípicos y es probable que converjan en mínimos locales. Las degradaciones como el ruido, los valores atípicos y los puntos faltantes complican significativamente el problema. Los valores atípicos son los puntos que se extraen incorrectamente de la imagen, estos valores atípicos no tienen correspondencias en el otro conjunto de puntos.

CPD (Coherent Point Drift) es un método de registro desarrollado originalmente por Myronenko y Xubo Song (2010). CPD presenta un robusto algoritmo de registro de nubes de puntos probabilístico para transformaciones rígidas y no rígidas. Mientras que ICP minimiza las distancias punto a punto, CPD utiliza un MMG (Modelo Mixto Gaussiana) para minimizar el error entre un punto y todos los demás puntos. La alineación de dos nubes de puntos se considera como un problema de estimación de densidad de probabilidad, donde una nube de puntos representa los centroides del MMG y el otro representa los puntos de datos. Los centroides de MMG se ajustan a los datos maximizando la probabilidad. En el punto óptimo, las nubes de puntos se alinean y la correspondencia se obtiene usando las probabilidades posteriores de los componentes MMG. El núcleo del algoritmo es obligar a los centroides MMG a moverse coherentemente como un grupo, lo que preserva la estructura topológica de los conjuntos de puntos.

El hecho de utilizar el Modelo Mixto Gaussiana entre punto y punto conlleva un rendimiento muy intensivo desde el punto de vista computacional. Este algoritmo utiliza FGT (Fast Gauss Transforms), una librería para acelerar las transformaciones gaussianas, también desarrollado por Myronenko y Xubo Song (2010). No obstante, tanto el algoritmo CPD como los cálculos de errores subyacentes tardan mucho tiempo, a pesar de utilizar FGT.

2.2. Matriz de transformación 3D

Como este TFG utiliza tipos de datos 3D, los conceptos que se exponen serán enfocados a este tipo de datos. La matriz de transformación (Transformation matrix, 25/05/2022) es el resultado que obtenemos tras aplicar el método de registro. Con la matriz de transformación, podemos transformar la nube de puntos de la escena y alinearla con la nube de puntos modelo. La matriz de transformación se aplica directamente sobre una nube de puntos para transformarla. Una nube de puntos es una matriz de $3 \times N$ donde N es la cantidad de puntos, para cada uno de los cuales tenemos el valor X, Y, Z .¹ Al aplicar la matriz sobre la nube de

¹Una nube de puntos RGB, es decir, con color, será de $6 \times N$: los valores de posición X, Y, Z ; y los valores de color R, G, B . En estos casos, las matrices de transformación sólo se aplicarán a los valores X, Y, Z .

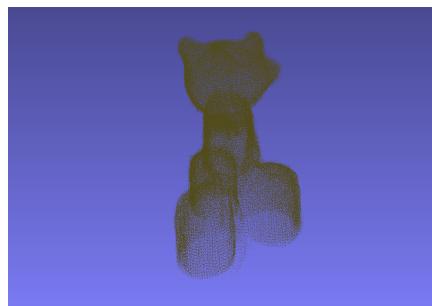
puntos conseguimos transformarla, rotando y trasladando los puntos en su interior, con la finalidad, en este caso, de alinearlos con otra nube de puntos.

Una matriz de transformación está compuesta de una matriz de rotación y una matriz de traslación.

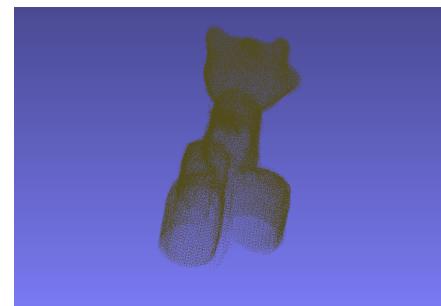
La matriz de rotación (Rotation matrix, 22/04/2022) es una matriz 3×3 que contiene el valor por el que se necesita multiplicar cada uno de los puntos de una nube de puntos para ser rotada. Una rotación básica consiste en una rotación que afecte solamente a uno de los ejes de coordenadas. En la Ecuación 2.1 podemos ver las tres matrices de rotación básicas para cada uno de los ejes, donde θ indica el ángulo con el que los ejes van a rotar. Por ejemplo, si queremos rotar nuestra nube de puntos 15° grados en el eje X, habría que aplicar la matriz de rotación $\mathbf{R}_x(15)$ a nuestra nube de puntos. Tras multiplicar nuestra nube de puntos $3 \times N$ por esta matriz, obtendremos una nueva nube de puntos con nuevos valores que resultará en la misma nube de puntos rotada 15° en el eje X, como se puede ver en el ejemplo de la Figura 2.6.

$$\begin{aligned}\mathbf{R}_x(\theta) &= \begin{vmatrix} 1 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta \\ 0 & \sin \theta & \cos \theta \end{vmatrix} \\ \mathbf{R}_y(\theta) &= \begin{vmatrix} \cos \theta & 0 & \sin \theta \\ 0 & 1 & 0 \\ -\sin \theta & 0 & \cos \theta \end{vmatrix} \\ \mathbf{R}_z(\theta) &= \begin{vmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{vmatrix}\end{aligned}\quad (2.1)$$

Ecuación 2.1: Matrices de rotación básicas.



(a) Modelo 3D original.



(b) Modelo 3D rotado 15° en el eje X.

Figura 2.6: Rotación de 15° de un modelo 3D en el eje X.

La traslación (Translation matrix, 14/01/2022) es una transformación geométrica que per-

mite mover cada punto de la nube de puntos la misma distancia y en la misma dirección. Este movimiento puede ser representado en forma de vector o de matriz (Ecuación 2.2).

$$\mathbf{T}_v = \begin{vmatrix} v_x \\ v_y \\ v_z \end{vmatrix} \quad \mathbf{T}_v = \begin{vmatrix} 1 & 0 & 0 & v_x \\ 0 & 1 & 0 & v_y \\ 0 & 0 & 1 & v_z \\ 0 & 0 & 0 & 1 \end{vmatrix} \quad (2.2)$$

Ecuación 2.2: Vector y matriz de Traslación.

La traslación es una transformación afín sin puntos fijos, sin embargo, la multiplicación en las matrices siempre tienen su origen como punto fijo. Para solucionar este inconveniente, al vector de 3 dimensiones $\mathbf{v} = (v_x, v_y, v_z)$ se le añade una dimensión más $\mathbf{v} = (v_x, v_y, v_z, 1)$, de esta forma conseguimos trabajar utilizando coordenadas homogéneas para representar la traslación en un espacio vectorial con multiplicación de matrices. Como podemos ver en la Ecuación 2.3, se está multiplicando una matriz de traslación por uno de los puntos de una nube de puntos. El resultado de esta multiplicación consiste en ese mismo punto sumándole el movimiento de la matriz de traslación.

$$\mathbf{T}_v \mathbf{p} = \begin{vmatrix} 1 & 0 & 0 & v_x \\ 0 & 1 & 0 & v_y \\ 0 & 0 & 1 & v_z \\ 0 & 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} p_x \\ p_y \\ p_z \\ 1 \end{vmatrix} = \begin{vmatrix} p_x + v_x \\ p_y + v_y \\ p_z + v_z \\ 1 \end{vmatrix} = p + v \quad (2.3)$$

Ecuación 2.3: Matriz de Traslación aplicada a un punto.

Por tanto, una matriz de transformación no es más que la combinación de un matriz de rotación y una matriz de transformación, dando lugar a una matriz 4×4 que contiene la información de ambas matrices (Ecuación 2.4). Para más detalles, se recomienda leer "Multiple view geometry in computer vision" de Hartley y Zisserman (2003).

$$\mathbf{T} = \begin{vmatrix} \mathbf{r}_{0,0} & \mathbf{r}_{0,1} & \mathbf{r}_{0,2} & \mathbf{v}_x \\ \mathbf{r}_{1,0} & \mathbf{r}_{1,1} & \mathbf{r}_{1,2} & \mathbf{v}_y \\ \mathbf{r}_{2,0} & \mathbf{r}_{2,1} & \mathbf{r}_{2,2} & \mathbf{v}_z \\ 0 & 0 & 0 & 1 \end{vmatrix} \quad (2.4)$$

Ecuación 2.4: Matriz de Transformación.

La matriz de transformación contiene información importante no solo para la alineación de una escena con el modelo, sino para la reconstrucción de todo el cuerpo 3D a partir de varias escenas distintas, ya que será necesario tener en cuenta cada una de las transformaciones aplicadas para usarlas como transformación inicial en una nueva escena a alinear.

Por ejemplo, si hemos realizado 3 capturas desde distintos ángulos (Figuras 2.7a, 2.7b, 2.7c), tomaremos la captura 1 como modelo, y las capturas 2 y 3 como escenas que hay

que alinear con el modelo. Tras conseguir la matriz de transformación que alinea la captura 2 (escena) con la captura 1 (modelo), nuestra captura 2 transformada pasará a convertirse también en el modelo, pues estará alineada con la captura 1 (Figura 2.7d). Para alinear la captura 3 (nueva escena) con las capturas 1 y 2 (actual modelo) se le aplicará a la captura 3 como transformación inicial el resultado de alinear la captura 2 con la 1 (Figura 2.7e), de esta forma, la captura 3 se acercará un poco más al modelo (capturas 1 y 2). Seguidamente volvemos a aplicar el método de registro para obtener la captura 3 alineada (Figura 2.7f).

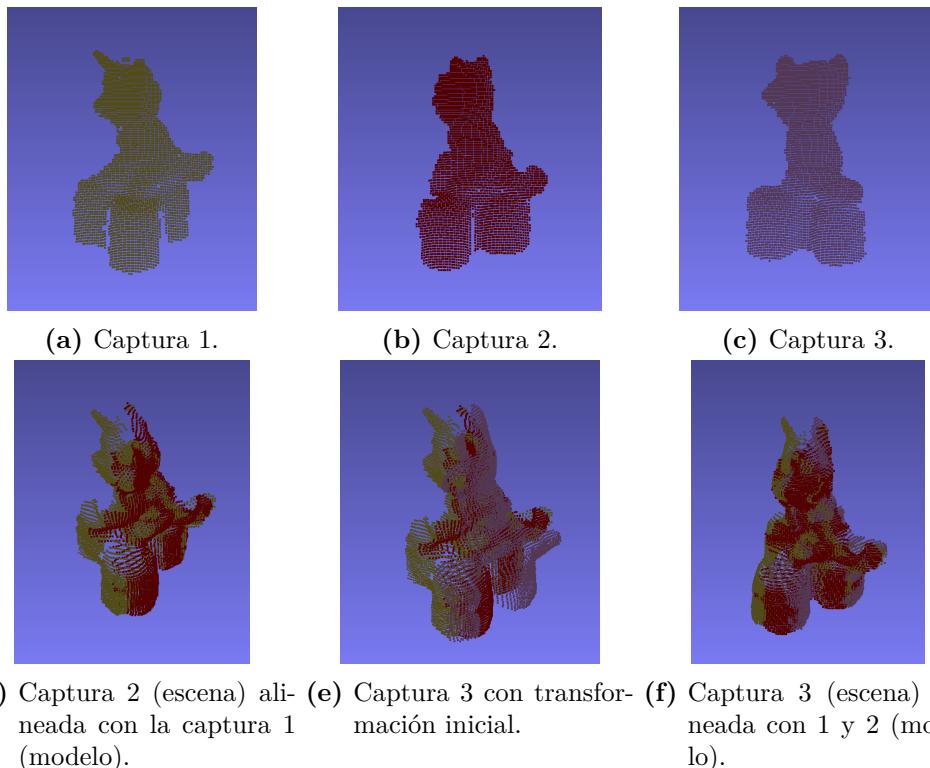


Figura 2.7: Ejemplo de matriz de transformación inicial.

2.3. Filtros para el pre-procesamiento y post-procesamiento de los datos

Tras la fase de adquisición, donde obtenemos una nube de puntos que identificamos como modelo y el resto de nubes de puntos que identificamos como escenas, es buena idea hacer uso de filtros sobre las nubes de puntos en una fase de pre-procesamiento antes de ser enviadas al método de registro donde serán finalmente procesadas. Esto puede ayudar al método de registro para que funcione mejor encontrando las correspondencias entre las nubes de puntos.

Además, tras ser procesadas por el método de registro, también se puede volver a aplicar otros filtros en una fase de post-procesamiento. Este filtro de post-procesamiento puede ayudar a obtener un resultado de mayor calidad para la fase final de análisis.

En la Figura 2.8 podemos ver un esquema que muestra en qué momento se aplicarán los

filtros de pre-procesamiento y post-procesamiento.

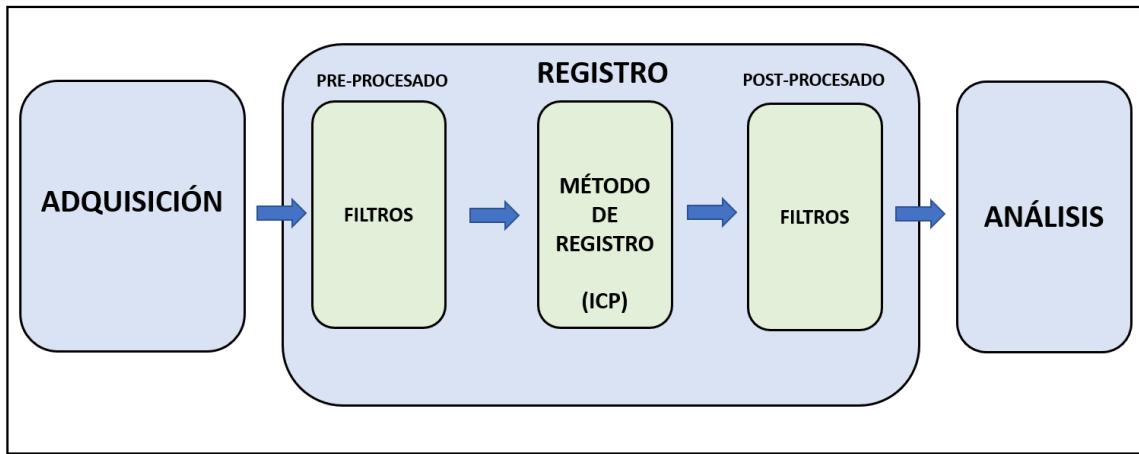


Figura 2.8: Filtros de pre-procesamiento y post-procesamiento en un esquema general de un sistema 3D.

A continuación, se detallarán una serie de filtros que se aplicarán en fase de pre-procesado que harán que el método de registro funcione mejor.

Filtro de diezmado (Decimation Filter): el filtro de diezmado ayuda a eliminar el exceso de ancho de banda y reduce la frecuencia de muestreo de la señal a una frecuencia de muestreo más baja que difiere de la frecuencia original en un valor entero. También ayuda a reducir los recursos computacionales necesarios para procesar y almacenar la señal (Ambitiously, 2022).

La razón para aplicar este filtro normalmente consiste en reducir la frecuencia de muestreo en la salida de un sistema para que un sistema que opera a una frecuencia de muestreo más baja pueda ingresar la señal. Pero una motivación mucho mayor para diezmar, y motivo por el cual es interesante para este proyecto, es reducir el costo de procesamiento al tratar con los datos tras ser aplicado el filtro. Gracias a este filtro obtenemos una nube de puntos con menor cantidad de puntos, lo cual permitirá al método de registro trabajar más rápido.

Este filtro es muy parecido al downsampling, sin perder demasiada calidad en la nube de puntos. El uso de este filtro está recomendado en la documentación de Intel, y proporcionan un método en la librería de RealSense para aplicarlo a la hora de tomar la captura.

PassThrough Filter: este filtro consiste en definir unos valores para cada dimensión donde la nube de puntos será recortada. Es decir, se especifica un rango para las dimensiones X, Y, Z, y todo lo que esté fuera de dicho rango será recortado de la nube de puntos.

Con este filtro conseguimos por una parte, reducir la cantidad de puntos finales, con lo que conseguimos una nube de puntos más liviana y, como se ha comentado en los filtros anteriores, esto hará que el método de registro funcione más rápido. Pero lo más importante es que podemos eliminar la parte de la nube de puntos que no nos

interesa y que, además, podría entorpecer en el funcionamiento del método de registro. Por ejemplo, si lo que estamos capturando con el sensor es una persona dando vueltas en el mismo sitio, todos los objetos, paredes y entorno que le rodea no es información relevante para el registro. De hecho, puede provocar un peor funcionamiento si se dejara, pues son puntos fijos que no cambian entre captura y captura.

Statistical Outlier Removal Filter: este filtro es muy interesante porque nos permite eliminar valores atípicos en la nube de puntos, que probablemente serán mediciones ruidosas que ha capturado el sensor. Los errores de medición en la fase de adquisición conducen a valores atípicos dispersos que corrompen los resultados del registro.

La eliminación estadística de valores atípicos funciona realizando un análisis estadístico de la vecindad de cada punto y recortando aquellos que no cumplen con un determinado criterio. La eliminación de valores atípicos dispersos se basa en el cálculo de la distribución de puntos a distancias vecinas en el conjunto de datos de entrada. Para cada punto, se calcula la distancia media desde él a todos sus vecinos. Todos los puntos cuyas distancias medias están fuera de un intervalo definido por la media y la desviación estándar de las distancias globales pueden considerarse valores atípicos y recortarse del conjunto de datos.

Por tanto el uso de este filtro nos permite obtener una nube de puntos más limpia eliminando los puntos lejanos, puntos atípicos y que no corresponden con el objeto o cuerpo capturado.

Filtro de mediana (Median Filter): El filtro de la mediana es uno de los filtros de procesamiento de imágenes más simples y más extendidos, se sabe que funciona bien con ruido de disparo o ruido de impulso.² Es simple de implementar y eficiente, ya que requiere una sola pasada sobre la imagen. Consiste en una ventana móvil de tamaño fijo que reemplaza el píxel del centro por la mediana dentro de la ventana.

Los filtros de eliminación estadística de valores atípicos (Statistical Outlier Removal) y de mediana también son filtros muy interesantes para aplicar en la fase de post-procesado, ya que además de permitir un mejor funcionamiento durante el método de registro, mejora la calidad de la nube de puntos resultante.

2.4. Aceleración por GPU, CUDA

Los métodos de registro 2D y 3D suelen proporcionar buenos resultados, sin embargo presentan restricciones temporales debido a su naturaleza secuencial y a la gran cantidad de cálculos que deben realizar. Es por ello que se buscan métodos para acelerar dichos algoritmos que minimicen todo lo posible las restricciones temporales. La forma tradicional de aceleración aplicada a estos métodos es la computación paralela.

El paradigma GPGPU (General-Purpose Computing on Graphics Processing Units) (Luebke y cols., 2006) es un concepto dentro de la informática que trata de estudiar y aprovechar las capacidades de cómputo de una GPU. Inicialmente los dispositivos GPU se utilizaban únicamente para el procesamiento de los gráficos de un ordenador cuyos datos a tratar estaban

²El ruido de disparo o de impulso son píxeles individuales que tienen valores extremos.

enfocados a ser convertidos a formato gráfico. En la actualidad combinan su gran potencia de cálculo para albergar cálculos paralelos de propósito general en los que se trabaja con grandes cantidades de datos que no tienen por qué ser convertidos a formato gráfico.

Existen dos tecnologías principales para la programación en dispositivos GPU: CUDA (Compute Unified Device Architecture) y OpenCL (Open Computing Language). En un principio la programación para aprovechar las características de este tipo de dispositivos debía hacerse mediante código ensamblador. Sin embargo, hoy día existen alternativas con lenguajes de alto nivel que facilitan el desarrollo de las aplicaciones. CUDA (Luebke, 2008) es una plataforma de computación en paralelo que incluye un compilador y un conjunto de herramientas de desarrollo creadas por Nvidia que permite utilizar una variación del lenguaje de programación C (CUDA C) para codificar algoritmos en GPUs de Nvidia. OpenCL (Stone y cols., 2010) por su parte consta de una interfaz de programación de aplicaciones y de un lenguaje de programación que permiten crear aplicaciones con paralelismo a nivel de datos y de tareas que pueden ejecutarse en GPU. Se trata de una alternativa a CUDA de código libre y abierta.

Los dispositivos GPUs consiguen mejor rendimiento que las CPUs procesando datos en coma flotante debido a su arquitectura. Como podemos ver en la Figura 2.9 (Stopper y Roth, 2017), un dispositivo GPU está diseñado para el cálculo intensivo altamente paralelo, por lo que tiene más transistores destinados al procesamiento de datos que al flujo de control o a la caché de datos. Las GPUs son dispositivos especialmente adecuados para resolver problemas que puedan expresarse como cálculos con datos realizados simultáneamente, ejecutando el mismo programa sobre muchos datos en paralelo. Los cálculos que se realizan llevan una alta intensidad aritmética, es decir, un elevado ratio de operaciones aritméticas por operación de memoria, lo que hace innecesaria la utilización de grandes cantidades de caché de datos.

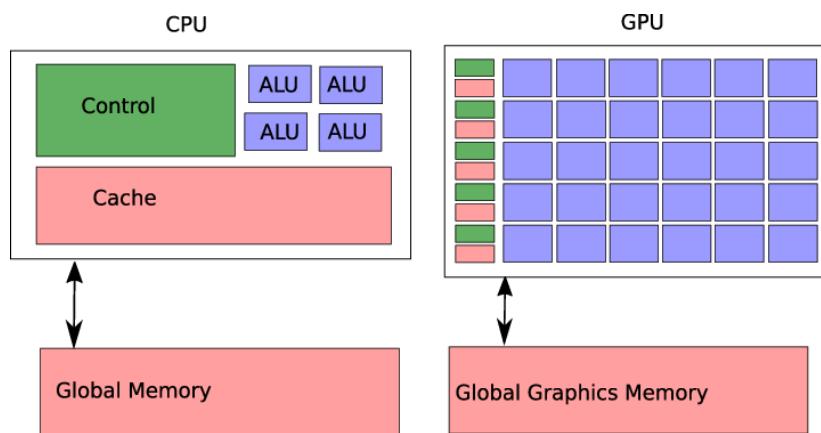


Figura 2.9: Comparación de arquitecturas CPU y GPU. Figura extraída de Stopper y Roth (2017).

CUDA es una arquitectura con modelo SIMD (Single Instruction Multiple Threads), es decir, una instrucción, múltiples hilos. Estos hilos se ejecutan simultáneamente, trabajando sobre grandes cantidades de datos en paralelo. En la Figura 2.10 (Glaskowsky, 2009) podemos ver la estructura de los dispositivos GPU compatibles con la arquitectura CUDA, los cuales están formados por un conjunto de multiprocesadores. Estos multiprocesadores reciben el nombre de SM (Streaming Multiprocessors) y permiten trabajar con múltiples hilos en paralelo.

lelo. Sin embargo, el número de multiprocesadores varía dependiendo de la arquitectura de la GPU. Cada SM se compone de una serie de SP (Streaming Processors) que comparten la lógica de control y la memoria caché. Cada uno de estos SP puede lanzar una gran cantidad de hilos en paralelo.

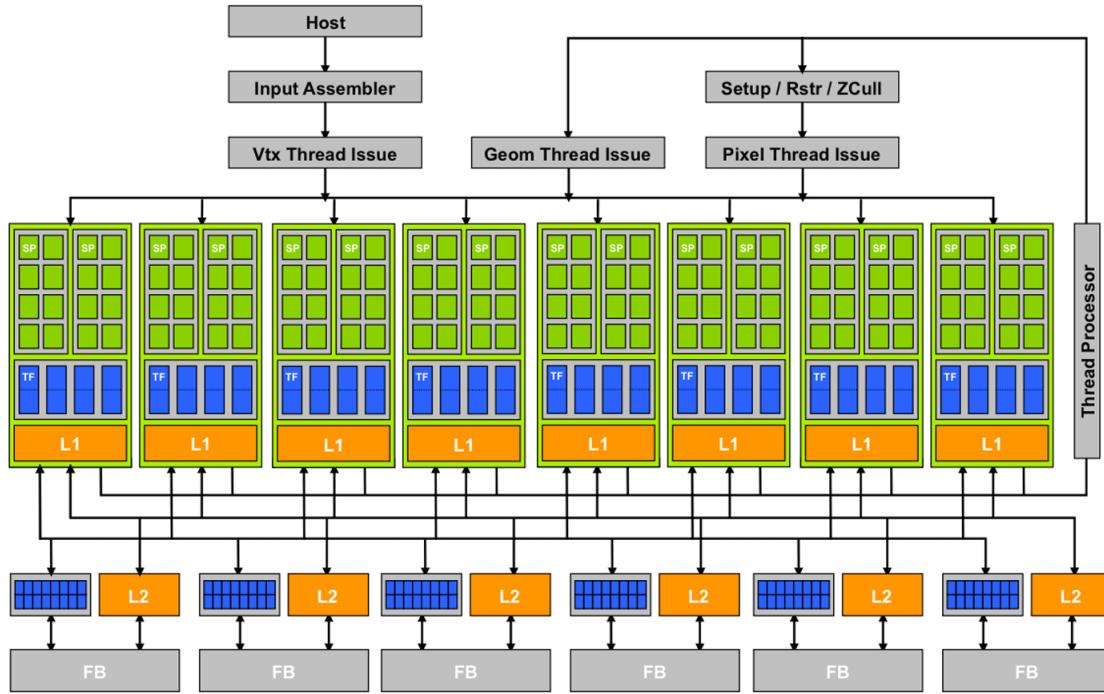


Figura 2.10: Arquitectura GPU con tecnología CUDA. Figura extraída de Glaskowsky (2009).

Una vez definida la importancia de los dispositivos GPU para la resolución de problemas de cálculo intensivo a los que se aplican computación paralela, para utilizar estos dispositivos con el objetivo de acelerar un determinado algoritmo será necesario realizar un rediseño del algoritmo que se adapte a la arquitectura de la GPU. Para parallelizar un algoritmo de forma eficiente será necesario identificar claramente unidades de trabajo independientes capaces de trabajar con subconjuntos de datos. Este es un aspecto crítico a la hora de rediseñar un algoritmo para que funcione concurrentemente.

El algoritmo ICP requiere mucho tiempo de proceso en la CPU debido a la gran cantidad de operaciones que deben realizar en cada una de sus etapas, concretamente la encargada de calcular los emparejamientos entre las nubes de puntos escena y modelo. En esta etapa es donde se encuentra el principal cuello de botella del algoritmo ICP, ya que el algoritmo debe encontrar para cada punto de la escena el punto del modelo más cercano. Esta etapa del algoritmo realizada en la CPU de forma secuencial emplea una gran cantidad de tiempo. Muchas de estas operaciones se pueden realizar en paralelo ya que son independientes y trabajan sobre el mismo conjunto de datos, lo cual hace que el algoritmo cumpla con los requisitos básicos para poder ser implementado de forma paralela, orientado a su ejecución en una GPU, de esta manera se puede reducir notablemente el tiempo de procesamiento que

emplea el algoritmo.

Existen varias implementaciones desarrolladas en GPU del algoritmo ICP (Langis y cols., 2001) (Barsukov, 2013). Incluso Nvidia ha creado oficialmente un repositorio con implementaciones sobre la librería PCL (Point Cloud Library), incluyendo una implementación sobre el algoritmo ICP (Lei Fan, 2021).

3. Intel RealSense Depth Camera D435

Este TFG se enmarca en un proyecto que utiliza la cámara RGB-D Intel RealSense Depth Camera D435, por lo que aquí se explica el funcionamiento de esta.

Intel RealSense Depth Camera D435 es un sensor RGB-D de propósito general. Este sensor 3D tiene una profundidad mínima de 0,2 metros y escanea entornos de hasta 10 metros de ancho con una profundidad de resolución de hasta 1280x720 a 90 fotogramas por segundo (fps). El sistema utiliza un procesador Intel D4vision de 28 nanómetros para procesar datos de profundidad complejos en tiempo real y un procesador de imágenes en color Realtek. El proyector láser IR mide solo 2,7x1,8 milímetros, utiliza tecnología VCSEL (Vertical Cavity Surface Emitting Laser) y puede grabar 60 fps. El dispositivo RealSense D435 tiene tres sensores para capturar la imagen: un sensor CMOS 1080p RGB para capturar imágenes con color y dos sensores de 1 megapíxel a los lados izquierdo y derecho del sensor IR VCSEL. Estos dos últimos sensores son utilizados para obtener las imágenes en profundidad mediante una tecnología estéreo activa. El sensor es capaz de proporcionar imágenes de alta definición gracias al procesador de visión, el proyector VCSEL y el sensor de imagen. En la Figura 3.1 podemos ver los sensores descritos en el Intel RealSense D435.

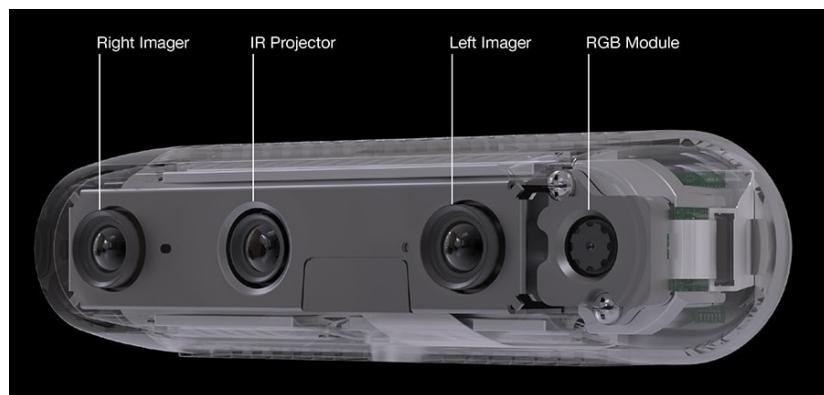


Figura 3.1: Sensores Intel RealSense D435.

La tecnología estéreo activa, o “Active Stereo” en inglés, funciona por triangulación. El sensor puede ver el mismo punto desde dos ubicaciones diferentes, desde el sensor de 1 megapíxel ubicado a la izquierda del sensor VCSEL y desde el sensor de 1 megapíxel ubicado a la derecha. Con esta información puede usar la trigonometría para medir la distancia real que hay desde el sensor hasta el punto. Tras identificar un punto común en ambas imágenes, usa el ángulo conocido de cada sensor para cada imagen y la distancia entre ambos sensores para calcular la distancia. Para ello, el sensor tiene un calibrado interno sobre el ángulo y distancia de ambos sensores. Cuanto más larga es la distancia entre los sensores de imagen, más precisa es la medición. En la Figura 3.2 podemos ver un ejemplo donde para un punto

común, conociendo el ángulo 'a' y 'b', y la distancia entre los sensores, se puede calcular la distancia al punto común.

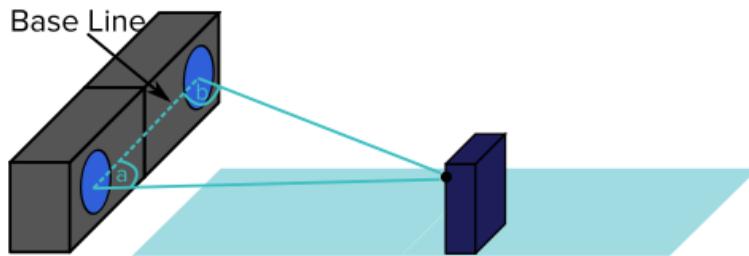


Figura 3.2: Ejemplo de identificación de un punto común con tecnología estéreo activa. Figura extraída de Whyte (2022).

Con la tecnología estéreo activa, es importante hacer coincidir las mismas características en ambas imágenes. Una forma de mejorar esto es proyectar patrones de puntos, cada sensor de profundidad puede identificar cada punto único. De esta forma, la proyección de puntos facilita la coincidencia de características y permite medir objetos sin características, como paredes blancas.

El D435 utiliza la combinación de estéreo activo y pasivo, por lo que puede medir objetos más cercanos sin rasgos distintivos y objetos más lejanos.

4. Propuesta de solución

El objetivo del TFG consiste en desarrollar un sistema que reconstruya un cuerpo humano completo, utilizando como sensor para la adquisición de los datos el Intel RealSense D435 y el sistema embebido Jetson Nano como computador que procesará los datos en la fase de registro. Por ello, en este capítulo detallaremos cuales han sido los pasos a seguir para conseguir un sistema que funcione y sea capaz de llevar a cabo esta reconstrucción.

En la Figura 4.1 podemos ver un diagrama de la implementación de los módulos del sistema, desde la fase de adquisición, pasando por la fase de registro, hasta la fase de análisis con los resultados finales.

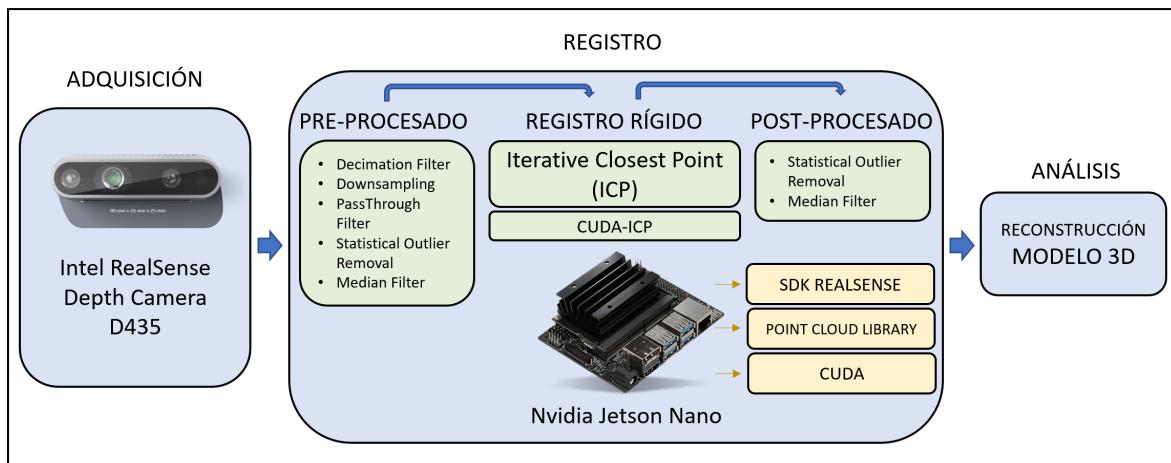


Figura 4.1: Diagrama de la propuesta e implementación de los módulos del sistema.

A continuación, durante este capítulo desarrollaremos cada una de las partes del diagrama explicando las cosas implementadas y el funcionamiento de cada uno de los módulos.

4.1. Adquisición de datos

Nuestro sistema de reconocimiento 3D consta del sensor Intel RealSense D435 conectado directamente a través de un puerto USB 3.0 al dispositivo Nvidia Jetson Nano.

Durante la fase de adquisición, el objetivo será realizar capturas RGBD a un modelo a través del sensor Intel RealSense D435. Para ello, el sensor deberá estar fijo y el modelo deberá estar situado en el centro de la imagen para una mayor nitidez en las capturas. El sensor realizará capturas constantemente y consecutivamente, mientras el modelo rota 360° poco a poco. De esta forma, obtendremos muchas capturas desde distintos ángulos del modelo que posteriormente se procesarán para ser alineadas y reconstruir el modelo completamente en 3D.

Por tanto, será importante tener en cuenta la frecuencia de capturación del sensor, ya que cuanto mayor frecuencia, más rápido capturará al modelo, y menor será el ángulo de rotación entre cada una de las capturas. Esto quiere decir que la distancia entre los puntos de dos capturas consecutivas será muy pequeña, y facilitará la tarea del registro con el algoritmo ICP.

El sensor Intel RealSense D435 es capaz de capturar a una frecuencia de hasta 30Hz en modo RGBD. Realmente esto es más que suficiente, ya que 30Hz quiere decir que es capaz de capturar 30 imágenes en un segundo. Hemos definido que cuanto más rápido se realicen las capturas es mejor, pero tanta velocidad realmente puede llegar a ser perjudicial, debido a que a esa frecuencia pueden llegar a realizarse cientos de capturas hasta que el modelo haya terminado de rotar 360°, haciendo que aumente el peso de trabajo a procesar debido a la enorme cantidad de datos.

Por ello, es necesario encontrar un punto medio. Como la frecuencia del sensor es mayor a las capturas que realmente queremos obtener por segundo, limitaremos la velocidad a la que se realizan las capturas introduciendo por parámetro un valor que indique cada cuanto realizar la captura.

Inicialmente y para testear, está bien separar la fase de adquisición de todas las capturas de la fase de registro. Sin embargo, se ha añadido una opción para funcionar de forma paralela e ir obteniendo el resultado a la vez que se realiza la adquisición total del modelo. Es decir, una vez se hayan realizado la primera y segunda captura, estas serán enviadas para procesar el registro en otro hilo y, de forma concurrente, seguimos realizando la adquisición de nuevas capturas mientras el modelo está rotando.

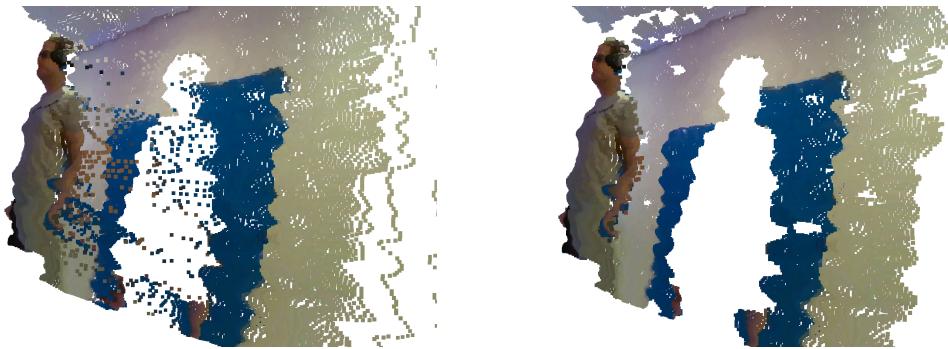
4.2. Pre-procesado

Tras obtener la captura del modelo en la fase de adquisición, la captura se envía a la fase de pre-procesado donde se le aplicarán los filtros ya mencionados en apartados anteriores para reducir ruido y facilitar el procesamiento en la fase de registro.

El primer filtro a aplicar será el filtro diezmado (Decimal Filter). Este filtro realmente viene aplicado desde la fase de adquisición debido a que el filtro está integrado en la librería de RealSense desde la cual realizamos la captura, y se utiliza en el mismo proceso donde guardamos los datos y los enviamos al pre-procesado. El filtro finalmente lo que consigue es una reducción de muestreo, como un downsampling, reduciendo el peso de la nube de puntos sin perjudicar demasiado la calidad, y facilitando las operaciones que se realizan dentro del algoritmo ICP que se procesará posteriormente.

Seguidamente al filtro de reducción de muestreo, se aplica el filtro “Statistical Outlier Removal” que nos permitirá deshacernos de los puntos atípicos que se encuentren más alejados del modelo en sí. Estos puntos normalmente son ruido por el borde del modelo que se genera en la fase de adquisición. En la Figura 4.2 podemos ver un ejemplo de aplicación de este filtro sobre un modelo 3D.

Por último, se aplica el filtro de la mediana, que permite homogeneizar el modelo 3D reduciendo el ruido que hay en el propio modelo, aplanandolo ligeramente.



(a) Nube de puntos original. (b) Nube de puntos con filtro aplicado.

Figura 4.2: Ejemplo de filtro Statistical Outlier Removal aplicado.

4.3. Registro

En la fase de registro es donde conseguiremos reconstruir, poco a poco, el modelo en 3D por completo. Para esta fase, nuestra solución consta de dos implementaciones sobre las que se han hecho las pruebas. En una de ellas se utiliza el algoritmo ICP de la librería PCL, y en la otra se utiliza CUDA-ICP, que se trata de una implementación en CUDA sobre el algoritmo ICP de la librería PCL también. No obstante, el uso de ambas implementaciones es el mismo, ambas reciben un modelo y una escena como entrada, y devuelven la matriz de transformación a aplicar sobre la escena.

Cabe destacar que no se hace uso de ningún algoritmo de grano grueso para un primer aproximamiento de la escena al modelo, como se ha explicado en la sección 2.1. Aunque es muy común utilizar RANSAC-ICP para estos procesos de reconstrucción de modelos 3D, el caso de nuestro proyecto se encuentra muy bien definido en un entorno y unas características concretas que hacen que no sea necesario utilizar estos métodos como apoyo. Debido a que la fase de adquisición es lo suficientemente rápida, la distancia entre cada par consecutivo de capturas es muy pequeña, lo cual nos permite procesar el método ICP sin necesidad de aproximar la escena previamente.

Por tanto, esta fase recibe dos nubes de puntos ya pre-procesadas, una de ellas será el modelo, y la otra será la escena. En una primera iteración de esta solución, el algoritmo recibirá la captura 1 como la nube de puntos modelo, y la captura 2 como la nube de puntos escena. Estas nubes de puntos son procesadas, obteniendo la matriz de transformación que hay que aplicar sobre la escena para que esta quede alineada con el modelo. Seguidamente, transformamos la escena para que quede alineada con el modelo y juntamos ambas nubes de puntos, consiguiendo así una única nube de puntos que tenga la información de las capturas 1 y 2. Nos guardamos la matriz de transformación obtenida para aplicarselo a las futuras escenas como matriz de transformación inicial.

En una segunda iteración de la solución, esta fase recibirá una nueva nube de puntos perteneciente a la captura 3, que será la nueva escena. Por tanto, tendremos la nube de puntos modelo que constará de las capturas 1 y 2, y la nube de puntos escena que corresponde con la captura 3. Previamente a procesar las nubes de puntos por el algoritmo ICP, se aplicará la matriz de transformación de la iteración anterior como transformación inicial para la captura

3, haciendo así que esta nube de puntos escena quede más próxima al modelo. Ahora sí, las nubes de puntos serán procesadas por el algoritmo y obtendremos una nueva matriz de transformación que habrá que aplicar sobre la captura 3 para que quede alineada con el modelo. Aplicaremos dicha matriz de transformación y nos la volveremos a guardar para la siguiente iteración, aunque realmente lo que haremos será multiplicarla con la matriz de transformación que ya teníamos de la iteración anterior, haciendo así que la futura captura 4 utilice como matriz de transformación el resultado de todas las matrices de transformación de las iteraciones anteriores. En el diagrama de la Figura 4.3 podemos ver este proceso con un esquema.

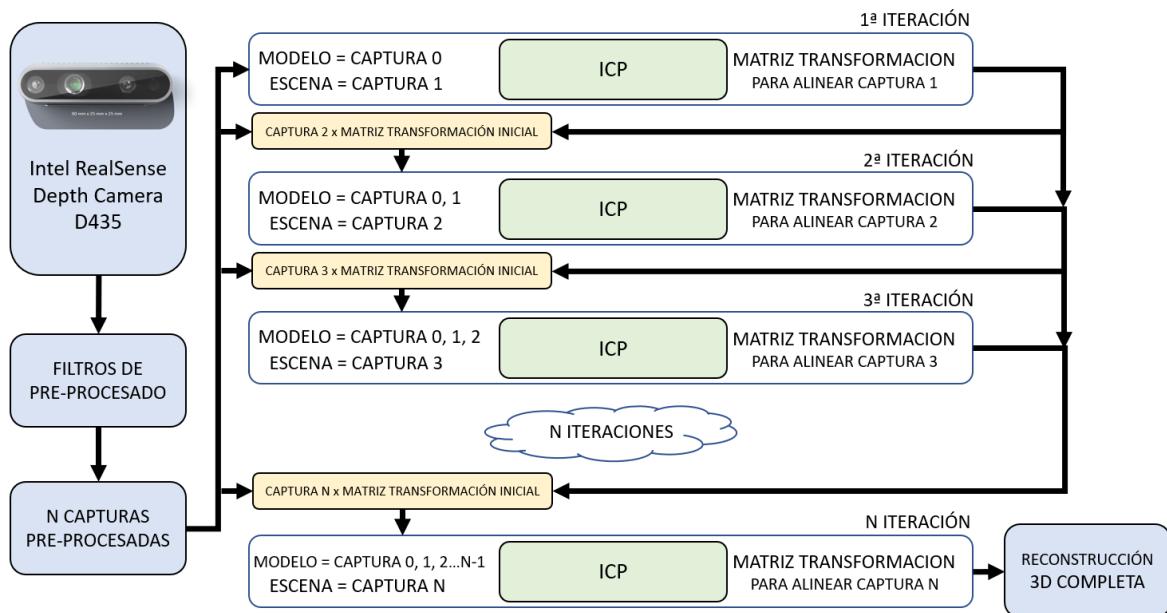


Figura 4.3: Diagrama de reconstrucción 3D acumulando matrices de transformación para cada nueva iteración.

4.3.1. Posible refinamiento con CPD

Como propuesta general, se puede hacer uso de un método de registro no rígido para corregir errores, como los producidos por el movimiento del cuerpo. Concretamente se ha probado una pequeña implementación con el algoritmo CPD explicado previamente en el apartado 2.1.2. La ventaja de este algoritmo es la posibilidad de utilizar el registro no rígido para corregir los errores de movimiento que se puedan haber producido durante la rotación 360° del cuerpo humano durante la fase de adquisición, ya que el cuerpo puede deformarse entre las capturas. El algoritmo ha sido implementado de una manera similar a los otros dos, sin embargo no se contempla como solución final debido al largo tiempo de cómputo que emplea, no pudiendo proporcionar un resultado final.

4.4. Post-procesado

Tras finalizar la fase de registro y obtener el modelo final alineado y reconstruido, opcionalmente podemos utilizar la fase de post-procesado para aplicar unos filtros con la finalidad de reducir un poco el posible ruido que se haya podido generar durante la fase de registro.

Estos filtros serán el filtro “Statistical Outlier Removal” que se deshará de puntos atípicos lejanos a los bordes del modelo y el filtro de mediana, para aplanar y homogeneizar el modelo en sí.

4.5. Análisis

En esta última fase nos encontramos con una única nube de puntos, resultado de alinear todas las escenas con la primera nube de puntos de la fase de adquisición.

Esta nube de puntos la encontramos en un fichero con formato “.ply” que puede ser abierta por programas para visualizar nubes de puntos de modelos 3D como lo es MeshLab. En la Figura 4.4 podemos ver un ejemplo de resultado en la fase de análisis.

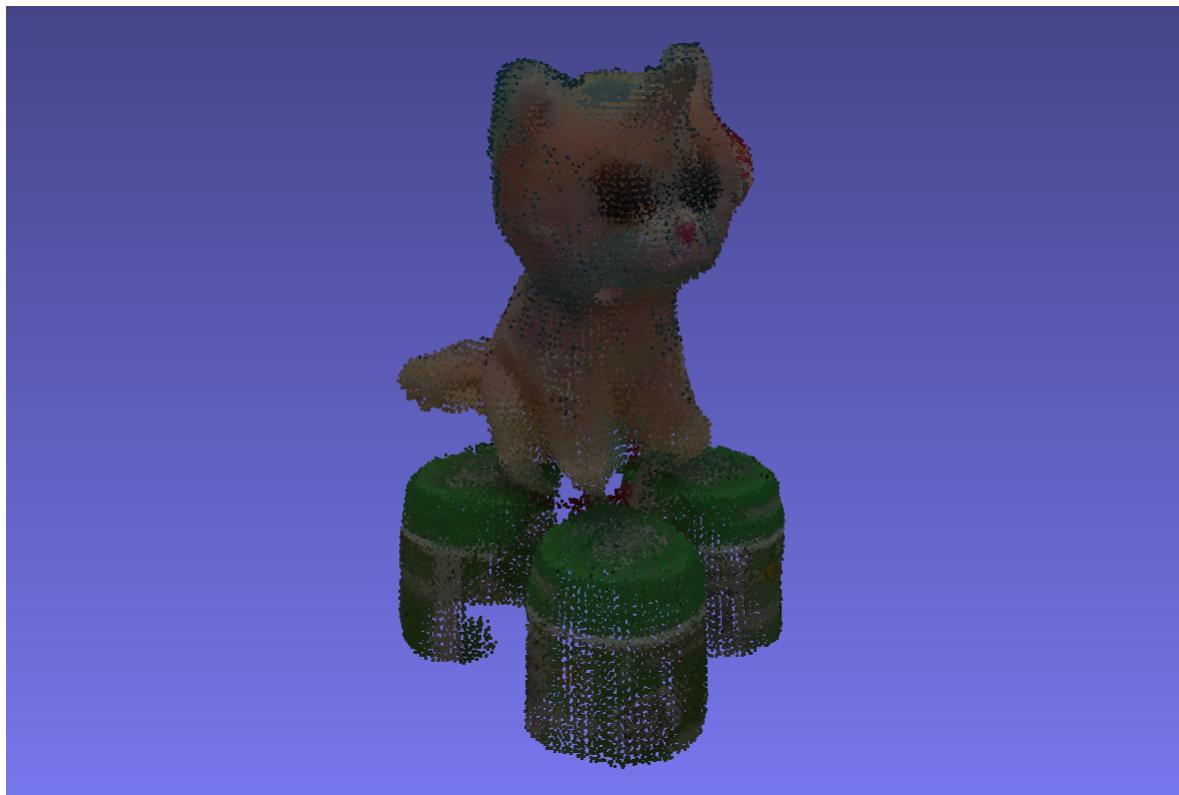


Figura 4.4: Ejemplo de modelo 3D en la fase de análisis.

Esta fase trata básicamente de visualizar el resultado para analizar posibles mejoras en la experimentación.

5. Solución del proyecto en Jetson Nano

En este capítulo detallaremos como se ha llevado a cabo la propuesta explicada en el capítulo anterior. Aunque existen muchas soluciones, librerías y proyectos compartidos en Python que podrían facilitar el desarrollo de este proyecto, nos hemos decantado por utilizar C++, debido a que estoy algo más familiarizado con esta tecnología y destaca por ser más rápida, con mejor rendimiento. Todo eso a pesar de haber desarrollado un código bastante simple limitándonos a utilizar librerías existentes, C++ ofrece la posibilidad de meterse más a bajo nivel para poder optimizar mejor el código, entrando más en el grano fino. Por lo que C++ nos ha parecido una propuesta inicial interesante.

5.1. Integración del Software. Dependencias

En primer lugar, explicaremos todo el software que ha sido necesario instalar, tanto para las dependencias con el proyecto C++ como el software necesario para hacer que el dispositivo Nvidia Jetson Nano funcione a la perfección con el sensor Intel RealSense D435.

5.1.1. Sistema Operativo. Ubuntu 18.04

La fuente de este sistema operativo se puede obtener desde la página oficial de Nvidia, a través de JetPack SDK donde nos guían para llevar a cabo la instalación. NVIDIA JetPack SDK es la solución más completa para un entorno de desarrollo completo en el desarrollo acelerado por hardware. Todos los módulos y kits para desarrolladores de Jetson Nano son compatibles con JetPack SDK. JetPack SDK incluye Jetson Linux Driver Package con cargador de arranque, kernel de Linux, entorno de escritorio Ubuntu y un conjunto completo de bibliotecas para la aceleración de computación GPU, multimedia, gráficos y visión por computadora.

El último SDK disponible para el dispositivo Nvidia Jetson Nano es el JetPack 4.6.1, cuyo sistema operativo es Ubuntu 18.04 con versión CUDA 10.2. El SDK JetPack 5.0 utiliza el sistema operativo de Ubuntu 20.04 con versión CUDA 11.4, pero actualmente no está disponible para Nvidia Jetson Nano, solo para modelos superiores.

5.1.2. CMake. Compilador C++

Como se ha mencionado anteriormente, hemos decidido elaborar el proyecto en el lenguaje C++. Para ello, se ha construido un proyecto utilizando CMake.

CMake es una familia de herramientas diseñada para construir, probar y empaquetar software. Se utiliza para controlar el proceso de compilación del software usando ficheros de configuración sencillos e independientes de la plataforma. Gracias al uso de CMake, el proyecto queda fácilmente configurado, encargándose de enlazar todas las dependencias que el

proyecto necesite. Por tanto, será necesario tener instaladas todas las herramientas necesarias para la correcta ejecución de CMake.

Además, será necesario tener instalado el compilador de C++ para poder compilar el proyecto. El compilador instalado en el dispositivo Nvidia Jetson Nano GCC versión Ubuntu/Linaro 7.5.0-3ubunt1 18.04.

5.1.3. SDK RealSense

Para la instalación del SDK de RealSense se ha utilizado un script que ha facilitado la instalación del mismo. En el GitHub de installLibrealsense podemos encontrar un script que se encarga de añadir al repositorio del gestor de paquetes las fuentes necesarias y de instalar todos los paquetes necesarios de forma automática. Por tanto, tras clonar dicho proyecto y ejecutar el script “./installLibrealsense.sh” tendremos instalado por completo el SDK de RealSense.

Una vez instalado el software, podremos utilizar la aplicación RealSense Viewer para visualizar la cámara tanto en 2D como en 3D. También podremos compilar sin problemas añadiendo las dependencias necesarias en el fichero CMakeLists.txt, como se muestra en el Código 5.1.

Código 5.1: Dependencia CMakeLists: RealSense Library

```

1 # RealSense Library
2 find_package(realsense2 REQUIRED)
3 include_directories(include ${realsense_INCLUDE_DIR})
4 target_include_directories(main PRIVATE ${realsense_INCLUDE_DIR})
5 target_link_libraries(main ${realsense2_LIBRARY})

```

5.1.4. Point Cloud Library (PCL)

PCL es una biblioteca de código abierto de algoritmos para tareas de procesamiento de nubes de puntos y procesamiento de geometría 3D, como ocurre en la visión tridimensional por computadora. La biblioteca contiene algoritmos para filtrado, estimación de características, reconstrucción de superficies, registro 3D, ajuste de modelos, reconocimiento de objetos y segmentación. Cada módulo se implementa como una biblioteca más pequeña que se puede compilar por separado. PCL tiene su propio formato de datos para almacenar nubes de puntos: PCD (Point Cloud Data), pero también permite cargar y guardar conjuntos de datos en muchos otros formatos, como PLY (Polygon File Format). También está escrito en C++ y publicado bajo la licencia BSD.

Esta librería es muy interesante y la utilizamos tanto para aplicar filtros de pre-procesado y post-procesado, como para el método de registro ICP.

Además de tener instalada la librería, debe estar incluida como dependencia en el fichero CMakeLists.txt del proyecto de la forma que se muestra en el Código 5.2.

Código 5.2: Dependencia CMakeLists: Point Cloud Library

```

1 # Point Cloud Library
2 find_package(PCL 1.8 REQUIRED)
3 include_directories(${PCL_INCLUDE_DIRS})
4 link_directories(${PCL_LIBRARY_DIRS})

```

```

5   add_definitions(${PCL_DEFINITIONS})
6   target_link_libraries(main ${PCL_COMMON_LIBRARIES} ${PCL_IO_LIBRARIES} ${←
    ↪ PCL_FILTERS_LIBRARIES})

```

5.1.5. CUDA. CUDA-ICP.

Gracias a haber instalado el sistema operativo con JetPack SDK, toda la librería de CUDA 10.2 ya viene instalada por defecto en el sistema operativo, por lo que solamente será necesario incluir como dependencia en el fichero CMakeLists.txt del proyecto de la forma que se muestra en el Código 5.3.

Código 5.3: Dependencia CMakeLists: CUDA

```

1 # CUDA Library
2 find_package(CUDA 10.2 REQUIRED)
3 include_directories(${CUDA_INCLUDE_DIRS})
4 target_link_libraries(main ${CUDA_LIBRARIES})

```

Además, encontramos un repositorio oficial de Nvidia que nos ofrece una implementación de ICP en CUDA. Este repositorio llamado CUDA-PCL contiene diversas implementaciones en CUDA de distintos algoritmos de la librería PCL, entre ellos CUDA-ICP.

Nos hemos descargado de ahí la implementación ICP y la hemos importado en nuestro proyecto con las líneas en el fichero CMakeLists.txt que se muestran en el Código 5.4.

Código 5.4: Dependencia CMakeLists: CUDA-ICP

```

1 # CUDA ICP Local Library
2 target_include_directories(main PRIVATE ${CMAKE_CURRENT_SOURCE_DIR})
3 add_library(libcudaicp SHARED IMPORTED)
4 set_target_properties(libcudaicp PROPERTIES IMPORTED_LOCATION ${←
    ↪ CMAKE_CURRENT_SOURCE_DIR}/lib/libcudaicp.so)
5 target_link_libraries(main libcudaicp)

```

5.1.6. JsonCpp

JsonCpp es una librería que permite leer y escribir fácilmente ficheros Json en C++. Esta librería no cumple ninguna función esencial para el proceso de reconstrucción de un modelo 3D, pero es muy útil ya que usamos un fichero .json donde definimos todos los parámetros que la aplicación debe tener en cuenta a la hora de ejecutarse. Por tanto es una dependencia necesaria si se pretende compilar el proyecto tal y como se encuentra ahora mismo, ya que los parámetros se leen de un fichero .json.

Además de tener instalada la librería, debe estar incluida como dependencia en el fichero CMakeLists.txt del proyecto de la forma que se muestra en el Código 5.5.

Código 5.5: Dependencia CMakeLists: JsonCpp

```

1 # JsonCpp Library
2 find_package(PkgConfig REQUIRED)
3 pkg_check_modules(JSONCPP jsoncpp)
4 link_libraries(${JSONCPP_LIBRARIES})

```

```

5     target_link_libraries(main ${JSONCPP_LIBRARIES})
6     configure_file(parameters.json parameters.json COPYONLY)

```

5.2. Desarrollo de la solución

Tras la instalación de todo el software necesario, explicaremos detalladamente los fragmentos más relevantes del código desarrollado, siguiendo el esquema explicado en el capítulo 4, Figura 4.1.

El programa consiste en una aplicación de consola con un pequeño menú numerado con las distintas opciones a realizar, desde la ejecución total de la reconstrucción pasando por todas las fases automáticamente tanto para la implementación de ICP de PCL como para la implementación de CUDA-ICP, hasta la ejecución de cada una de las fases por separado, con finalidad de analizar y experimentar mejor con los resultados.

Debido a que la ejecución del algoritmo conlleva variables que no son fijas para distintos casos de uso, se ha creado un fichero json para poder personalizar los parámetros de ejecución. A la hora de lanzar el programa, lo primero que se hace es leer el fichero json llamado “parameters.json” que contiene la siguiente información que se muestra en el fragmento de Código 5.6.

Código 5.6: Parámetros de la aplicación

```

1   {
2       "DataPath" : "/home/edgar/tfg/Code Solutions/JetsonNano/.data/",
3       "RealSenseCaptureTotalTime_ms" : 27500,
4       "RealSenseCaptureTimingBetweenCaptures_ms" : 500,
5       "NumberOfCaptureFiles" : 52,
6       "ResizeCenterDistance_m" : 0.63,
7       "ResizeCaptureBottom_m" : 0.47,
8       "ResizeCaptureTop_m" : 0.25,
9       "ResizeCaptureLeft_m" : 0.5,
10      "ResizeCaptureRight_m" : 0.5,
11      "ResizeCaptureBehind_m" : 0.35,
12      "ResizeCaptureFront_m" : 0.4,
13      "ICP_Iterations" : 200,
14      "ICP_MaxCorrespondenceDistance" : 0.015
15  }

```

“DataPath” especifica la ruta donde se van a guardar y tratar toda la información que sea necesaria. Por ejemplo, si ejecutamos la fase de adquisición por separado, se guardarán ahí los ficheros para poder ser procesados posteriormente por otra fase. El resto de parámetros los iremos viendo en este capítulo a medida que se va explicando el código.

5.2.1. Adquisición de datos

Para la fase de adquisición necesitaremos conectarnos a la cámara a través del SDK de RealSense. Para ello, inicializaremos las variables que podemos ver a continuación en el Código 5.7. “rs2::pipeline” es la clase que se encarga de gestionar el sensor y que nos permitirá recibir los frames que está captando. “rs2::pointcloud” nos permitirá gestionar los frames que

recibimos del sensor para convertirlos a nubes de puntos y de esta manera poder procesar el algoritmo posteriormente. Por último, “decimation_filter” se encargará de aplicar el filtro de diezmado a los frames de profundidad capturados por el sensor antes incluso de convertirlos a nubes de puntos. Como se ha mencionado en el Capítulo 4, este es el único filtro que se ejecuta antes de la fase de pre-procesamiento, ya que se hace en el momento de la adquisición del frame.

Además de la inicialización también podemos observar un bucle inicial que se encarga de crear autoexposición en la imagen, de forma que las primeras capturas se realicen correctamente y no aparezcan oscuras o con ruido.

Código 5.7: Inicialización del sensor RealSense D435 para la adquisición de datos

```

1   rs2::decimation_filter decimation_filter;
2   rs2::pointcloud pointcloud;
3   rs2::pipeline pipe;
4   pipe.start();
5
6   std::cout << "Capturando 30 frames para autoexposición..." << std::endl;
7   for (auto i = 0; i < 30; i++) pipe.wait_for_frames();

```

A continuación, en el Código 5.8 tenemos un bucle cuya duración estará establecida por parámetro a través de la variable “RealSenseCaptureTotalTime_ms”. Con la variable “RealSenseCaptureTimingBetweenCaptures_ms” podemos establecer el tiempo de espera entre frame y frame que también estará establecido por parámetro.

En este bucle guardaremos los frames que recibe la cámara, aplicando el filtro de diezmado y mapeandolos a nube de puntos con RGB. Una vez obtenemos la nube de puntos, se guarda en un vector que será tratado posteriormente.

Código 5.8: Bucle para adquisición de los datos con el sensor RealSense D435

```

1   while (time < RealSenseCaptureTotalTime_ms)
2   {
3       std::this_thread::sleep_for(std::chrono::milliseconds(←
4           ↗ RealSenseCaptureTimingBetweenCaptures_ms));
5
6       auto fs = pipe.wait_for_frames();
7       auto depth = fs.get_depth_frame();
8       auto color = fs.get_color_frame();
9
10      depth.keep();
11      color.keep();
12      depths.push_back(depth);
13      colors.push_back(color);
14      pointcloud.map_to(colors[i]);
15      depths[i] = decimation_filter.process(depths[i]);
16      depths[i].keep();
17      points.push_back(pointcloud.calculate(depths[i]));
18      points[i].keep();
19
20      i++;
}

```

Finalmente, tenemos dos maneras de continuar con los datos. Por una parte, en el fragmento de Código 5.9 podemos ver que guarda todas las nubes de puntos almacenadas en el vector en fichero “.ply” para su posterior tratamiento. Esto es útil cuando queremos separar la ejecución del programa en sus diferentes fases, ya que nos permite analizar la adquisición antes de continuar con el procesamiento de los datos.

Código 5.9: Guardar las nubes de puntos en un fichero de datos

```

1   for (i = 0; i < depths.size(); i++)
2   {
3       std::stringstream filename;
4       filename << DataPath << "adquisitionFromRealSense/" << i << ".ply";
5       points[i].export_to_ply(filename.str(), colors[i]);
6   }

```

Sin embargo, cuando sabemos que el método de adquisición funciona correctamente, tenemos la alternativa de guardar las nubes de puntos directamente en memoria convirtiéndolas en nubes de puntos compatibles con la librería PCL como podemos ver en el Código 5.10.

Código 5.10: Conversión de una nube de puntos de la librería RealSense a una nube de puntos de PCL

```

1   for (i = 0; i < depths.size(); i++)
2   {
3       pcl::PointCloud<pcl::PointXYZRGB>::Ptr cloud_in = parseToPCLRGB(points[i
4           ↪ ], colors[i]);
5       applyPreprocessingFilterTo(cloud_in);
}

```

5.2.2. Pre-procesado

Como se ha visto a lo largo del TFG, a pesar de estar enfocado en la reconstrucción de cuerpos humanos, también se han procesado objetos debido a su mayor facilidad ya que son cuerpos rígidos. La adquisición de datos de estos objetos han sido tomadas desde un ángulo superior, por lo que para este tipo de datos se utiliza una matriz de rotación de 30° en el eje X para que la nube de puntos quede bien alineada inicialmente. En la subfigura 5.1a podemos ver la nube de puntos original y en la subfigura 5.1b la misma nube de puntos perfectamente orientada respecto al eje X. Para ello se usa la matriz de rotación que se muestra en el Código 5.11 sobre la nube de puntos adquirida en la fase de adquisición.

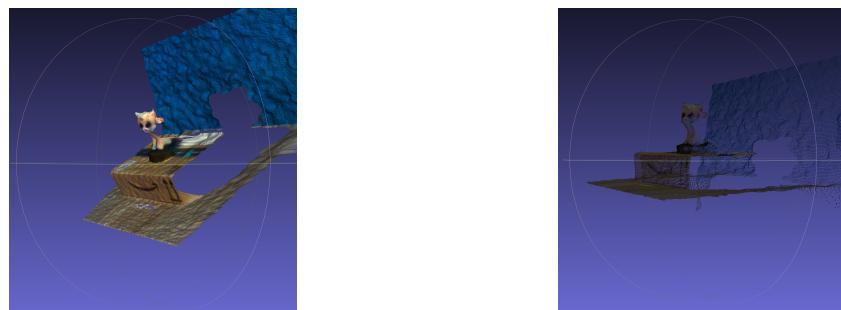
Código 5.11: Rotación en fase de pre-procesamiento de la nube de puntos

```

1 Eigen::Matrix4f transformation_matrix = Eigen::Matrix4f::Identity();
2 double theta = degreesToRadians(-30);
3 transformation_matrix(1, 1) = cos(theta);
4 transformation_matrix(1, 2) = -sin(theta);
5 transformation_matrix(2, 1) = sin(theta);
6 transformation_matrix(2, 2) = cos(theta);
7 pcl::transformPointCloud(*cloud, *cloud, transformation_matrix);

```

A continuación, veremos la implementación de los diferentes filtros utilizados en el pre-



(a) Nube de puntos original con un ángulo de 30° desde arriba.

(b) Nube de puntos rotada.

Figura 5.1: Ejemplo de rotación en el pre-procesamiento de datos para dejar la nube de puntos alineada con el eje X.

procesamiento de los datos. En el Código 5.12 podemos ver el filtro PassThrough que nos permite limitar la nube de puntos, eliminando todos los puntos de la escena que no corresponden con el cuerpo al que queremos procesar el algoritmo ICP. En este filtro se utilizan los parámetros del Json para establecer los límites del tamaño del cuerpo que se está procesando.

Los parámetros “ResizeCaptureLeft_m” y “ResizeCaptureRight_m” establecen el ancho del cuerpo a escanear en el eje X; los parámetros “ResizeCaptureBottom_m” y “ResizeCaptureTop_m” indican el alto del cuerpo en el eje Y; y por último los parámetros “ResizeCaptureBehind_m” y “ResizeCaptureFront_m” indican la profundidad del cuerpo en el eje Z. Para este último eje, hay que tener en cuenta el parámetro “ResizeCenterDistance_m” que indica la distancia desde la cámara hasta el cuerpo.

Código 5.12: Aplicación del Filtro PassThrough

```

1  pcl::PassThrough<pcl::PointXYZRGB> pass;
2  pass.setInputCloud(cloud);
3  pass.setFilterFieldName("x");
4  pass.setFilterLimits(-ResizeCaptureLeft_m, ResizeCaptureRight_m);
5  pass.filter(*cloud);
6  pass.setInputCloud(cloud);
7  pass.setFilterFieldName("y");
8  pass.setFilterLimits(-ResizeCaptureBottom_m, ResizeCaptureTop_m);
9  pass.filter(*cloud);
10 pass.setInputCloud(cloud);
11 pass.setFilterFieldName("z");
12 pass.setFilterLimits(ResizeCenterDistance_m - ResizeCaptureBehind_m, ↵
    ↵ ResizeCenterDistance_m + ResizeCaptureFront_m);
13 pass.filter(*cloud);

```

En el Código 5.13 podemos ver la aplicación del filtro Statistical Outlier Removal que elimina todos los puntos atípicos que se encuentran alejados del cuerpo que queremos procesar.

Código 5.13: Aplicación del Statistical Outlier Removal

```

1  pcl::StatisticalOutlierRemoval<pcl::PointXYZRGB> sor;
2  sor.setInputCloud(cloud);

```

```

3     sor.setMeanK(50);
4     sor.setStddevMulThresh(1);
5     sor.filter(*cloud);

```

Por último, aplicamos el filtro de la Mediana que vemos en el Código 5.14 para homogeneizar los puntos del cuerpo restantes en la nube de puntos.

Código 5.14: Aplicación del Filtro Mediana

```

1   pcl::MedianFilter<pcl::PointXYZRGB> mf;
2   mf.setInputCloud(cloud);
3   mf.setWindowSize(10);
4   mf.setMaxAllowedMovement(0.01);
5   mf.applyFilter(*cloud);

```

Como podemos observar, todos estos filtros se utilizan desde la librería PCL.

5.2.3. Registro

En la fase de registro comienza el procesamiento de los datos para la reconstrucción en 3D del cuerpo. En el Código 5.15 podemos ver la aplicación de la matriz de transformación inicial para todas las nubes de puntos menos la primera, ya que esta parte de 0 respecto al modelo.

Código 5.15: Matriz de transformación inicial

```

1   if (i != 1)
2   {
3       pcl::PointCloud<pcl::PointXYZRGB>::Ptr sourcePointCloud_transformed(new ←
4           → pcl::PointCloud<pcl::PointXYZRGB>);
5       pcl::transformPointCloud(*sourcePointCloud, *←
6           → sourcePointCloud_transformed, initialTransformationMatrix);
7       sourcePointCloud = sourcePointCloud_transformed;
}

```

Seguidamente, se procederá a efectuar el algoritmo ICP con las nubes de puntos fuente y destino, es decir, las nubes de puntos escena y modelo, respectivamente. En el Código 5.16 podemos ver la ejecución del algoritmo implementada por la librería PCL.

Código 5.16: Preparación y llamada al algoritmo ICP de Point Cloud Library

```

1   pcl::IterativeClosestPoint<pcl::PointXYZRGB, pcl::PointXYZRGB> icp;
2   icp.setMaximumIterations(ICP_Iterations);
3   icp.setMaxCorrespondenceDistance(ICP_MaxCorrespondenceDistance);
4   icp.setInputSource(sourcePointCloud);
5   icp.setInputTarget(targetPointCloud);
6   icp.align(*sourcePointCloud);
7   return icp.getFinalTransformation();

```

El resultado de esta ejecución consistirá en la transformación de la nube de puntos escena (fuente), siendo alineada con el modelo (destino), y la obtención de la matriz de transformación que ha conseguido este cambio. Una vez obtenida la matriz de transformación, nos la guardamos acumulándola con todas las anteriores para las siguientes iteraciones con las

escenas restantes, como podemos ver en el Código 5.17. Además, después de guardarnos la matriz de transformación sumamos los puntos de la nube de puntos modelo con la nube de puntos escena alineada, obteniendo un nuevo modelo que, poco a poco, se va completando.

Código 5.17: Resultado del algoritmo ICP. Matriz de Transformación y nube de puntos resultante

```

1   if (i == 1)
2   {
3       initialTransformationMatrix = icpTransformationMatrix;
4   }
5   else
6   {
7       initialTransformationMatrix = icpTransformationMatrix * ↵
8           ↪ initialTransformationMatrix;
9   }
*targetPointCloud = (*targetPointCloud) + (*sourcePointCloud);

```

Por último, podemos ver la ejecución del algoritmo ICP con CUDA en el Código 5.18. Lo primero que se hace es preparar los datos, pues no se envía a tratar directamente a la nube de datos de PCL. Se obtiene la información de los puntos en un listado de valores “float” y nos apuntamos la cantidad de puntos en cada nube para poder reservar la memoria suficiente para almacenar todos los puntos. Seguidamente, reservamos memoria para la matriz de transformación y ejecutamos el algoritmo enviandole todos los datos. Este algoritmo nos devolverá la matriz rellenada con la información sobre la transformación necesaria para alinear la escena con el modelo. Finalmente, liberamos la memoria reservada para este proceso y se devuelve dicha matriz para continuar con el mismo procedimiento que se hace con PCL.

Código 5.18: Preparación y llamada al algoritmo de CUDA-ICP

```

1   int nP = pcl_cloud_source->size();
2   int nQ = pcl_cloud_target->size();
3   float* nPdata = (float*)pcl_cloud_source->points.data();
4   float* nQdata = (float*)pcl_cloud_target->points.data();
5
6   void* cudaMatrix = NULL;
7   cudaMatrix = malloc(sizeof(float) * 4 * 4);
8   memset(cudaMatrix, 0, sizeof(float) * 4 * 4);
9   cudaStream_t stream = NULL;
10  cudaStreamCreate(&stream);
11
12  float* PUVM = NULL;
13  cudaMallocManaged(&PUVM, sizeof(float) * 4 * nP, cudaMemAttachHost);
14  cudaStreamAttachMemAsync(stream, PUVM);
15  cudaMemcpyAsync(PUVM, nPdata, sizeof(float) * 4 * nP, ↵
16      ↪ cudaMemcpyHostToDevice, stream);
17  float* QUVM = NULL;
18  cudaMallocManaged(&QUVM, sizeof(float) * 4 * nQ, cudaMemAttachHost);
19  cudaStreamAttachMemAsync(stream, QUVM);
20  cudaMemcpyAsync(QUVM, nQdata, sizeof(float) * 4 * nQ, ↵
21      ↪ cudaMemcpyHostToDevice, stream);
22  cudaStreamSynchronize(stream);

```

```
22     cudaICP cudaIcp(nP, nQ, stream);
23     cudaIcp.icp((float*)PUVM, nP, (float*)QUVM, nQ, ICP_Iterations, 1e-20, ↪
24         ↪ cudaMatrix, stream);
25
26     Eigen::Matrix4f matrix_icp = Eigen::Matrix4f::Identity();
27     memcpy(matrix_icp.data(), cudaMatrix, sizeof(float) * 4 * 4);
28     transformation_matrix = matrix_icp;
29
30     cudaFree(PUVM);
31     cudaFree(QUVM);
32     free(cudaMatrix);
33
34     return matrix_icp;
```

En la fase de post-procesamiento se vuelven a aplicar los mismos filtros ya explicados anteriormente y finalmente se guarda la nube de puntos resultante en un fichero “.ply” para su análisis.

6. Experimentación

En este capítulo se van a explicar los resultados de la experimentación para la reconstrucción de cuerpos humanos en 3D. En primer lugar se muestran las herramientas utilizadas para el análisis de los resultados; seguidamente como experimentación inicial se aplicará el método de registro sobre un modelo y una única escena, para analizar los tiempos y el resultado visual; a continuación experimentaremos con la reconstrucción 3D completa; y finalmente analizaremos los resultados visualmente con las herramientas ya mencionadas.

Durante la fase de experimentación se han estado usando distintos objetos para la reconstrucción 3D. Esto es debido a que la experimentación se enfoca en el método de registro rígido, y el cuerpo humano es deformable por lo que es muy complicado realizar pruebas rígidas en él. Por ello, se han utilizado algunos objetos como experimentación inicial para finalmente realizar algunas pruebas con el cuerpo humano. En la Figura 6.1 podemos ver una foto del objeto utilizado para estas pruebas.



Figura 6.1: Objeto de pruebas para reconstrucción 3D.

6.1. Herramientas para visualizar las nubes de puntos

Intel RealSense Viewer es una herramienta oficial de Intel para los sensores RealSense. Con esta aplicación se puede visualizar la adquisición de la cámara en tiempo real, como se puede ver en la Figura 6.2. Tiene muchas herramientas y configuraciones que pueden ser útiles para experimentar con la cámara.

Otra herramienta interesante es MeshLab. Meshlab un software de código abierto para el procesamiento y edición de mallas triangulares no estructuradas en 3D (Ranzuglia y cols., 2013) (Cignoni y cols., 2008). Este sistema tiene varias herramientas para la edición, optimización, inspección y renderización de mallas. Ha sido muy útil debido a la comodidad y facilidad con la que representa los datos en pantalla. En la Figura 6.3 podemos ver un ejemplo con tres nubes de puntos. Fácilmente podemos diferenciar cada nube de puntos otorgándoles un color distinto y solapándolos. Además, MeshLab contiene herramientas interesantes para el análisis de los resultados, pudiendo convertir las nubes de puntos resultantes en mallas 3D

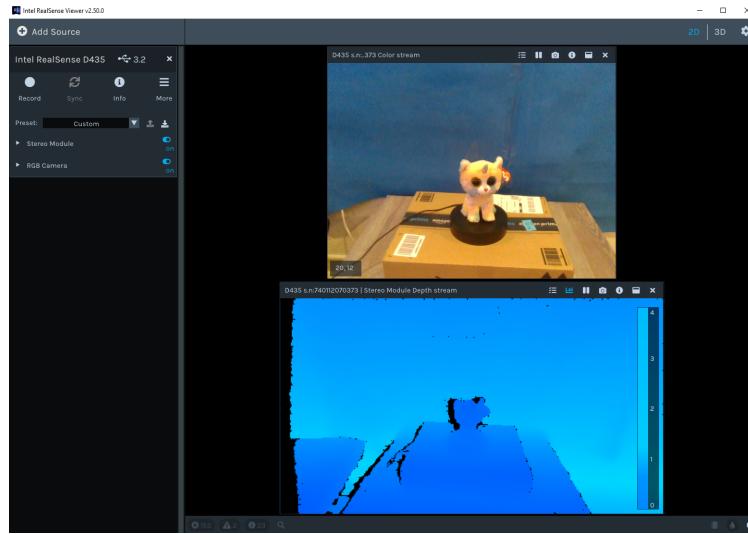


Figura 6.2: Captura de pantalla de Intel RealSense Viewer.

con mejor visualización (Kazhdan y Hoppe, 2013).

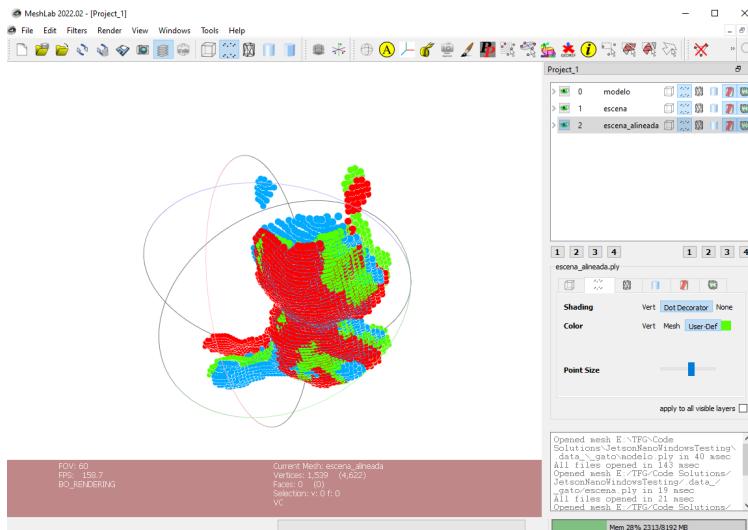


Figura 6.3: Captura de pantalla de MeshLab con 3 nubes de puntos.

6.2. Resultados entre un modelo y una escena

Para probar la solución del algoritmo ICP entre un modelo y una escena, utilizaremos un objeto rígido que rote ligeramente entre una captura y otra. Para ello, utilizaremos las capturas de la Figura 6.4.

En la Figura 6.5 podemos ver los resultado de aplicar los algoritmo ICP de PCL, CUDA-ICP y CPD. Para lograr estos resultados se ha establecido un máximo de 200 iteraciones en



(a) Modelo.

(b) Escena.

Figura 6.4: Ejemplo de alineación de una escena con un modelo.

los algoritmos, y un límite de distancia máxima de correspondencias de 0.01m, ya que las correspondencias deben ser muy cercanas. Como se puede observar en las subfiguras 6.5d y 6.5e la convergencia ha funcionado bastante bien y ambos resultados son muy parecidos. Se ha probado a utilizar el algoritmo CPD con la funcionalidad no rígida, sin embargo los resultados no son tan satisfactorios como se puede ver en la subfigura 6.5f. Además, cabe mencionar que los tiempos para lograr estos resultados han sido de 0.97s para ICP, 0.72s para CUDA-ICP y 87.3s para CPD. Por el coste computacional tan elevado y los resultados obtenidos tras diversas pruebas con CPD se ha decidido abandonar la idea de una transformación no rígida para centrarnos en los buenos resultados que nos pueden llegar a dar ICP y CUDA-ICP.

6.3. Resultados de la reconstrucción completa con un objeto

Para la reconstrucción completa de un modelo 3D comenzaremos experimentando con el objeto de pruebas de la Figura 6.1 y, finalmente, se realizarán pruebas con el cuerpo humano. En estas pruebas se utilizarán los algoritmos ICP y CUDA-ICP para comparar la calidad de la reconstrucción y los tiempos empleados en el dispositivo Jetson Nano. Comenzaremos con el objeto de pruebas que ya hemos utilizado anteriormente.

Tras la fase de adquisición y pre-procesamiento obtenemos las capturas desde distintos ángulos del objeto. El objeto ha estado girando con velocidad constante en un plato giratorio que tarda 21 segundos en realizar una vuelta 360° y para la fase de adquisición se ha establecido que el sensor capture imágenes cada 0.5 segundos. Por tanto, obtenemos un total de 42 capturas del objeto. A continuación, la primera de estas capturas será interpretada como el modelo y el resto irán siendo alineadas consecutivamente, hasta tener la reconstrucción completa.

En las Figuras 6.6 y 6.7 podemos ver los resultados de la reconstrucción con ICP y CUDA-ICP, respectivamente. Las cuatro primeras imágenes muestran el resultado de la nube de puntos reconstruida completamente, las siguientes cuatro imágenes muestran la reconstrucción de una malla generada a partir de la nube de puntos.

En la Tabla 6.1 tenemos datos interesantes respecto a los tiempos de reconstrucción. En la primer columna se indica el número de iteración que está pasando, seguidamente se indica la cantidad de puntos que tenía la escena en esa iteración, a continuación se expresa el tiempo que ha tardado la iteración en alinearse utilizando ICP y el tiempo total acumulado, seguidamente los mismos tiempos para CUDA-ICP, y finalmente los porcentajes de mejora

de velocidad usando CUDA-ICP frente a ICP respecto al tiempo que tarda la iteración en sí y respecto al tiempo total.

Tras analizar los resultados vemos que los tiempos de ejecución de CUDA-ICP son mucho más rápidos que el algoritmo ICP, por lo que hay una buena mejora al estar aprovechando los núcleos de la gráfica, sin embargo el resultado final no ha conseguido converger tan bien como lo ha hecho ICP en PCL.

6.4. Resultados de la reconstrucción completa de un cuerpo humano

Para lograr un buen resultado en la reconstrucción del cuerpo humano se ha realizado la fase de adquisición con el sujeto sentado en una silla de oficina, de esta forma el sujeto puede rotar cómodamente 360° intentando mantener su cuerpo rígido de forma que la reconstrucción sea más limpia. En esta fase de adquisición se han realizado 30 capturas al sujeto, que es lo que ha tardado en rotar 360° con una frecuencia de capturación de 1 imagen por cada 0.5 segundos.

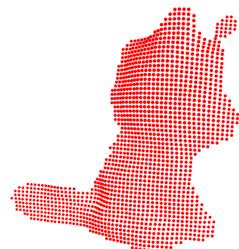
Para el algoritmo ICP se ha establecido un máximo de 200 iteraciones y el límite máximo de distancia entre correspondencias se ha subido a 0.5m debido a que el cuerpo a escanear es mucho más grande y se mueve a una velocidad que no es constante como sí lo hacía el objeto encima del plato giratorio, por lo que es posible que alguna distancia pueda llegar a medir más.

De nuevo, en las figuras 6.8 y 6.9 podemos ver los resultados de la reconstrucción expresados de la misma forma que el objeto, es decir, las primeras cuatro imágenes en nubes de puntos, y las siguientes cuatro con una malla generada a partir de los puntos.

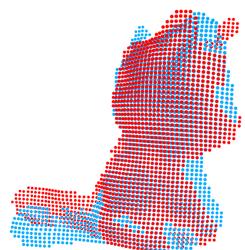
Si analizamos la Tabla 6.2, podemos ver que los resultados son parecido al del objeto, con unos tiempos en general más grandes debido a que la cantidad de puntos de cada escena es mucho mayor comparado con las escenas del objeto. Aunque el algoritmo con CUDA vuelve a ser mucho más rápido, volvemos a tener el problema de que el resultado no es tan correcto como lo es en ICP de PCL. Este es un problema que tenemos con la librería de CUDA ya que no permite mucha configuración a la hora de lanzar el algoritmo, como si lo hace PCL, y no se ha podido corregir el error.



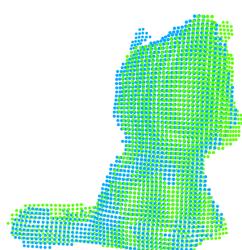
(a) Nube de puntos modelo.



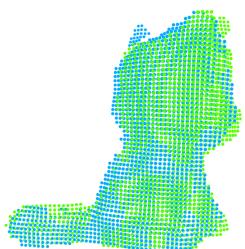
(b) Nube de puntos escena.



(c) Modelo y escena sin alinear.



(d) Alineamiento con ICP.



(e) Alineamiento con CUDA-ICP.



(f) Alineamiento con CPD.

Figura 6.5: Resultados experimentación modelo y escena alineados.



Figura 6.6: Resultados de Reconstrucción 3D con objeto usando ICP.



Figura 6.7: Resultados de Reconstrucción 3D con objeto usando CUDA-ICP.

Iteración	Puntos	Tiempo ICP (s)		Tiempo CUDA-ICP (s)		Mejora de velocidad	
		t/iteracion	Total	t/iteracion	Total	%/iteración	Total
0	3540						
1	3523	1,837	1,837	0,607	0,607	302,37%	302,37%
2	3497	3,056	4,892	0,977	1,585	312,62%	308,69%
3	3557	4,613	9,505	1,121	2,705	411,64%	351,33%
4	3597	3,982	13,487	0,984	3,689	404,79%	365,59%
5	3710	4,133	17,620	1,159	4,849	356,47%	363,41%
6	3702	5,726	23,346	1,125	5,973	509,01%	390,83%
7	3634	3,965	27,311	1,035	7,009	382,93%	389,66%
8	3755	3,877	31,188	1,051	8,060	368,94%	386,96%
9	3686	4,259	35,446	1,305	9,365	326,30%	378,50%
10	3792	3,364	38,811	1,012	10,377	332,35%	374,00%
11	3775	3,536	42,347	1,315	11,693	268,81%	362,17%
12	3883	3,586	45,933	0,971	12,663	369,47%	362,73%
13	3855	3,746	49,678	1,163	13,827	321,95%	359,30%
14	3842	7,131	56,809	1,375	15,201	518,65%	373,71%
15	3994	5,695	62,504	1,217	16,419	467,87%	380,69%
16	4127	6,641	69,145	1,292	17,711	514,04%	390,42%
17	4443	6,107	75,252	1,198	18,909	509,77%	397,98%
18	4268	4,798	80,050	1,621	20,530	295,94%	389,92%
19	4128	5,402	85,451	1,464	21,993	369,04%	388,53%
20	3955	5,822	91,273	1,315	23,309	442,66%	391,59%
21	3931	4,329	95,602	1,262	24,570	343,10%	389,10%
22	4000	5,393	100,995	1,587	26,157	339,94%	386,11%
23	3922	6,951	107,947	1,553	27,710	447,57%	389,56%
24	3828	5,326	113,272	1,621	29,331	328,51%	386,18%
25	3739	6,527	119,799	1,574	30,905	414,58%	387,63%
26	3687	5,665	125,464	1,531	32,436	370,08%	386,80%
27	3665	7,183	132,647	1,810	34,246	396,91%	387,34%
28	3658	8,995	141,642	1,468	35,714	612,74%	396,60%
29	3746	6,553	148,195	1,497	37,211	437,62%	398,25%
30	3747	8,390	156,585	1,774	38,985	473,00%	401,65%
31	3632	9,176	165,761	1,908	40,893	480,98%	405,35%
32	3516	8,287	174,048	1,726	42,619	480,13%	408,38%
33	3497	5,666	179,714	1,971	44,590	287,51%	403,04%
34	3513	5,748	185,462	1,948	46,538	295,03%	398,52%
35	3522	5,483	190,945	1,837	48,375	298,48%	394,72%
36	3558	5,042	195,986	1,798	50,173	280,45%	390,62%
37	3613	5,722	201,708	1,961	52,134	291,73%	386,90%
38	3678	5,355	207,063	1,835	53,969	291,74%	383,67%
39	3693	8,961	216,024	1,999	55,969	448,18%	385,97%
40	3638	6,771	222,795	1,999	57,967	338,78%	384,34%
41	3662	8,201	230,996	1,928	59,896	425,32%	385,66%

Tabla 6.1: Resultados de Reconstrucción 3D con objeto.

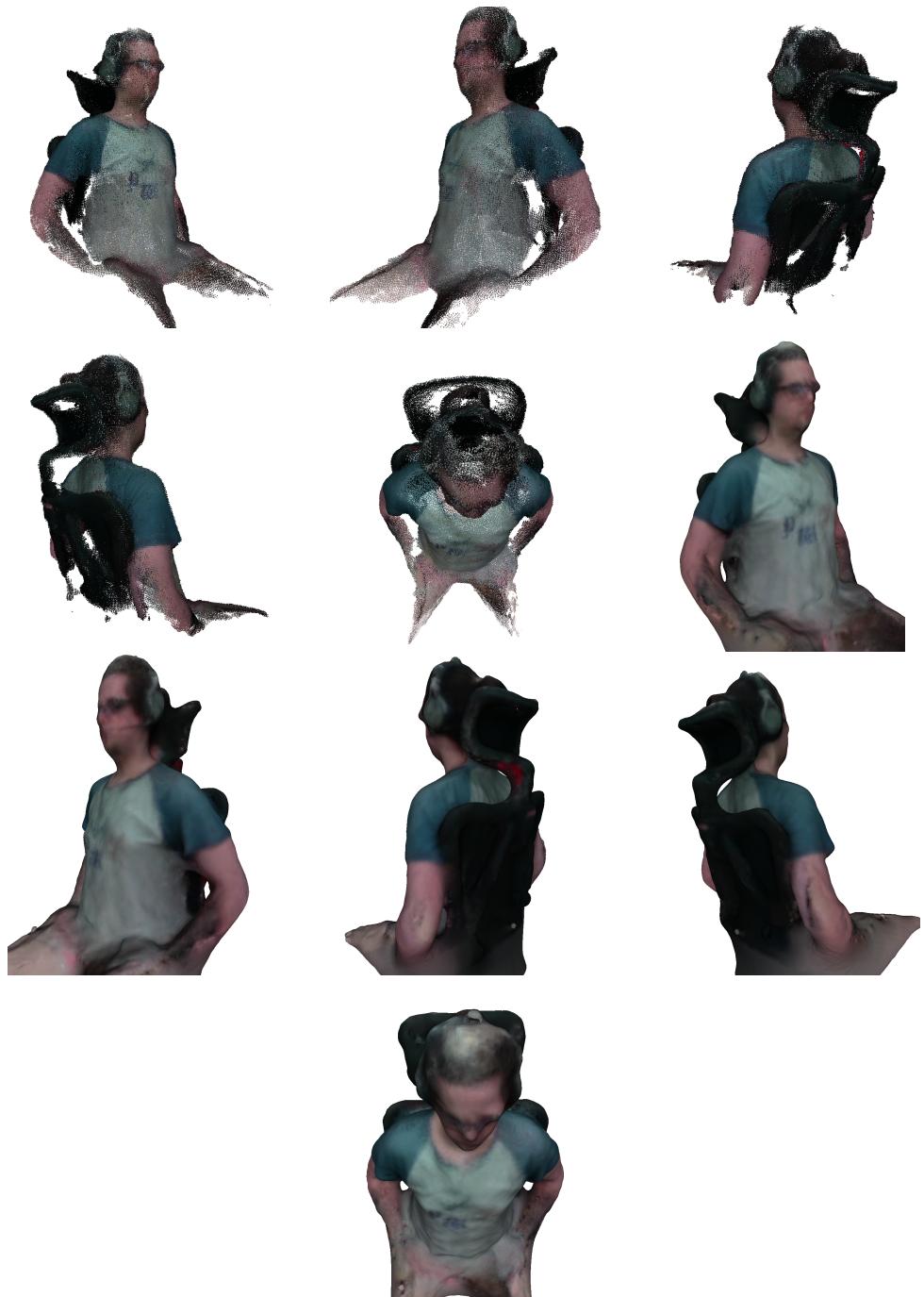


Figura 6.8: Resultados de Reconstrucción 3D de un cuerpo humano usando ICP.



Figura 6.9: Resultados de Reconstrucción 3D de un cuerpo humano usando CUDA-ICP.

Iteración	Puntos	Tiempo ICP (s)		Tiempo CUDA-ICP (s)		Mejora de velocidad	
		t/iteracion	Total	t/iteracion	Total	%/iteración	Total
0	19256						
1	19164	11,020	11,020	3,189	3,189	345,56%	345,56%
2	19021	18,334	29,354	5,132	8,321	357,28%	352,79%
3	19349	27,676	57,030	5,883	14,204	470,44%	401,52%
4	19569	23,892	80,922	5,165	19,368	462,62%	417,81%
5	20181	24,798	105,720	6,087	25,455	407,39%	415,32%
6	20137	34,354	140,074	5,906	31,361	581,73%	446,66%
7	19768	23,790	163,864	5,436	36,797	437,64%	445,32%
8	20426	23,262	187,126	5,517	42,314	421,64%	442,24%
9	20052	25,552	212,678	6,852	49,166	372,91%	432,58%
10	20626	20,186	232,864	5,315	54,480	379,83%	427,43%
11	20536	21,216	254,080	6,906	61,386	307,21%	413,91%
12	21124	21,516	275,596	5,096	66,482	422,25%	414,55%
13	20973	22,474	298,070	6,108	72,590	367,94%	410,62%
14	20902	42,784	340,854	7,218	79,808	592,74%	427,10%
15	21729	34,168	375,022	6,390	86,198	534,71%	435,07%
16	22452	39,848	414,870	6,783	92,981	587,47%	446,19%
17	24171	36,642	451,512	6,290	99,270	582,59%	454,83%
18	23220	28,786	480,298	8,511	107,781	338,22%	445,62%
19	22454	32,410	512,708	7,685	115,466	421,76%	444,04%
20	21515	34,930	547,638	6,904	122,370	505,90%	447,53%
21	21384	25,974	573,612	6,624	128,994	392,12%	444,68%
22	21760	32,360	605,972	8,330	137,324	388,50%	441,27%
23	21333	41,708	647,680	8,154	145,478	511,50%	445,21%
24	20827	31,954	679,634	8,511	153,989	375,44%	441,35%
25	20341	39,160	718,794	8,265	162,254	473,81%	443,01%
26	20055	33,992	752,786	8,037	170,291	422,94%	442,06%
27	19938	43,098	795,884	9,501	179,792	453,62%	442,67%
28	19902	53,970	849,854	7,707	187,499	700,27%	453,26%
29	20376	39,318	889,172	7,862	195,360	500,13%	455,15%
30	20385	50,338	939,510	9,312	204,672	540,57%	459,03%
31	19757	55,054	994,564	10,016	214,688	549,69%	463,26%

Tabla 6.2: Resultados de Reconstrucción 3D de un cuerpo humano.

7. Conclusión

En este trabajo se presenta un sistema económico y portable que es capaz de reconstruir objetos y cuerpos en 3D de forma eficiente utilizando un sensor RGB-D y un pequeño computador embebido que es capaz de aprovechar la GPU para acelerar el funcionamiento. Se ha implementado uno de los métodos de registro de nubes de puntos más utilizado haciendo uso de la librería PCL y se ha conseguido usar aprovechando la concurrencia en el algoritmo con los núcleos de la GPU. Sin embargo, se ha hecho uso de una librería oficial implementada por Nvidia que no proporciona mucho margen para personalizarla, lo que ha sido un problema para obtener resultados de buena calidad. Se plantea como mejora a corto plazo para este trabajo la posibilidad de desarrollar una implementación del algoritmo ICP personalizada, que de juego a modificar parámetros para conseguir mejores resultados.

Por otro lado, también es interesante como mejora a corto plazo seguir investigando en la línea de los métodos de registro no rígidos, desarrollando también una versión personalizada del algoritmo PCL que utilice CUDA y pueda aprovechar los núcleos de la GPU. De esta forma podría llegar a ser viable la ejecución de estos algoritmos en este sistema, lo cual implicaría una mejora de calidad en las reconstrucciones sobre cuerpos humanos debido a que permitiría realizar una reconstrucción 3D con modelos no rígidos.

Anexos

A. Entorno experimental

Con el objetivo de realizar la reconstrucción de cuerpos u objetos en modelos 3D se ha creado un pequeño entorno experimental donde realizar las pruebas. El entorno está compuesto por el sensor Intel RealSense D435, montado en un trípode y con posición fija y un plato giratorio para las pruebas con objetos pequeños y rígidos.

En la Figura A.1 podemos ver un objeto encima del plato giratorio y el sensor Intel RealSense D435 montado en el trípode.



Figura A.1: Entorno experimental.

El plato sobre el que está subido el objeto es un plato con velocidad fija que rota indefinidamente. Utilizarlo facilita la tarea de reconstrucción 3D sobre los objetos ya que con el plato no necesitan ninguna interacción extra para rotar y ser capturados por todos los ángulos. Esto ha facilitado la tarea de experimentación en objetos rígidos para su posterior prueba en

cuerpos humanos.

La pared está cubierta con un panel de poliestireno azul con la finalidad de reducir ruido en la adquisición de datos, debido a que el sensor tiene pequeños fallos a la hora de identificar zonas concretas y muy detalladas y se llega a equivocar en el cálculo de profundidad de algunas zonas. No obstante, aunque el poliestireno ha ayudado un poco, tampoco ha marcado gran diferencia, ya que esto ha seguido ocurriendo.

B. Estructura del proyecto subido a GitHub

El código fuente de todo el proyecto está subido en un repositorio público en GitHub y es accesible para poder realizar las mismas pruebas desarrolladas en este TFG.

El proyecto entero se puede encontrar en GitHub desde el siguiente enlace:

<https://github.com/ems107/EdgarMartinezSerranoTFG>.

Dentro de la estructura del proyecto, nos encontraremos con un README que nos explicará cómo compilar y ejecutar del proyecto.

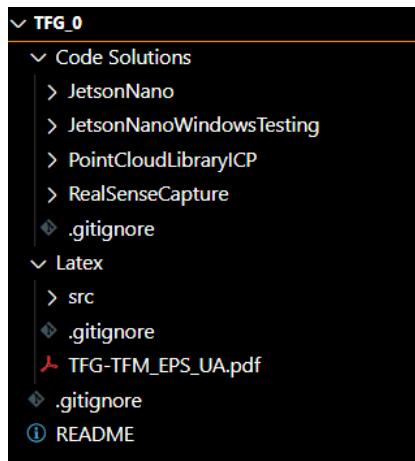


Figura B.1: Estructura del proyecto en GitHub.

En la Figura B.1 podemos ver cómo se estructura el repositorio. Por un lado tenemos el directorio “Latex” donde podemos encontrar todo el código fuente utilizado para construir esta memoria, y la memoria en sí.

Por otro lado, dentro del directorio “Code Solutions” encontraremos todo el código utilizado en el TFG. Dentro del subdirectorío “JetsonNano” encontramos el proyecto CMake que se ha estado explicando detalladamente en la memoria. En el subdirectorío “JetsonNanoWindowsTesting” encontramos una solución en Visual Studio capaz de ejecutar el mismo código que ha sido desarrollado en la Jetson para testear en Windows, excluyendo la librería de CUDA-ICP que no funciona en Windows. Por último, los subdirectorios “PointCloudLibraryICP” y “RealSenseCapture” son pequeñas soluciones en Visual Studio con diversas pruebas sobre estas librerías.

Bibliografía

- Ambitiously, E. (2022). *Decimation (digital filter design toolkit)*. Descargado de https://www.ni.com/docs/en-US/bundle/labview-digital-filter-design-toolkit-api-ref/page/lvdfdtconcepts/dfd_decimation.html#:~:text=Decimation%20filters%20help%20you%20remove,processing%20and%20storing%20the%20signal. ([Online; accedido en 12-Mayo-2022])
- Aoki, Y., Ujihara, S., Saito, T., y Yuminaka, Y. (2020, jul). Development of augmented reality systems displaying three-dimensional dynamic motion in real time. *Physics Education*, 55(4), 045020. Descargado de <https://iopscience.iop.org/article/10.1088/1361-6552/ab9213> doi: 10.1088/1361-6552/ab9213
- Barsukov, Y. (2013). Development of gpicp, a gpu-only registration algorithm based on iterative closest point..
- Besl, P., y McKay, N. D. (1992, feb). A method for registration of 3-D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2), 239–256. Descargado de <http://ieeexplore.ieee.org/document/121791/> doi: 10.1109/34.121791
- Blazquez, S. Z. (2015). *Geotecnologías láser y fotogramétricas aplicadas a la modelización 3d de escenarios complejos en infografía forense* (Tesis Doctoral, Universidad de Salamanca). Descargado de <https://dialnet.unirioja.es/servlet/dctes?codigo=101680>
- Cignoni, P., Callieri, M., Corsini, M., Dellepiane, M., Ganovelli, F., y Ranzuglia, G. (2008). MeshLab: an Open-Source Mesh Processing Tool. En V. Scarano, R. D. Chiara, y U. Erra (Eds.), *Eurographics italian chapter conference*. The Eurographics Association. doi: 10.2312/LocalChapterEvents/ItalChap/ItalianChapConf2008/129-136
- da Silva Neto, J. G., da Lima Silva, P. J., Figueiredo, F., Teixeira, J. M. X. N., y Teichrieb, V. (2020, nov). Comparison of RGB-D sensors for 3D reconstruction. En *2020 22nd symposium on virtual and augmented reality (svr)* (pp. 252–261). IEEE. Descargado de <https://ieeexplore.ieee.org/document/9262651/> doi: 10.1109/SVR51698.2020.00046
- Fioriti, E. (2021). *Analysis of 3d perception based on depth sensors in order to perform 3d scene understanding* (Tesis Doctoral, Politecnico di Torino). Descargado de <https://webthesis.biblio.polito.it/21023/>
- Fischler, M. A., y Bolles, R. C. (1981, jun). Random sample consensus. *Communications of the ACM*, 24(6), 381–395. Descargado de <https://dl.acm.org/doi/10.1145/358669.358692> doi: 10.1145/358669.358692
- Fuster-Guilló, A., Azorín-López, J., Saval-Calvo, M., Castillo-Zaragoza, J. M., Garcia-D'Urso, N., y Fisher, R. B. (2020, jul). RGB-D-Based Framework to Acquire, Visualize and

- Measure the Human Body for Dietetic Treatments. *Sensors*, 20(13), 3690. Descargado de <https://www.mdpi.com/1424-8220/20/13/3690> doi: 10.3390/s20133690
- Glaskowsky, P. N. (2009). Nvidia's fermi: The first complete gpu computing architecture. *A White Paper of NVIDIA*.
- Haleem, A., y Javaid, M. (2019, jun). 3D scanning applications in medical field: A literature-based review. *Clinical Epidemiology and Global Health*, 7(2), 199–210. Descargado de <https://linkinghub.elsevier.com/retrieve/pii/S2213398418300952> doi: 10.1016/j.cegh.2018.05.006
- Hartley, R., y Zisserman, A. (2003). *Multiple view geometry in computer vision*. Cambridge university press.
- Isaza, J., Serna, A., Restrepo, D., Gutiérrez, F., Ramírez, J., y Correza, A. (2011). Reconstrucción digital del muñón de un amputado transfemoral a partir de datos obtenidos de escáner 3d. Universidad Nacional de Educación a Distancia (España). Descargado de <http://e-spacio.uned.es/fez/view/bibliuned:iberoingmecanica-2011-vol15-n1-05>
- Kazhdan, M., y Hoppe, H. (2013). Screened poisson surface reconstruction. *ACM Transactions on Graphics (TOG)*, 32(3), 29.
- Langis, C., Greenspan, M., y Godin, G. (2001). The parallel iterative closest point algorithm. En *Proceedings third international conference on 3-d digital imaging and modeling* (pp. 195–202). IEEE Comput. Soc. Descargado de <http://ieeexplore.ieee.org/document/924434/> doi: 10.1109/IM.2001.924434
- Lei Fan, L. L. (2021, 31 de 01). Accelerating lidar for robotics with nvidia cuda-based pcl.. Descargado de <https://developer.nvidia.com/blog/accelerating-lidar-for-robotics-with-cuda-based-pcl/>
- Luebke, D. (2008, may). CUDA: Scalable parallel programming for high-performance scientific computing. En *2008 5th ieee international symposium on biomedical imaging: From nano to macro* (pp. 836–838). IEEE. Descargado de <http://ieeexplore.ieee.org/document/4541126/> doi: 10.1109/ISBI.2008.4541126
- Luebke, D., Harris, M., Govindaraju, N., Lefohn, A., Houston, M., Owens, J., ... Buck, I. (2006). S07—GPGPU. En *Proceedings of the 2006 acm/ieee conference on supercomputing - sc '06* (p. 208). New York, New York, USA: ACM Press. Descargado de <http://portal.acm.org/citation.cfm?doid=1188455.1188672> doi: 10.1145/1188455.1188672
- Martínez Belda, M. d. M. (2020). *Diseño de prótesis inferiores impresas en 3d para personas amputadas en camerún* (Tesis Doctoral, Universitat Politècnica de València). Descargado de <https://riunet.upv.es/handle/10251/153323>
- Myronenko, A., y Xubo Song. (2010, dec). Point Set Registration: Coherent Point Drift. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(12), 2262–2275. Descargado de <http://ieeexplore.ieee.org/document/5432191/> doi: 10.1109/TPAMI.2010.46

- Ranzuglia, G., Callieri, M., Dellepiane, M., Cignoni, P., y Scopigno, R. (2013). Meshlab as a complete tool for the integration of photos and color with high resolution 3d geometry data. En *Caa 2012 conference proceedings* (p. 406-416). Pallas Publications - Amsterdam University Press (AUP). Descargado de <http://vcg.isti.cnr.it/Publications/2013/RCDCS13>
- Rayón Encinas, E., Patricia Arrieta, M., Ferrández Bou, S., y López Martínez, J. (2015, jun). Desarrollo de metodología docente enfocada a alumnos de grado en Ingeniería de Diseño Industrial y del Producto. Generación de prototipos por modelado, escaneado e impresión 3D. En *Libro de actas in-red 2015 - congreso nacional de innovación educativa y de docencia en red* (pp. 237–246). Editorial Universitat Politècnica de València. Descargado de <http://ocs.editorial.upv.es/index.php/INRED/INRED2015/paper/view/1590> doi: 10.4995/INRED2015.2015.1590
- Rotation matrix. (22/04/2022). *Rotation matrix — Wikipedia, the free encyclopedia*. Descargado de https://en.wikipedia.org/w/index.php?title=Rotation_matrix&oldid=1084060907 ([Online; accedido en 22-Abril-2022])
- Stone, J. E., Gohara, D., y Shi, G. (2010, may). OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems. *Computing in Science & Engineering*, 12(3), 66–73. Descargado de <http://ieeexplore.ieee.org/document/5457293/> doi: 10.1109/MCSE.2010.69
- Stopper, D., y Roth, R. (2017, aug). Massively parallel GPU-accelerated minimization of classical density functional theory. *The Journal of Chemical Physics*, 147(6), 064508. Descargado de <http://aip.scitation.org/doi/10.1063/1.4997636> doi: 10.1063/1.4997636
- Suzen, A. A., Duman, B., y Sen, B. (2020, jun). Benchmark Analysis of Jetson TX2, Jetson Nano and Raspberry PI using Deep-CNN. En *2020 international congress on human-computer interaction, optimization and robotic applications (hora)* (pp. 1–5). IEEE. Descargado de <https://ieeexplore.ieee.org/document/9152915/> doi: 10.1109/HORA49412.2020.9152915
- Tech4Diet. (2020). *Tech4Diet - Principales resultados del proyecto*. Descargado de <http://tech4d.dtic.ua.es/publicaciones/resultados-priv/>
- Transformation matrix. (25/05/2022). *Transformation matrix — Wikipedia, the free encyclopedia*. Descargado de https://en.wikipedia.org/w/index.php?title=Transformation_matrix&oldid=1089725131 ([Online; accedido en 25-Mayo-2022])
- Translation matrix. (14/01/2022). *Translation matrix — Wikipedia, the free encyclopedia*. Descargado de [https://en.wikipedia.org/w/index.php?title=Translation_\(geometry\)&oldid=1065586209](https://en.wikipedia.org/w/index.php?title=Translation_(geometry)&oldid=1065586209) ([Online; accedido en 14-Enero-2022])
- Valverde-Bastidas, J., Cesén-Arteaga, M., y Sarmiento-Borja, E. (2020, jan). Restauración y conservación digital de fósiles mediante escaneado 3D y la reproducción con prototipado rápido. *Revista Arbitrada Interdisciplinaria Koinonía*, 5(9), 392. Descargado de <https://fundacionkoinonia.com.ve/ojs/index.php/revistakoinonia/article/view/657> doi: 10.35381/r.k.v5i9.657

Whyte, R. (2022). *Comparing depth cameras: itof versus active stereo.* Descargado de <https://medium.com/chronoptics-time-of-flight/comparing-depth-cameras-itof-versus-active-stereo-e163811f3ac8> ([Online; accedido en 14-Junio-2022])

Lista de Acrónimos y Abreviaturas

2D	Dos Dimensiones.
3D	Tres Dimensiones.
ATC	Arquitectura y Tecnología de los Computadores.
CPD	Coherent Point Drift.
CPU	Central Processing Unit.
CUDA	Compute Unified Device Architecture.
DTIC	Departamento de Tecnología Informática y Computación.
FGT	Fast Gauss Transforms.
GPGPU	General-Purpose Computing on Graphics Processing Units.
GPU	Graphics Processing Unit.
ICP	Iterative Closest Point.
IDIDP	Ingeniería en Diseño Industrial y Desarrollo de Producto.
IR	Radiación Infrarroja.
IS	Ingeniería del Software.
LIDAR	Light Detection And Ranging.
MMG	Modelo Mixto Gaussiana.
OpenCL	Open Computing Language.
PCD	Point Cloud Data.
PCL	Point Cloud Library.
PLY	Polygon File Format.
PVP	Precio de Venta al Públco.
RAM	Random Access Memory.
RANSAC	Random Sample Consensus.
RGB	Red Green Blue.
RGB-D	Red Green Blue – Depth.
SIMT	Single Instruction Multiple Threads.
SM	Streaming Multiprocessors.
SP	Streaming Processors.
TFG	Trabajo Final de Grado.
ToF	Time of Flight.
UPV	Universidad Politécnica de Valencia.
VCSEL	Vertical Cavity Surface Emitting Laser.