Universidad del País Vasco
Euskal Herriko Unibertsitatea
VITORIA-GASTEIZKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE VITORIA-GASTEIZ

GRADO EN INGENIERÍA INDUSTRIAL ELECTRÓNICA Y AUTOMATIZACIÓN

TRABAJO DE FIN DE GRADO

ALGORITMO DE LOCALIZACIÓN Y MAPEADO SIMULTANEO BASADO EN CÁMARA CON SENSOR DE PROFUNDIDAD INFRARROJO

Alumno: Jon Alexander Pradini Santos

Tutor: Ekaitz Zulueta Guerrero

Curso: 2022-2023

Fecha: Vitoria-Gasteiz, 28 de marzo de 2023

RESUMEN

En este proyecto se va a tratar de implementar un algoritmo de SLAM basado en cámara de profundidad. Para ello se van a comparar diferentes estrategias: correlación de fase, NDT, CPD e ICP. De dicha comparación se selecciona la estrategia de ICP, más concretamente la versión propuesta por P. Henry (2014) [8].

Esta estrategia combina las características visuales (SIFT) de fotogramas RGB y las nubes de puntos halladas mediante imágenes de profundidad para dar como resultado una nueva estrategia RGB-ICP que aúna lo mejor de las dos.

El reto de afrontar una estrategia mixta hará necesario entrar en profundidad en el aspecto teórico de la misma.

LABURPENA

Proiektu honen helburua sakontasun-kameran oinarritutako SLAM algoritmo bat inplementatzea da. Horretarako, hainbat estrategia alderatuko dira: fase-korrelazioa, NDT, CPD eta ICP. Konparazio horretatik ICP estrategia hautatzen da, zehazki P. Henryk (2014) proposatutako bertsioa [8].

Estrategia honek RGB fotogramen ezaugarri bisualak (SIFT) eta sakontasun-irudien bidez aurkitutako puntu-hodeiak konbinatzen ditu, bien onena bateratzen duen RGB-ICP estrategia berri bat lortzeko.

Estrategia misto bati aurre egiteko erronkak beharrezkoa egingo du estrategia horren alderdi teorikoan sakontzea.

ABSTRACT

In this project we are going to try to implement a SLAM algorithm based a on depth camera. For this, different strategies will be compared: phase correlation, NDT, CPD and ICP. From this comparison, the ICP strategy is selected, more specifically the version proposed by P. Henry (2014) [8].

This strategy combines the visual characteristics (SIFT) of RGB frames and point clouds found by depth imaging to result in a new RGB-ICP strategy that combines the best of both.

The challenge of facing a mixed strategy will make it necessary to delve into its theoretical aspect in depth.

ÍNDICE

ÍNDICE DE FIGURAS

GLOSARIO

- AGV: Autonomus Ground Vehicle (Vehículo Guiado Automático)
- SLAM: Simultaneus Location and Mapping
- RGB: Red Green Blue
- ICP: Iterative Clasest Point
- SIFT: Scale-Invariant Feature Transform
- RANSAC: Random Sample Consensus
- LIDAR: Laser Imaging Detection And Ranging
- NDT: Normal Distribution Transform
- CPD: Coherent Point Drift
- GMM: Gaussian Mixture Model
- SDK: System Development Kit
- TORO: Tree-based Network Optimizer

1. INTRODUCCIÓN

La conducción autónoma es un ámbito en alza en varios campos de aplicación, desde la asistencia a la conducción en vehículos convencionales hasta el uso de carros de piezas autónomos en la industria. También es un ámbito con proyección a futuro, como en el caso de la conducción completamente automatizada en turismos.

A todos estos vehículos se los conoce, en su conjunto, como AGVs (por sus siglas en inglés, **A**utonomous **G**round **V**ehicle) y presentan algunos retos específicos derivados del entorno en el que se desenvuelven.

Es esencial en todos los casos la adquisición y procesamiento de información sobre el entorno para un correcto funcionamiento. A este conocimiento del entorno hay que sumarle el conocimiento de la posición y orientación del propio vehículo en dicho entorno. Con frecuencia, hay que resolver estas dos tareas (mapeado y localización) de forma simultánea debido al desconocimiento o mutabilidad del entorno o a la ausencia o imprecisión de herramientas de posicionamiento absoluto.

Para la adquisición de información del entorno, existen multitud de tipos de sensores, desde las cámaras convencionales (RGB) hasta los LiDARes, incluyendo transductores ultrasónicos o, en el caso de este proyecto, las cámaras de profundidad.

Cada sensor ofrece sus propias ventajas e inconvenientes en aspectos como precisión, alcance, bidimensionalidad o tridimensionalidad de las medidas, y sin olvidar el coste.

En la actualidad, son muchas las estrategias utilizadas para el procesamiento de los datos y la construcción de ese conocimiento del entorno.

En primer, lugar hay que tener en cuenta el tipo de información que nos provee el sensor utilizado, o la combinación de varios sensores.

Una opción es partir de dos imágenes obtenidas con uso de dos cámaras convencionales, separadas entre sí en una distancia conocida, para hallar puntos de interés en las imágenes obtenidas, puntos con características que sirvan para identificarlos como únicos y relacionarlos con sus equivalentes en las dos vistas obtenidas, creando una visión en estéreo y calculando la posición de los puntos de interés con respecto a un punto de referencia en relación a las cámaras. Para esto existen varios algoritmos de extracción de características, y a partir de ahí se aplica un

cálculo trigonométrico de la posición, teniendo en cuenta las propiedades ópticas de las cámaras.

Una variante de esta opción, es utilizar un vehículo que cuente con una sola cámara y un medio para estimar su propio movimiento a través del seguimiento del movimiento de sus ruedas, siempre que la tracción de las ruedas y la velocidad del movimiento permitan que la odometría sea fiable. En este caso, es posible tomar dos capturas desde posiciones diferentes, aplicando después los mismos pasos que en el apartado anterior para reconstruir el entorno en 3D.

Otra opción es utilizar sensores de la familia de medidores de tiempo de vuelo, aunque la naturaleza de los datos obtenidos es muy diferente. Estos sensores se basan en la velocidad de transmisión de una onda por su medio (sonido en el caso de los transductores ultrasónicos, y luz láser infrarroja en el caso del LiDAR). Cronometrando el tiempo transcurrido entre la emisión de un pulso de dicha onda (o un patrón reconocible de pulsos) y la recepción del mismo en un detector colocado junto al emisor, se puede calcular la distancia a la que se encuentra el objeto que haya reflejado la señal, puesto que la velocidad de transmisión es una constante conocida. Conociendo la orientación del sensor y habiendo medido la distancia hasta el objeto, es posible calcular su posición relativa.

Aunque esta tecnología solo ofrece la lectura de un punto, con una disposición de abanico (común con los transductores por su bajo coste) o con un sistema de barrido, es posible reconstruir un modelo 2D del entorno.

Una tercera opción, y la elegida para este proyecto, son las cámaras de profundidad. Estas aúnan los mayores puntos fuertes de las dos opciones anteriores, sirven para la reconstrucción 3D del entorno, como las cámaras convencionales, y tienen la facilidad aritmética de los sensores de tiempo de vuelo.

Se trata de cámaras similares a las convencionales con la diferencia de que son sensibles a la luz infrarroja, gracias a ello y acompañándolas de un emisor intenso de infrarrojos podemos calcular la distancia de los objetos al ser esta proporcional a su grado de reflexión, se puede ver como representan la distancia en la ilustración 1.

Estas cámaras suelen ir acompañadas de cámaras convencionales, lo que permite la utilización de métodos mixtos, como desarrollaremos más adelante.

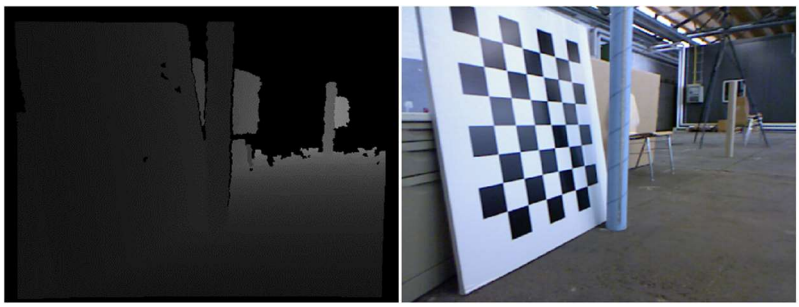

Ilustración 1. a) Captura de profundidad por infrarrojos. b) Captura del espectro visible

2. OBJETIVOS

Como primer eslabón en la cadena de la conducción autónoma, el objetivo de este proyecto es el desarrollo de una solución al problema del SLAM (de sus siglas en inglés, Simultaneous Location And Mapping).

Para ello, nos basaremos en una cámara de profundidad de grado comercial (Intel® RealSense™ D435f) y desarrollaremos el programa en Matlab.

Se utilizará para el desarrollo del programa un "dataset" ofrecido gratuitamente por la Technological University of Munich [1], el set "freiburg2_pioneer_slam", que contiene las imágenes capturadas por un sensor de tipo Kinect montado sobre un robot móvil y conducido manualmente.

Este proyecto aportará una alternativa a la tendencia mayoritaria del desarrollo en base a LiDAR, ofreciendo en comparación a este una sensorización más económica y una reconstrucción 3D del entorno.

3. FASES DEL TRABAJO

Para la ejecución y desarrollo del proyecto se han seguido las siguientes etapas:

- Estudio del estado del arte y análisis de estrategias.
- Adquisición y pre-procesamiento de imágenes.
- Registro de imágenes secuenciales.
- Detección de cierre de ciclos, corrección y registro de localizaciones.
- Almacenamiento eficiente del mapa.

4. RECURSOS UTILIZADOS

Para la realización de este trabajo se han utilizado una serie de recursos materiales e informáticos, sin los cuales no hubiese sido posible alcanzar los objetivos anteriormente escritos. Dichos recursos son:

- **Lenovo ideapad 330S:** Ordenador portátil en el que se ha desarrollado el trabajo. Con sistema operativo Windows 11 Home 22H2 (x64), procesador Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz 2.00 GHz, y 8 GB de memoria RAM.

- **Intel® RealSense™ D455:** Conjunto de cámaras convencionales y de espectro infrarrojo, con foco infrarrojo integrado y la electrónica necesaria para su control. Tiene un rango útil de ente 60 cm y 6 m con un error de menos de 2%. Viene acompañado de un SDK gratuito.

- **Matlab R2022b:** Entorno de desarrollo de extensa implantación en la industria con mucha variedad de librerías de funciones organizadas por campo de aplicación y una documentación oficial muy detallada. Se utilizan las librerías “Image Adquisition Toolbox” y “Computer Vision Toolbox”.

- **Dataset:** El Grupo de Visión por Computadora de la Escuela de Computación, Información y Tecnología de la Universidad Técnica de Munich ofrece de forma abierta y gratuita multitud de paquetes de datos tomados de un sensor Microsoft Kinect, funcionalmente equivalente al Intel RealSense, aunque con diferente resolución y campo visual.

  Se trata de conjuntos de carpetas que contienen las imágenes tomadas por las cámaras de este sensor, además de archivos de texto sin formato que contienen de forma ordenada las rutas hasta esas imágenes para facilitar su importación por software. Estos datos se utilizan para poder desarrollar el programa en cualquier entorno, independientemente de la disponibilidad del sensor y para debuggear y comparar resultados de una forma internamente comparable.

5. ESTADO DEL ARTE Y ANÁLISIS DE ESTRATEGIAS

5.1. DEFINICIÓN DE SLAM

SLAM (localización y mapeo simultáneos) es un método utilizado en vehículos autónomos que permite crear un mapa y localizar el vehículo en ese mapa al mismo tiempo. Los algoritmos de SLAM permiten que el vehículo cree mapas de entornos desconocidos. [2]

5.2. PERSPECTIVA HISTÓRICA

La resolución de este problema forma parte fundamental de la capacidad de autonomía que se persigue con estos vehículos y ha sido objeto de investigación y desarrollo durante los últimos treinta años.

Desde el advenimiento de los robots móviles en los años 40, la ambición de que estos pudieran ejecutar tareas autónomas ha sido uno de los mayores focos de la investigación en robótica. Inicialmente, los problemas de localización y mapeado se trataban de forma determinista y se resolvían de forma independiente hasta la popularización de aproximaciones probabilísticas en los 90, cuando los investigadores se dieron cuenta de que ambas tareas están correlacionadas y dependen la una de la otra [2]

En un principio, se desarrollaron múltiples y fiables soluciones para entornos estáticos, estructurados y de pequeñas dimensiones, pero quedó abierto el problema de desarrollo de soluciones fiables para entornos dinámicos o de gran tamaño.

5.3. EN LA ACTUALIDAD

Actualmente, se observa que el SLAM está recibiendo atención renovada gracias a las oportunidades ofrecidas por las redes neuronales y otros métodos basados en aprendizaje.

Sin embargo, a pesar del rol del SLAM en muchas aplicaciones, la disparidad y la ausencia de unificación en la literatura ha evitado que la comunidad de investigadores ofrezca una marco coherente y cohesionado, que desarrolle algoritmos hasta la madurez y transicionen éstos a aplicaciones reales.

Lamentablemente, la falta de instrumentos cohesivos de las distintas investigaciones impide o dificulta la comparación y la reproducibilidad de los trabajos,

así como la métrica de sus resultados, entorpeciendo la maduración del campo y su impacto en el mundo real.

5.4. ESTRATEGIAS

La tarea principal de cualquier solución de SLAM es la de alinear las capturas tomadas por los sensores en poses consecutivas. Para esto es necesario estimar, de la diferencia entre estas mismas capturas, el movimiento sucedido entre ellas. De esta manera se localiza la pose del sensor en la segunda captura en coordenadas relativas a la pose anterior, y al ser conocida ésta en coordenadas absolutas, se pueden calcular las coordenadas absolutas de la segunda pose. Y con esos datos puede añadirse la información nueva de la segunda captura al mapa en una posición correcta.

A continuación, se exponen diferentes técnicas de alineamiento, y se justifica la elección de la que se utilizará en el desarrollo.

Correlación de fase: Esta aproximación utiliza representaciones en el dominio de frecuencia de las imágenes para estimar el desplazamiento traslacional entre ellas. Utiliza la transformada de Fourier y la correlación cruzada para calcular la similitud entre ambas imágenes en función de su posición relativa, hallando un pico máximo reconocible en torno a la posición del alineamiento. Esta estrategia escala exponencialmente en costo computacional con cada dimensión adicional a evaluar, y aunque eficaz en imágenes bidimensionales sin rotación, queda fuera de consideración para un problema de 6 grados de libertad. [3]

Transformación por distribuciones normales (NDT): Este es un algoritmo creado específicamente para el registro (alineamiento) de nubes de puntos en 2003. En una primera fase, asocia una distribución normal definida por partes a la nube de puntos fija (aquella cuyas coordenadas absolutas ya son conocidas). Esa distribución cuantifica la probabilidad de medir un punto al muestrear unas coordenadas específicas del espacio. En una segunda fase, el algoritmo busca maximizar la probabilidad de muestreado de todos los puntos de la segunda nube, hallando para ello la transformación necesaria. Es un algoritmo rápido y preciso, y escalable a cantidades elevadas de datos, es vulnerable a hallar máximos locales que no se correspondan con el alineamiento real, por lo tanto, requiere de inicialización, haciéndolo útil en estrategias de alineamiento basto a fino. [4]

Deriva de puntos coherente (CPD): Este también es un método probabilístico. Utiliza un modelo de mezcla de Gaussianas (GMM) para representar la nube de puntos fija. Despues se calcula la probabilidad posterior de cada punto de la nube móvil, y se maximiza. Para preservar la estructura topológica, los centroides de estos se desplazan de forma coherente. Se utiliza el algoritmo de Esperanza-Maximización para optimizar la función de coste. Es un método robusto y que posibilita el registro no rígido, pero a expensas de una complejidad conceptual muy elevada. [5]

Punto más cercano iterativo (ICP): Esta estrategia no solo es de las más veteranas, propuesta inicialmente en 1987 [6], sino que sigue siendo relevante y de hecho el estándar contra el que se miden todas las demás estrategias [7]. Se fundamenta en la asignación de correspondencias entre cada punto de la nube móvil y el punto más cercano en la nube fija, y en la minimización de la suma de las distancias entre pares correspondientes. Aunque en su forma más sencilla es vulnerable a outliers, ruido y solapamientos parciales, existe un amplio abanico de variantes que, con sutiles diferencias en los métodos de asignación de correspondencias, inclusión de diferentes consideraciones en el termino de error a minimizar, y diferentes métodos de resolución de dicha minimización, mantienen a este algoritmo al frente de las soluciones para el registro de nubes de puntos.

El método elegido para el desarrollo de este proyecto es una de estas variantes, el RGB-ICP, propuesto por P.Henry et al. [8], que no solo implementa un pre-alineamiento basado en características SIFT, sino que además las introduce en el término de error del propio ICP, consiguiendo así solventar la deriva que tiende a ocurrir cuando los espacios contienen grandes superficies planas como paredes.

6. ADQUISICIÓN Y PREPROCESAMIENTO DE IMAGENES

6.1. ADQUISICIÓN DE IMÁGENES

En cualquier caso, el primer paso a ejecutar es el de la adquisición de las imágenes de las cámaras. Para eso es necesario instalar el SDK (System Development Kit) del GitHub de Intel [9]. Este pack incluye las librerías de funciones para acceder al control del dispositivo desde Matlab. Habrá de hacerse primero creando un objeto para la cámara mediante la función “objCam = realsense.pipeline();” iniciando el objeto “perfil = objCam.start();” y finalmente adquiriendo las imágenes “fs = objCam.wait_for_frames();”. Despues podemos elegir guardar la imagen de

profundidad “depth = fs.get_depth_frame();” o la de color “color = fs.get_color_frame();”.

Puesto que se trabaja con un dataset que contiene carpetas con las imágenes, la forma de cargar éstas cambia. Primero deben cargarse los .txt que contienen los nombres de todos los archivos ordenados de la siguiente forma:

```
Dchart = readcell('STORAGE2/depth.txt','NumHeaderLines',3);

RGBchart = readcell('STORAGE2/rgb.txt','NumHeaderLines',3);
```

Así “Dchart” contiene dos columnas con una fila por imagen de la secuencia. La primera columna contiene el “timestamp” de cuándo se capturó la imagen, y la segunda el nombre del archivo de la imagen. “RGBchart” es análoga para con las imágenes a color.

A continuación, se pueden cargar las imágenes indexando la fila correspondiente.

```
IMD1=imread(append('STORAGE2/',Dchart{(StartPoint),2}));

IMRGB1=imread(append('STORAGE2/',RGBchart{(StartPoint),2}));
```

Donde “StartPoint” es un escalar que indica la fila de la primera imagen a cargar. Esto se hace porque pasa un tiempo considerable entre el inicio de la grabación y el comienzo del movimiento.

6.2. CALIBRACIÓN DE LA CÁMARA Y CORRECCIÓN

Llevar a cabo los cálculos necesarios para hallar las posiciones relativas de los objetos representados en una fotografía requiere construir un modelo matemático de la proyección del objeto capturada en la imagen.

El modelo pin-hole, representado en la ilustración 2, es el modelo matemático que nos permite relacionar las coordenadas de los objetos del mundo real y sus coordenadas en la imagen, siendo la imagen una proyección plana de estos.

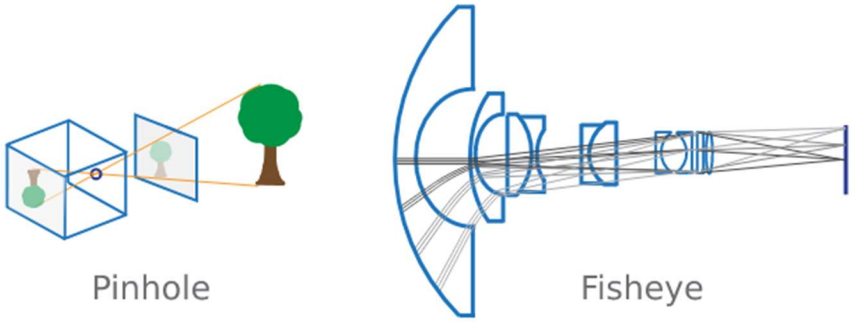

Ilustración 2. Representación gráfica del modelo y de la deformación causada por el sistema de lentes

Conociendo la geometría del modelo específico de la cámara utilizada se puede calcular, partiendo de un punto de la imagen, la recta que pasa por el centro de coordenadas de la cámara y el punto del espacio en el que se encuentra el objeto representado en la imagen. Los parámetros necesarios para construir este modelo son las coordenadas del centro focal de la cámara y la distancia al plano virtual de la proyección de la imagen.

Este modelo simplifica los cálculos a mera trigonometría.

El modelo pin-hole, aunque útil, falla al no tener en cuenta las deformaciones introducidas en la imagen por lentes curvas. Para corregir estas deformaciones el método más utilizado es el de la calibración mediante tablero de ajedrez.

Para utilizar este método es necesaria la captura de múltiples vistas en múltiples posiciones y orientaciones de un tablero de ajedrez con un tamaño de casilla conocido. A través de las deformaciones medibles en las imágenes de casillas que sabemos que son perfectamente cuadradas, podemos extrapolar la transformación geométrica necesaria para corregir dichas imágenes.

Una vez efectuadas las correcciones necesarias, se puede proceder al análisis de las imágenes.

6.3. RECONSTRUCCIÓN TRIDIMENSIONAL

Para la representación tridimensional del entorno, nos basamos en las capturas de la cámara de profundidad que recogen, en cada píxel, el grado de reflexión de la luz infrarroja emitida por un foco próximo al sensor, para cada punto del entorno.

Ese índice es proporcional a la distancia entre la cámara y el objeto reflectante, de manera que, con los parámetros de la cámara necesarios para construir el modelo

pin-hole podemos calcular las coordenadas de los objetos que aparecen en la imagen con respecto a la cámara. El modelo pin-hole nos aportará la dirección y el valor del píxel, la distancia.

Ilustración 3. Imagen de profundidad, a color y reconstrucción de nube de puntos

A este conjunto de coordenadas de puntos se le denomina nube de puntos y será una de las bases, junto con las imágenes RGB convencionales, para mapear el entorno.

El mapeado se llevará a cabo mediante el alineamiento iterativo, tomando un fotograma como referencia y alineando el siguiente con respecto al anterior de forma sucesiva.

Para alinear cada fotograma con su precedente, utilizaremos RGB-ICP, un algoritmo ICP mejorado propuesto por P. Henry et al. [8] que aprovecha la combinación de la información RGB y de profundidad.

Su propuesta, tras el paso del alineamiento, añade un nuevo fotograma al modelo 3D y, en paralelo, ejecuta un proceso de detección de cierre de ciclos utilizando puntos clave para comparar el fotograma actual con las observaciones previas, teniendo en cuenta restricciones espaciales. Si se detecta un cierre de ciclo, es decir si se detecta que el robot está en una localización en la que ha estado previamente, se añadiría una restricción al registro de localizaciones y se iniciaría un proceso de realineamiento global.

7. REGISTRO DE IMÁGENES SECUENCIALES

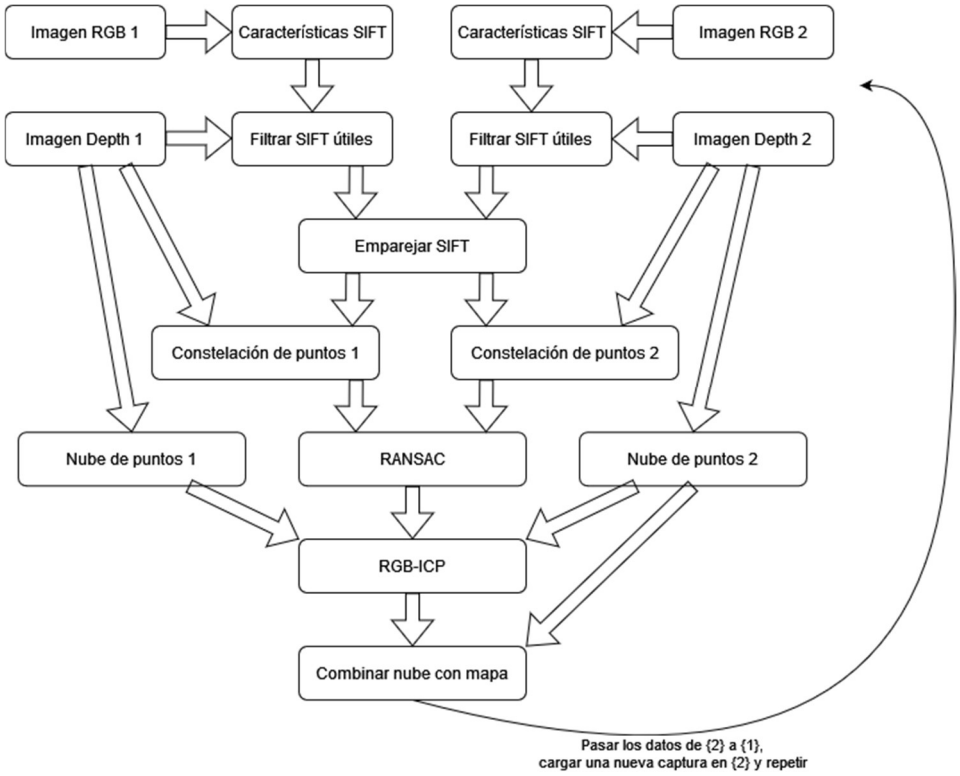

Ilustración 4. Diagrama de flujo del algoritmo

El alineamiento de fotogramas sucesivos se hará en dos partes.

Para agilizar la explicación, convendrá establecer cierta terminología: llamaremos al fotograma recién adquirido el fotograma fuente, y al anterior a este, el fotograma objetivo. De esta manera, la nube de puntos construida a través de la imagen de profundidad del fotograma fuente será la nube de puntos fuente **Ps** y la nube de puntos construida a través de la imagen de profundidad del fotograma objetivo será la nube de puntos objetivo **Pt**.

Primero, en una fase de pre-alinemiento, se extraerán las características SIFT de las imágenes RGB tanto fuente como objetivo, se emparejarán entre si y, mediante RANSAC, se hará el pre-alineamiento basto.

Segundo, para el alineamiento fino, se ejecuta una versión modificada del algoritmo ICP. En esta versión se tienen en cuenta de forma especial las distancias entre las características visuales (SIFT) para la función de error, además de utilizar

algoritmos como la búsqueda en árbol k-d y Levenberg-Marquardt para diferentes partes de este.

7.1. PRE-ALINEAMIENTO

ICP, por su propia naturaleza optimizadora e iterativa, es vulnerable a hallar mínimos locales en la función de error que no se corresponden con el mínimo absoluto o la solución real del alineamiento. Por eso es necesario aplicar un método más robusto, aunque menos preciso, para aproximar las nubes de puntos antes de alinearlas mediante ICP. Para ello, en este proyecto utilizaremos la extracción y emparejamiento de características visuales (SIFT). Una vez obtenidas correspondencias entre puntos con un alto grado de confianza, utilizaremos RANSAC para hallar la transformación rígida que mejor alinee los datos, para luego utilizar esa transformación como punto de partida en el ICP.

7.1.1. CARACTERISTICAS SIFT

Una ventaja clave de las características visuales es que pueden procurar alineamiento sin requerir inicialización. Un detector y descriptor de características extensamente utilizado es SIFT [11]. Este algoritmo detecta y describe puntos clave independientemente de la escala de la imagen.

Para conseguir esto, primero se han de identificar los puntos clave. Esto se logra a través de la convolución de diferentes máscaras construidas a través de diferencias de funciones gaussianas. Buscando los máximos en esas convoluciones, se pueden identificar las ubicaciones (por la posición del máximo) y los tamaños (por la distribución típica (σ) de la máscara empleada) de las características de interés para el algoritmo.

Hallados los puntos clave se procede al cálculo de la orientación de cada punto, calculando el gradiente de los mismos. Después se extrae un fragmento de la imagen que corresponde a un cuadrado de 1,5 veces σ de lado (en pixeles) centrado en el punto clave, y se rota dicho fragmento para alinear la orientación de la característica con el eje horizontal y hacia la derecha.

El siguiente paso es dividir el fragmento en 16 particiones, dividiendo cada lado en 4 partes y se calcula el histograma de gradientes en 8 direcciones para cada partición. Se concatenan los valores de dichos histogramas, consiguiendo un vector de 128 valores, y se normaliza.

De esta manera se construye un descriptor para cada punto clave que resulta invariante a la escala, a la rotación y a cambios moderados de iluminación e identifica cada característica de forma que se pueda reconocer como la misma en otra imagen diferente.

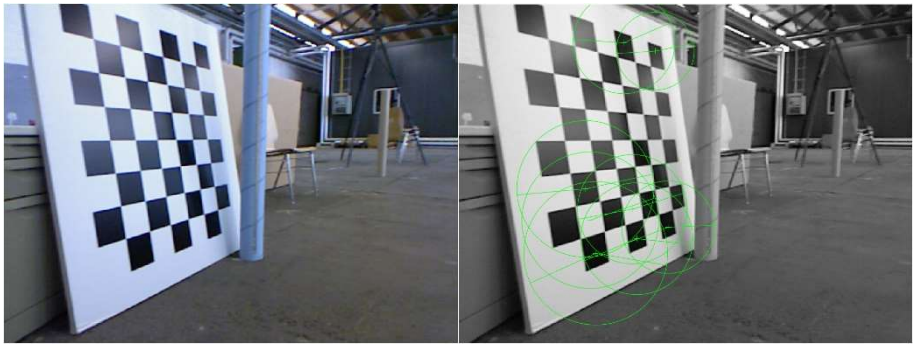

Ilustración 5. a) Imagen RGB. b) Imagen en escala de grises con características SIFT

Tras hacer esto para las dos imágenes, la fuente **Is** y la objetivo **It**, podemos comparar los descriptores de los puntos clave de ambas y buscar coincidencias, estableciendo así emparejamientos entre puntos clave, o lo que es lo mismo, identificando la posición en coordenadas de imagen de los mismos puntos en ambas imágenes. Podemos descartar, por lo tanto, todos los puntos que no encuentren emparejamiento.

El poder emparejar puntos entre dos imágenes nos da la capacidad de estimar una transformación rígida que alinee los puntos de **Is** con los de **It**.

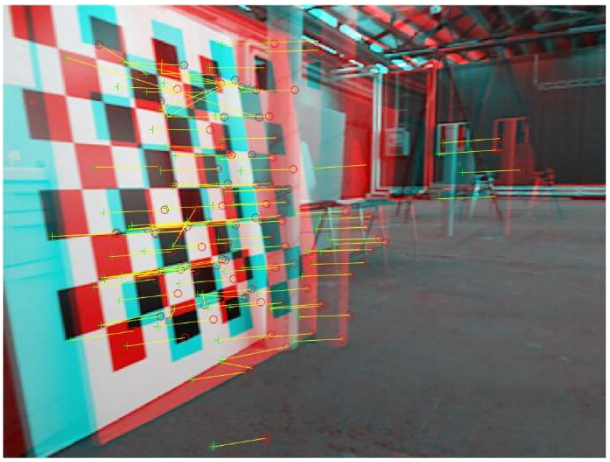

Ilustración 6. Imágenes en escala de grises superpuestas con características SIFT emparejadas

El emparejamiento de una cantidad comparativamente reducida de puntos entre las dos nubes de puntos nos permite utilizar herramientas como el algoritmo RANSAC para determinar una transformación rígida que pre-alinee las nubes de puntos antes de ejecutar el ICP y así evitar los falsos mínimos locales a los que ICP es vulnerable.

7.1.2. CONSTELACIÓN DE PUNTOS

Podemos aprovechar la tridimensionalidad de las coordenadas de nuestros datos para filtrar los pares de puntos que no tengan una medición fiable en la imagen de profundidad, así como para construir con los puntos que superen el filtro dos "constelaciones" de puntos emparejados que nos sirvan para generalizar las correspondencias SIFT halladas en imágenes 2D a nuestro escenario tridimensional.

Para ello no tenemos más que utilizar la geometría del modelo de cámara pin-hole y aplicarle el valor hallado en cada pixel de la imagen de profundidad a la longitud de la recta reproyectada desde el punto focal de la cámara (su origen de coordenadas) para calcular las coordenadas 3D de cada punto relativas a la posición de la cámara.

7.1.3. MODELADO POR RANSAC

Como los puntos clave extraídos por el algoritmo SIFT se emparejan heurísticamente, existe la posibilidad de emparejamientos erróneos, y para filtrarlos, una técnica robusta y eficaz es la implementación de un proceso de RANSAC [12] para encontrar los mejores.

En el paso anterior hemos extraído las características SIFT de las dos imágenes, fuente y objetivo, las hemos emparejado y asociado con sus posiciones en 3D a través de reproyectarlas a partir de la imagen de profundidad. Esto nos deja con dos "constelaciones" (por ser menos densas que una nube) de puntos y las correspondencias entre ellos, como podemos ver en la imagen siguiente.

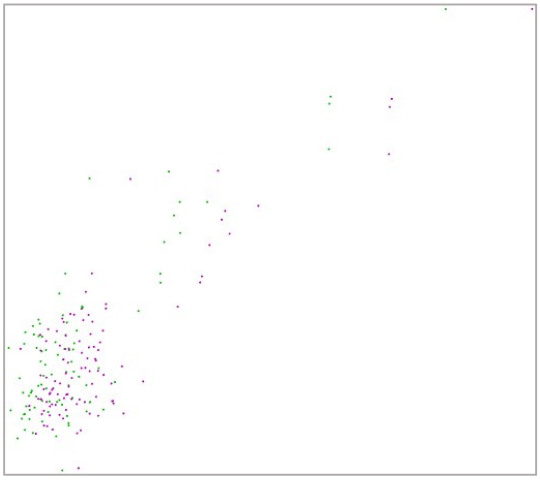

Ilustración 7. Constelaciones de ambas imágenes superpuestas

El algoritmo RANSAC es en esencia un método para hallar el modelo que mejor se adapte a un conjunto de mediciones, y es bastante resistente a la influencia de outliers entre esas mediciones.

Para implementarlo, lo primero que se ha de hacer es decidir el tipo de modelo que se busca, en nuestro caso estamos buscando la transformación rígida (la matriz homogénea) que mejor haga coincidir los puntos de la constelación fuente con la constelación objetivo.

La matriz homogénea es una matriz cuadrada que contiene parámetros para describir la rotación y la traslación necesarias para transformar las coordenadas de un objeto con respecto a una referencia en sus coordenadas con respecto a otra. Esto requiere de 12 parámetros, 3 de traslación y 3 por la rotación de cada eje. Para calcularlos necesitaremos 12 ecuaciones, y como cada punto tiene 3 coordenadas, necesitaremos 4 puntos para definir el problema completamente, 4 parejas de puntos que elegiremos al azar. Podremos hallar la matriz con una sola ecuación matricial si ordenamos las coordenadas de los puntos en forma de matriz.

Una vez calculada la transformación necesitaremos una forma de evaluar su validez. Esto lo haremos aplicándola a todos los pares de puntos y comprobando cuántos quedan lo suficientemente cerca de sus parejas.

Estableceremos una distancia máxima a la que consideraremos que un punto está lo bastante cerca de su pareja y comprobaremos para todos los pares cuales

superan o no esta distancia. Aquellos que no la superen los contaremos como inliers y almacenaremos esta cuenta y la matriz homogénea correspondiente.

Por último, repetiremos este proceso una y otra vez, comparando el número de inliers de cada iteración, y actualizando la matriz que más inliers haya conseguido además del número de ellos. Esto se repetirá hasta que hayamos encontrado un número determinado de soluciones con la misma cuenta de inliers o un número total de veces.

Tras este proceso habremos encontrado la transformación rígida que mejor pre-alinee las constelaciones, que a su vez será la que mejor lo haga con las nubes densas de puntos, derivadas de la reproyección de todos los píxeles de las imágenes de profundidad, y podremos ejecutar el algoritmo ICP para un ajuste más fino sin miedo a caer en soluciones falsas de mínimos locales.

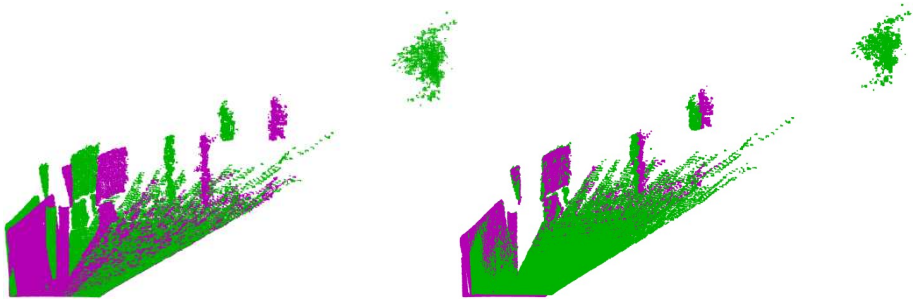

Ilustración 8. a) Nubes de puntos sin alinear. b) Nubes de puntos tras RANSAC

7.2. RGB-ICP

Para implementar el algoritmo ICP primero hemos de determinar la asociación entre los puntos de ambas nubes densas. Para ello aplicaremos a los puntos de la nube fuente la transformación obtenida con RANSAC, y buscaremos en la nube objetivo el punto más cercano a la posición hallada, utilizando una búsqueda rápida en árbol-kd.

Después definiremos una función de error del alineamiento. En primer lugar, definiremos un elemento que dé importancia a la distancia entre los puntos emparejados previamente mediante SIFT. De esa manera evitaremos desalineamientos por deriva, generados por grandes superficies planas. En segundo lugar, definiremos otro término que tenga en cuenta la distancia entre puntos de las nubes densas emparejados por ser los más próximos entre ellos. Estos dos elementos se ponderarán

con un peso α y como esta minimización no tiene solución de forma cerrada, requiere del uso de un optimizador no lineal. La propuesta de P. Henry utiliza Levenberg-Marquardt.

Tras haber encontrado otra transformación que minimiza el error entre los puntos asociados, estos pasos, la asociación y la minimización, se repiten iterativamente. Es importante señalar que los emparejamientos de las características visuales no se recomputan tras la ejecución del RANSAC. El bucle se termina cuando el error no se reduce significativamente (convergencia) o tras un número preestablecido de iteraciones.

Una vez encontrada esta transformación óptima, se le aplica a la nube de puntos fuente y esta se combina con el origen para ir construyendo el mapa del entorno del robot.

Este proceso se repite pasando los datos, nube e imagen, de fuente a ser objetivo, y capturando nuevas imágenes fuente a medida que el robot se desplaza.

7.2.1. ASIGNACIÓN DE PUNTO MÁS CERCANO

Para la asignación del punto de la nube objetivo más cercano a cada punto de la nube fuente podríamos utilizar una estrategia de fuerza bruta, y medir la distancia a cada punto quedándonos con el que dé como resultado la mínima. Pero el cálculo de una distancia incluye una raíz cuadrada, una operación computacionalmente cara, y que de tener que hacer unas pocas veces no resultaría un problema, pero con nubes de aproximadamente 307.200 puntos, nos encontraríamos con 94.400.000.000 de cálculos de distancia por cada iteración de ICP, que suele necesitar cerca de 12 iteraciones para converger en un resultado. Cualquier estrategia que nos sirva para reducir el número de puntos a comprobar será de gran ayuda a la hora mejorar la eficiencia del sistema y de garantizar que es capaz de funcionar a tiempo real. Es decir, de procesar todo lo necesario de un fotograma antes de capturar el siguiente.

Una buena alternativa, y la que utilizaremos, es el algoritmo de búsqueda rápida en árbol k-d. Es una estrategia que reordena los datos para descartar la mayoría de los puntos y comprobar la distancia de unos pocos candidatos. Por lo tanto, son diferenciables dos etapas, la de reordenación o construcción del árbol, que solo se ejecuta una vez por alineamiento de cada nube, y la de selección de candidatos o navegación del árbol, que sí se repite con cada punto.

Para explicar de forma más clara el proceso de construcción del árbol, lo haremos en dos dimensiones, aunque el método es escalable a cualquier numero (k) de ellas. El algoritmo se basa en separar el espacio en sectores con el mismo número de puntos a cada lado, alternando la dimensión en la que se realiza la división. De esta manera se construye una estructura de datos en la que cada punto está representado por un nodo, y conectado a un nodo padre y a uno o dos nodos hijos, o a ninguno si ya es un nodo final.

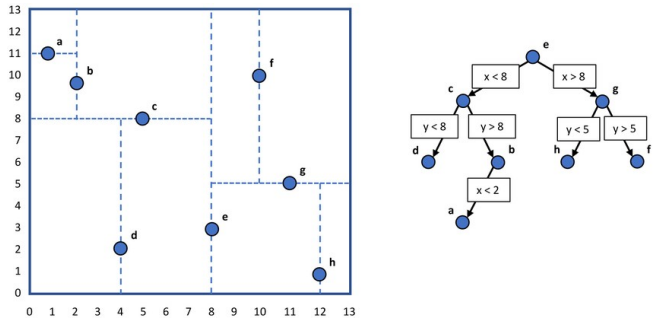

Ilustración 9. Diagrama del proceso árbol k-d en 2D

Empezaremos por elegir arbitrariamente una de las dimensiones del espacio en el que están definidos los puntos, ordenándolos con respecto al valor de la coordenada en ese eje, y seleccionaremos la mediana. Ese punto será el primer nodo o nodo raíz. A continuación, para el subconjunto de puntos que quedan a un lado, ordenaremos los puntos con respecto al valor de la coordenada en el siguiente eje, y seleccionamos la mediana también. Este segundo punto estará vinculado al primero. Repetimos la misma operación con los puntos del otro lado, y tenemos los primeros dos niveles del árbol. Tenemos que repetir este proceso hasta haber seleccionado todos los puntos, y cada uno estará vinculado a otros tres, dos “hijos” y un “padre” y tendrá también almacenadas sus propias coordenadas y la dimensión que le correspondía a la hora de elegirlo como mediana.

A la hora de hallar el vecino más cercano a un punto arbitrario, el proceso será el siguiente. Empezando por el nodo raíz, se compara la coordenada de la dimensión correspondiente y se procede a uno u otro nodo “hijo” dependiendo del resultado. Se repite este proceso hasta llegar a un nodo “hoja” (nodo sin hijos). Entonces se calcula la distancia de este nodo al punto de referencia y como este es el primer nodo que se mide, también se guarda como “actualmente mejor”. Para agilizar el proceso, se utiliza el cuadrado de la distancia, evitando así operar raíces cuadradas.

A continuación, se va desandando el camino recorrido repitiendo los siguientes pasos en cada nodo. Primero se comprueba si este nodo es más cercano al punto de interés que el "actualmente mejor" y de ser así se actualiza este como "actualmente mejor". A continuación, se comprueba si pudiera haber puntos al otro lado del plano de separación, definido por el punto y la dimensión correspondiente, que pudieran estar más cerca que el "actualmente mejor".

Conceptualmente se está comparando la esfera con centro en el punto de interés y de radio la distancia al "actualmente mejor" con el plano correspondiente al punto del nodo que se está comprobando y la dimensión que le corresponde, y comprobando si hay o no intersección. De no haber intersección no hay ninguna posibilidad de que al otro lado haya un candidato mejor y el algoritmo seguirá desandando el camino hasta el nodo raíz, y de haberlo, habrá que entrar a comprobar los nodos derivados de la otra rama del nodo en cuestión. Al estar los planos fijados en las orientaciones de las dimensiones, esta comprobación es muy sencilla. Basta con comparar la distancia al actual mejor con la diferencia entre las coordenadas relevantes del punto del plano y el de interés.

Cuando el algoritmo llega al nodo raíz, la comprobación ha finalizado y tendremos almacenado en "actualmente mejor" el punto de la nube objetivo más cercano a dicho punto de la nube fuente. Esto nos ahorra una cantidad muy elevada de cálculos de distancia y optimiza el proceso, que tendremos que repetir para cada punto de la nube fuente.

7.2.2. FUNCIÓN DE ERROR

Una vez emparejados todos los puntos de la nube fuente, tenemos que definir la función de error que utilizaremos para medir la calidad del alineamiento de las nubes. En este aspecto, la propuesta de P. Henry aporta otro elemento diferenciador con respecto a un algoritmo ICP clásico.

$$error(t) = \alpha\left(\frac{1}{|A_f|}\sum_{i\in A_f} w_i\left|t(f_s^i) - f_t^i\right|^2\right) + (1-\alpha)\left(\frac{1}{|A_d|}\sum_{j\in A_d} w_j\left|\left(t(p_s^j) - p_t^j\right)\cdot n_t^j\right|^2\right)$$

Como podemos apreciar, esta función de error se divide en dos términos ponderados a través de un factor α. El primer término hace referencia al sumatorio de las distancias entre los puntos emparejados mediante características visuales (SIFT). La función $t()$ es la transformación rígida aplicada a los puntos de la nube fuente. En la primera iteración esta será la hallada mediante RANSAC, y en las siguientes será la

resultante de la minimización de esta misma función. El término *w* representa el valor de la confianza en que cada emparejamiento sea correcto, en nuestro caso, al no calcularlo sería siempre 1. El término *A* equivale al número de emparejamientos hallados. Y, por último, el término *n* representa el vector normal de cada punto de la nube objetivo. Esto se introduce al cálculo porque está demostrada la mayor robustez de utilizar la distancia punto a plano que la punto a punto, como veremos en el siguiente apartado.

7.2.3. DISTANCIA PUNTO-PLANO

Puesto que las capturas de las imágenes de profundidad miden la distancia entre el centro focal de la cámara y la superficie del objeto a lo largo de la línea proyectada a través de cada píxel, el punto resultante en una captura no tiene por qué ser el mismo punto del objeto que en otra captura. Debido a esto, la distancia entre ellos puede no ser nula y llevarnos a error si esa es la que intentamos minimizar. Sin embargo, sí que ha de formar parte de la misma superficie, y por lo tanto, la distancia con respecto a la superficie tangencial correspondiente al punto objetivo se acercará más a 0.

Representaremos la superficie tangencial en cada punto de la nube objetivo mediante su vector normal *n.* Para calcular este vector normal, primero hemos de seleccionar un conjunto de puntos "vecinos". Para esto utilizaremos un procedimiento similar al de búsqueda en árbol k-d, y de hecho utilizaremos el mismo árbol que hemos construido para la selección de puntos más cercanos. En está versión, la esfera correspondiente al punto de interés tendrá un radio constante, previamente definido, y se seleccionarán todos los puntos que entren dentro de ese radio. Pero a la hora de navegar el árbol y de decidir qué ramas de este merece la pena navegar, el algoritmo funciona de la misma manera que en la versión de punto más cercano.

Una vez construido el subconjunto de k puntos vecinos P^k donde cada punto $p_i \in P^k$ está representado por sus coordenadas $p_i = \left\{p_{i_x}, p_{i_y}, p_{i_z}\right\}$ pasamos a aproximar el plano que estos definirían. Para eso vamos a representar el plano con un punto x y un vector normal $\vec{n}$ de modo que la distancia entre un punto $p_i \in P^k$ y el plano queda definida como:

$$d_i = (p_i - x)\vec{n}$$

Además, x se define como el centroide de P^k de la siguiente manera:

$$x = \frac{1}{k}\sum_{i=1}^{k} p_i$$

Considerando lo anterior, se toman valores de x y $\vec{n}$ de forma que, resolviendo un problema de mínimos cuadrados, d_i sea cero para utilizar la mejor aproximación posible del plano tangente a la superficie de la nube en p_i.

Finalmente, la solución para $\vec{n}$ se da estudiando los autovalores y autovectores de la matriz de covarianzas $C \in \mathbb{R}3x3$ de P^k calculada como:

$$C = \frac{1}{k}\sum_{i=1}^{k}(p_i - p)(p_i - p)^T \; ; \; Cv_j = \lambda_j v_j \; ; \; j \in \{0,1,2\}$$

C es simétrica y semidefinida positiva con autovalores $\lambda_j \in \mathbb{R}$ y autovectores $\overrightarrow{v_j}$.

Los autovectores son ortogonales entre sí y dan una aproximación de las principales componentes de P^k. No solo eso, sino que si se cumple que:

$$0 \leq \lambda_0 \leq \lambda_1 \leq \lambda_2$$

Entonces el autovector $\overrightarrow{v_0}$, el cual se corresponde con el autovalor de menor valor λ_0, es una aproximación de $\vec{n} = \{n_x, n_y, n_z\}$, vector normal a la superficie en el punto p_i de la nube estudiada P^k.

Debido a que no hay forma estricta de determinar el signo del vector normal, su orientación es ambigua tras ser calculada con los procedimientos indicados anteriormente. Esto significa que las normales estimadas en la superficie de una nube de puntos no están consistentemente orientadas.

La solución para este problema es sencilla si se conoce la posición desde la cual se ha adquirido la nube, es decir, la posición de la cámara. Para orientar todas las normales $\overrightarrow{n_i}$ hacia el punto de vista del sensor v_p se debe cumplir:

$$\overrightarrow{n_i}(v_p - p_i) > 0$$

Una vez calculado el vector normal de cada punto de la nube objetivo, calcular la distancia punto plano es tan sencillo como multiplicar escalarmente el vector de distancia entre los puntos por el vector normal. Tal como se indica en el apartado de la función de error.

7.2.4. MINIMIZACIÓN DEL ERROR

Con todo esto hemos conseguido calcular todos los elementos necesarios para construir la función de error definida anteriormente.

$$error(t) = \alpha\left(\frac{1}{|A_f|}\sum\nolimits_{i\in A_f} w_i\left|t(f_s^i) - f_t^i\right|^2\right) + (1-\alpha)\left(\frac{1}{|A_d|}\sum\nolimits_{j\in A_d} w_j\left|\left(t(p_s^j) - p_t^j\right)\cdot n_t^j\right|^2\right)$$

La función de transformación $t(x)$ donde $x \in \mathbb{R}^3$ a su vez está definida como:

$$t(x) = R(\vartheta,\varphi,\gamma)x + \left[\tau_x,\tau_y,\tau_z\right]^{\mathrm{T}}$$

Y a su vez,

$$R(\vartheta,\varphi,\gamma) = \begin{bmatrix} c(\gamma) & -s(\gamma) & 0 \\ s(\gamma) & c(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} c(\varphi) & 0 & s(\varphi) \\ 0 & 1 & 0 \\ -s(\varphi) & 0 & c(\varphi) \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 \\ 0 & c(\vartheta) & -s(\vartheta) \\ 0 & s(\vartheta) & c(\vartheta) \end{bmatrix}$$

$$R(\vartheta,\varphi,\gamma) = \begin{bmatrix} c(\gamma)c(\varphi) & c(\gamma)s(\varphi)s(\vartheta) - s(\gamma)c(\vartheta) & c(\gamma)s(\varphi)c(\vartheta) + s(\gamma)s(\vartheta) \\ s(\gamma)c(\varphi) & s(\gamma)s(\varphi)s(\vartheta) + c(\gamma)c(\vartheta) & s(\gamma)s(\varphi)c(\vartheta) - c(\gamma)s(\vartheta) \\ -s(\varphi) & c(\varphi)s(\vartheta) & c(\varphi)c(\vartheta) \end{bmatrix}$$

Donde ϑ, φ y γ representan los ángulos de rotación en los ejes x, y y z respectivamente, τ_x , τ_y y τ_z representan traslaciones en los ejes x, y y z respectivamente y $c(\ \ )$ y $s(\ \ )$ representan las funciones coseno y seno, abreviadas para facilitar la lectura.

Por lo tanto, de forma explícita:

$$\begin{aligned} &error(\tau_x,\tau_y,\tau_z,\vartheta,\varphi,\gamma) \\ &\quad= \alpha\left(\frac{1}{|A_f|}\sum\nolimits_{i\in A_f} w_i\left|R(\vartheta,\varphi,\gamma)f_s^i + \left[\tau_x,\tau_y,\tau_z\right]^{\mathrm{T}} - f_t^i\right|^2\right) \\ &\quad+ (1-\alpha)\left(\frac{1}{|A_d|}\sum\nolimits_{j\in A_d} w_j\left|\left(R(\vartheta,\varphi,\gamma)p_s^j + \left[\tau_x,\tau_y,\tau_z\right]^{\mathrm{T}} - p_t^j\right)\cdot n_t^j\right|^2\right) \end{aligned}$$

Si denominamos $\vec{t}$ al vector que contiene los parámetros de la transformación $[\tau_x \quad \tau_y \quad \tau_z \quad \vartheta \quad \varphi \quad \gamma]^{\mathrm{T}}$ podemos definir la función de error como:

$$error(\vec{t}) = \alpha\left(\frac{1}{A_f}\sum\nolimits_{i\in A_f} f_i^2(\vec{t})\right) + (1-\alpha)\left(\frac{1}{A_d}\sum\nolimits_{j\in A_d} g_j^2(\vec{t})\right)$$

Se trata de un problema no lineal de mínimos cuadrados. Es no lineal debido a la presencia de funciones trigonométricas introducidas por la matriz de rotación.

Podemos ver este problema como la resolución de un sistema de ecuaciones no lineales con más ecuaciones que incógnitas, y por tanto entenderlo como un problema de optimización.

Podemos también definir el vector residual de cada sumando de nuestra función como:

$$f(t) = \left(f_1(t), f_2(t), \dots, f_{A_f}(t)\right)^{\mathrm{T}} \; ; \; g(t) = \left(g_1(t), g_2(t), \dots, g_{A_d}(t)\right)^{\mathrm{T}}$$

Las matrices jacobianas de estos vectores residuales son $J_f(t) \in \mathbb{R}^{m \times n}$

$$J_f(t) = \left[\frac{\partial f_i}{\partial t_j}\right]_{\substack{i=1,2,\dots,A_f \\ j=1,2,\dots,n}} = \begin{bmatrix} \nabla f_1(t)^{\mathrm{T}} \\ \nabla f_2(t)^{\mathrm{T}} \\ \vdots \\ \nabla f_m(t)^{\mathrm{T}} \end{bmatrix}$$

$$J_f(t)_{i,j} = \frac{\partial f_i}{\partial t_j}$$

Por otra parte, la matriz hessiana de cada residuo $f_i(t)$ es:

$$G_i(t) = \nabla^2 f_i(t) \in \mathbb{R}^{n \times n}; \; G_i(t)_{jk} = \frac{\partial^2 f_i(t)}{\partial t_j \partial t_k} \; ; i = 1,2, \dots, A_f$$

Con esta notación se puede reescribir la función objetivo como:

$$F(t) = \frac{1}{A_f} \sum_{i=1}^{A_f} f_i^2(t) = \frac{1}{A_f} \|f(t)\|_2^2$$

El gradiente y la matriz hessiana de esta función objetivo, $F(t)$, se puede expresar en términos de las matrices jacobiana $J(t)$ y hessiana $G(t)$ de la siguiente manera:

$$\nabla F(t) = \sum_{i=1}^{A_f} f_i(t) \nabla f_i(t) = J(t)^T f(t)$$

$$\nabla^2 F(t) = \sum_{i=1}^{A_f} \nabla f_i(t) \, \nabla f_i(t)^T + \sum_{i=1}^{A_f} f_i(t) \, \nabla^2 f_i(t) = J(t)^T J(t) + Q(t)$$

Donde $Q(t) = \sum_{i=1}^{A_f} f_i(t) \, G_i(t)$

Las derivadas de los residuos son relativamente sencillas y por ende el cálculo de la matriz jacobiana $J(t)$ es relativamente sencillo, pudiendo así deducir el gradiente de $F(t)$ con un pequeño coste operativo. Por otra parte, la hessiana se compone de dos términos, y el primero puede obtenerse sin calcular ninguna derivada segunda, por lo que es deducible fácilmente utilizando la jacobiana previamente calculada. Además, el primer término es mucho más importante, ya que los residuos tienden a ser muy pequeños.

Partiendo de una trasformación inicial t_o calculada a través de RANSAC, se busca una solución $(t_o + \Delta t)$ para la que el error sea mínimo. Es aquí cuando, a pesar de que $F(t)$ no es lineal, se busca linealizar $f(t_o + \Delta t)$ llegando a un problema lineal de mínimos cuadrados de la forma:

$$m_k(\Delta t_k) = \|f(t_{k-1}) + J(t_{k-1})\Delta t_k\|_2$$

La linealización no va ser válida para para cualquier valor de Δt_k, por lo tanto debemos añadir la condición:

$$\|D_k \Delta t_k\|_2 \leq \delta_k$$

Donde δ_k marca el radio de una región de confianza y D_k es una matriz no singular que tiene en cuenta el problema del escalado. Es importante tener en cuenta este problema, que se da si al cambiar el vector t en una determinada dirección se produce un cambio mayor que si se hubiera realizado el mismo cambio en una dirección distinta con el mismo módulo. La matriz no singular, diagonal y positiva, D_k proporciona una invarianza de escala.

La elección de δ_k dependerá del ratio entre la reducción actual y la predicha, o lo que es lo mismo:

$$\rho_k = \frac{\|f(t_{k-1})\|_2^2 - \|f(t_{k-1}) + \Delta t_k\|_2^2}{\|f(t_{k-1})\|_2^2 - \|f(t_{k-1}) + J(t_{k-1})\Delta t_k\|_2^2}$$

Con todos los elementos calculados, podemos ejecutar el algoritmo de Levenberg-Marquardt.

Dados unos valores $t_o, D_o \; y \; \delta_o$ iniciales, y un valor $\beta \in (0,1)$, para $k = 1,2, \ldots$

1. Se calcula $\|f(t_k)\|_2^2$
2. Se determina la solución Δt_k de

$$\min_{\Delta t}\|f(t_{k-1}) + J(t_{k-1})\Delta t\|_2^2, \text{ con } \|D_k\Delta t\|_2 \leq \delta_k$$

3. Se calcula el valor de ρ_k según la ecuación dada anteriormente.
4. Si $\rho_k \leq \beta$ entonces $t_{k+1} = t_{k-1}$ y $J_{k+1} = J_{k-1}$.

   Si $\rho_k > \beta$ entonces $t_{k+1} = t_{k-1} + \Delta t$ y se calcula J_{k+1}.
5. Se actualizan D_k y δ_k.

   Si $\rho_k \leq {}^1/_4$ entonces se escoge $\delta_{k-1} \in \left[\frac{1}{10}\delta_k, \frac{1}{2}\delta_k\right]$

   Si $\rho_k \in \left(\frac{1}{4}, \frac{3}{4}\right)$ y $\lambda = 0$, o si $\rho_k > {}^3/_4$ entonces $\delta_{k+1} = 2\|D_k\Delta t\|_2$

   En otro caso, $\delta_{k+1} = \delta_k$

   $D_{k+1} = diag(d_1^{k+1}, \dots, d_n^{k+1})$, donde $d_i^{k+1} = máx\{d_i^k, \|\partial_i f(t_{k+1})\|_2\}$

Esto quiere decir, que cambiaremos las dimensiones de la región de confianza si se alcanza su frontera en la iteración anterior o si se ha perdido la aproximación lineal deseada.

Si la solución del paso 2 está dentro de la región de confianza ya habremos acabado, siendo ésta $(t + \Delta t)$ la solución de esta iteración del ICP, si no, se continuara iterativamente hasta alcanzar la región de confianza o hasta la convergencia.

Tras hallar una nueva transformación que minimiza la distancia entre todos los elementos emparejados, estos se re-emparejan y se repite el proceso iterativamente hasta un máximo de iteraciones o hasta la convergencia.

Se fusionan las nubes de puntos con la trasformación resultante del RGB-ICP, se pasan los datos de la captura fuente a las variables designadas para la captura objetivo y se toma una nueva captura, repitiendo el proceso una y otra vez.

8. DETECCIÓN DE CIERRE DE CICLOS, GRAFO DE POSES Y OPTIMIZACIÓN GLOBAL

El alineamiento entre fotogramas sucesivos es un buen método para seguir el movimiento del robot en distancias moderadas. Sin embargo, posibles errores de alineamiento entre algunos fotogramas y el ruido en las mediciones hacen que la estimación de la posición del robot se desvíe con el tiempo, causando imprecisiones en el mapa. Esto es especialmente visible cuando el robot sigue un largo recorrido para volver a algún lugar en el que ya había estado previamente. El error acumulativo tiende a dar como resultado un mapa que muestra dos representaciones del mismo

lugar en diferentes posiciones. A esto se lo conoce como el problema de cierre de ciclos.

La solución propuesta por P. Henry tiene dos partes, la detección de cierre de ciclos y la corrección del mapa para combinar las regiones duplicadas.

8.1. DETECCIÓN DE CIERRE DE CICLOS

La estrategia general que presenta es la de representar las restricciones en forma de grafo, con los bordes entre fotogramas correspondiéndose con restricciones geométricas.

Las transformaciones relativas del alineamiento de fotogramas secuenciales aportan algunas restricciones, por lo tanto, sin cierres de ciclos, el grafo consiste en una cadena lineal de fotogramas. Los cierres de ciclo se representan como restricciones entre fotogramas que no son temporalmente adyacentes.

Para mantener el grafo relativamente ligero se definen unos fotogramas clave, que serán un subconjunto de los fotogramas alineados. Estos fotogramas clave se determinan basándose en solapamiento visual, adaptando así la densidad de fotogramas clave al movimiento del robot y la naturaleza del entorno.

Tras alinear un fotograma **F** utilizaremos sus características SIFT y el algoritmo RANSAC previamente desarrollado para encontrar una transformación rígida que lo alinee con el último fotograma clave. Mientras el número de inliers RANSAC se mantenga por encima de un umbral, no necesitaremos añadir **F** a los fotogramas clave. A medida que el robot se siga desplazando, la vista de este contendrá cada vez menos características 3D emparejables con el último fotograma clave. De esa manera, el primer fotograma que falle en emparejarse con el último fotograma clave, se convertirá en el próximo fotograma clave.

Cada vez que se añada un nuevo fotograma clave, intentaremos detectar un cierre de ciclo. Comprobaremos los inliers del emparejamiento de características del nuevo fotograma clave con todos los anteriores. Si el procesamiento de RANSAC produce suficientes emparejamientos geométricamente consistentes, se habrá detectado un cierre de ciclo, y se añadirá un nuevo borde al grafo representando la restricción descubierta.

8.2. OPTIMIZACIÓN GLOBAL

Para corregir el error de desviación acumulada cuando se detecta un cierre de ciclo, P. Henry propone utilizar el método TORO (Tree-based netwORk Optimizer). Este en un sistema que eficientemente minimiza el error en grafos en los que los vértices están parametrizados por sus componentes de traslación y rotación y los bordes representan las restricciones entre los parámetros con matrices de covarianza asociados. TORO utiliza el algoritmo de gradiente descendente estocástico para maximizar la fiabilidad de los parámetros de los vértices sujetos a restricciones.

9. PRESUPUESTO FINAL

Concepto	Valor de mercado (€)	Amortización (5 meses) (€)
Equipos		
Lenovo ideapad 330S	599	62,4
Intel® RealSense™ D455	387	40,3
Software		
Matlab R2022b (licencia de 1 año)	1830	762,5
Dataset	0	0
Mano de obra		
750 horas, 12.6€/h	9.491	9.491
Total gastos		10.356,2
Gastos de la empresa		
15% coste indirecto		11.909,6
20% beneficio industrial		14.291,5
Total sin IVA		14.291,5
21% IVA		17.292,8
TOTAL PRESUPUESTO		17.292,8

- Amortización de los equipos a 4 años.
- Software de licencia anual ponderado al intervalo de uso.

10. CONCLUSIÓN Y VIAS DE FUTURO

El objetivo original de este trabajo era la implementación de un algoritmo de SLAM completo basado en una cámara de profundidad. A pesar del esfuerzo puesto en este proyecto, no ha sido posible completar dicho objetivo, quedando la implementación limitada a la captura de imágenes y el prealineamiento de estas. Esto se debe a que, aunque tras un extenso periodo de estudio de diferentes estrategias la propuesta de P. Henry resultó ser la más completa y prometedora, la modificación de la función de error contenida en el RGB-ICP obliga a la implementación propia de los sub-algoritmos que lo componen. Ello aumenta significativamente la complejidad del proyecto por la ausencia de librerías pre-existentes que contengan estas versiones modificadas de los algoritmos que requiere el planteamiento teórico.

Planteamiento teórico que aborda diferentes ámbitos, como el análisis de datos en la búsqueda rápida en árbol k-d o la optimización de sistemas no lineales en el algoritmo Levenberg-Marquardt. Cada uno de estos ámbitos con entidad suficiente como para afrontarse, en su conjunto, de forma colaborativa.

11. BIBLIOGRAFÍA

[1] J. Sturm, N. Engelhard, F. Endres, W. Burgard and D. Cremers "*A Benchmark for the Evaluation of RGB-D SLAM Systems",* Proc. of the International Conference on Intelligent Robot Systems (IROS), 2012.

[2] Julio A. Placed, Jared Strader, Henry Carrillo, Nikolay Atanasov, Vadim Indelman, Luca Carlone, José A. Castellanos *"A Survey on Active Simultaneous Localization and Mapping: State of the Art and New Frontiers*", Cornell University, 2022.

[3] Harold S. Stone, "A Fast Direct Fourier-Based Algorithm for Subpixel Registration of Images", IEEE Transactions on Geoscience and Remote Sensing, V. 39, No. 10, Oct. 2001, pp.2235-2242

[4] Li, Leihui; Wang, Riwei; Zhang, Xuping *"A Tutorial Review on Point Cloud Registrations: Principle, Classification, Comparison, and Technology Challenges"*. Mathematical Problems in Engineering. Hindawi. (2021).

[5] Myronenko, A., and X. Song. "Point Set Registration: Coherent Point Drift. "*Proceedings of IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*." Vol 32, Number 12, December 2010, pp. 2262–2275.

[6] K. S. Arun, T. S. Huang and S. D. Blostein, "Least-Squares Fitting of Two 3-D Point Sets," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-9, no. 5, pp. 698-700, Sept. 1987

[7] Juyong Zhang, Yuxin Yao, Bailin Deng, "Fast and Robust Iterative Closest Point" in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 44, no. 07, pp. 3450-3466, 2022

[8] Peter Henry, Michael Krainin, Evan Herbst, Xiaofeng Ren, and Dieter Fox "RGB-D Mapping: Using Depth Cameras for Dense 3D Modeling of Indoor Environments", *Experimental Robotics,* Springer Tracts in Advanced Robotics 79, 2014.

[9] **"**Intel® RealSense™ SDK 2.0" [Online] Available: https://github.com/IntelRealSense/librealsense [Accessed: 8 February 2023]

[10] Juyon Zhang, Yuxin Yao, Bailin Deng *"Fast and Robust Iterative Closest Point"*, IEEE 2020.

[11] "Local Feature Detection and Extraction - MATLAB & Simulink - Mathworks España" [Online] Available: https://www.mathworks.com/help/vision/ug/local-feature-detection-and-extraction.html [Accessed: 8 February 2023]

[12] Data Analytics, "RANSAC Regression" [Online] Available: https://vitalflux.com/ransac-regression-explained-with-python-examples/ [Accessed: 9 February 2023]

ANEXOS

A1. CÓDIGO DEL PROGRAMA PRINCIPAL

```
% INICIALIZACIÓN DE VARIABLES
H_total=[1 0 0 0;0 1 0 0;0 0 1 0;0 0 0 1];
Dchart = readcell('STORAGE2/depth.txt','NumHeaderLines',3); % LECTURA DE LOS
NOMBRES DE LOS ARCHIVOS DE IMÁGENES
RGBchart = readcell('STORAGE2/rgb.txt','NumHeaderLines',3); % (equivalente a
inicialización de la cámara de profundidad (D) y convencional (RGB))

StartPoint= 20;
CapStep=5;
StepNum=20;

%LECTURA DEL PRIMER PAR DE IMÁGENES (equivalente a capturar imágenes con la
cámara)
IMD1=imread(append('STORAGE2/',Dchart{(StartPoint),2}));
IMRGB1=imread(append('STORAGE2/',RGBchart{(StartPoint),2}));

pC1=D2pC(IMD1); %Conversión de la imagen de profundidad a nube densa de puntos

IMGS1=rgb2gray(IMRGB1); %Conversión a escala de grises de la imagen
convencional
pSIFT1=detectSIFTFeatures(IMGS1); %Detección de las caracteristicas SIFT de la
primera imágen

filteredSIFT=SIFTPoints(ones(pSIFT1.Count,2));
b=1;
for a=1:pSIFT1.Count
    loc=fix(pSIFT1(a).Location);
    if IMD1(loc(2),loc(1)) ~= 0
        filteredSIFT(b)=pSIFT1(a);
        b=b+1;
    end
end
selectedSIFT=filteredSIFT(1:b,:);

[SIFTfeatures1,SIFTvalidPoints1]=extractFeatures(IMGS1,selectedSIFT);
Mapa=pC1;
%--------------------------------------------------------------------------
for loop=1:StepNum

    IMD2=imread(append('STORAGE2/',Dchart{(StartPoint+(loop*CapStep)),2}));

IMRGB2=imread(append('STORAGE2/',RGBchart{(StartPoint+(loop*CapStep)),2}));

    pC2=D2pC(IMD2);
    IMGS2=rgb2gray(IMRGB2);

    pSIFT2=detectSIFTFeatures(IMGS2);

    filteredSIFT=SIFTPoints(ones(pSIFT2.Count,2));
    b=1;

    for a=1:pSIFT2.Count
        loc=fix(pSIFT2(a).Location);
        if IMD2(loc(2),loc(1)) ~= 0
            filteredSIFT(b)=pSIFT2(a);
            b=b+1;
```

```
        end
    end

    selectedSIFT=filteredSIFT(1:b,:);
    [SIFTfeatures2,SIFTvalidPoints2]=extractFeatures(IMGS2,selectedSIFT);

    indexPairs = matchFeatures(SIFTfeatures1,SIFTfeatures2);
    matchedPoints1 = SIFTvalidPoints1(indexPairs(:,1),:);
    matchedPoints2 = SIFTvalidPoints2(indexPairs(:,2),:);

    SIFTpC1=SIFT2pC(matchedPoints1,IMD1);
    SIFTpC2=SIFT2pC(matchedPoints2,IMD2);

%     fprintf('Pairs: %i \n',SIFTpC1.Count);
    if SIFTpC1.Count > 8

[H_ransac,Af,NumInliers]=SIFTpCransac(SIFTpC1.Location,SIFTpC2.Location,500,0.
1);

        H_total=H_total*H_ransac;

        PointsIna=transpose([pC2.Location ones(size(pC2.Location,1),1)]);
        PointsOut=H_total*PointsIna;
        pC2_preAligned = pointCloud(transpose(PointsOut(1:3,:)));

        Mapa=pcmerge(Mapa,pC2_preAligned,0.05);

        IMD1=IMD2;
        SIFTfeatures1=SIFTfeatures2;
        SIFTvalidPoints1 = SIFTvalidPoints2;
    end

end
```

A2. CÓDIGO FUNCIÓN D2pC

```
function [outputArg1] = D2pC(IM_IR)
% Cálculo de una nube de puntos partiendo de una imagen de profundidad
% https://vision.in.tum.de/data/datasets/rgbd-dataset/download
% INTRINSIC CAMERA PARAMETERS Freiburg 2 IR
    fx = 580.8;
    fy = 581.8;
    cx = 308.8;
    cy = 253;
% Freiburg 2 depth correction factor
    factor = 5000; %Para cm
    %dF = 1; %Comentado por ser igual a 1. No se utiliza.

%START_____________________________________________________________
Size_IR = size(IM_IR);
xyzPoints=zeros(Size_IR(1)*Size_IR(2),3);
count=1;
    for v = 1:Size_IR(1) %in range(depth_image.height):
        for u = 1:Size_IR(2) %in range(depth_image.width):
            if IM_IR(v,u) ~= 0 %Si hay punto

                Y = double(IM_IR(v,u)) / factor;
                X = (u - cx) * Y / fx;
                Z = (v - cy) * Y / fy;

                xyzPoints(count,:)    =   [X Y -Z];  %la inversión del eje z
se debe a que en la imagen el eje y es positivo hacia abajo
                count=count+1;
            end
        end
    end

pointsPacked=xyzPoints(1:count,:);
outputArg1=pointCloud(pointsPacked);

end
```

A3. CÓDIGO FUNCIÓN SIFT2pC

```
function [outputArg1] = SIFT2pC(SIFTpoints,IMD)
% Cálculo de la constelación 3D de puntos de las caracteristicas SIFT
% https://vision.in.tum.de/data/datasets/rgbd-dataset/download

% INTRINSIC CAMERA PARAMETERS Freiburg 2 IR
    fx = 580.8; %Distancia focal en el eje x
    fy = 581.8; %Distancia focal en el eje y
    cx = 308.8; %Posicioón en el eje x del centro focal en coordenadas de
imagen
    cy = 253;   %Posicioón en el eje y del centro focal en coordenadas de
imagen

% Freiburg 2 depth correction factor
    factor = 5000; %Para cm
    %dF = 1; %Comentado por ser igual a 1. No se utiliza.

% START_____________________________________________________________________
pointNum = length(SIFTpoints);
xyzPoints=zeros(pointNum,3);

    for v = 1:pointNum %in range(depth_image.height):
            loc=fix(SIFTpoints(v).Location);

                Y = double(IMD(loc(2),loc(1))) / factor;
                X = (loc(1) - cx) * Y / fx;
                Z = (loc(2) - cy) * Y / fy;

                xyzPoints(v,:)    =   [X Y -Z]; %la inversión del eje z se
debe a que en la imagen el eje y es positivo hacia abajo
   end


outputArg1=pointCloud(xyzPoints);

end
```

A4. CÓDIGO FUNCIÓN SIFTpCransac

```
function [H_best,Af,InlierMax] =
SIFTpCransac(pCfixed,pCmoving,testNum,ModelSphereRad)

pairNum=size(pCfixed,1);
InlierMax=0;
Dmed_best=0;
NumBestT=0;

%____Point Selection_____________________________________________________
%_Randomly picked
for j=1:testNum
pair1=randi(pairNum);
while true
    pair2=randi(pairNum);
    if pair2 ~= pair1
        break;
    end
end
while true
    pair3=randi(pairNum);
    if pair3 ~= pair1 && pair3 ~= pair2
        break;
    end
end
while true
    pair4=randi(pairNum);
    if pair4 ~= pair1 && pair4 ~= pair2 && pair4 ~= pair3
        break;
    end
end

%____Point Extraction____________________________________________________

p1f=pCfixed((pair1),:);
p2f=pCfixed((pair2),:);
p3f=pCfixed((pair3),:);
p4f=pCfixed((pair4),:);

p1m=pCmoving((pair1),:);
p2m=pCmoving((pair2),:);
p3m=pCmoving((pair3),:);
p4m=pCmoving((pair4),:);

%_Homogeneous Transform Matrix Calculation_______________________________

FixedMatrix=  (cart2hom([p1f; p2f; p3f; p4f]))';
MovingMatrix= (cart2hom([p1m; p2m; p3m; p4m]))';

H=FixedMatrix/MovingMatrix;

%____Transform Test______________________________________________________

PointsIn=ones(pairNum,4);
PointsIn(:,1:3)=pCmoving;
PointsIn=transpose(PointsIn);

Tform_pC_Loc=H*PointsIn;

Inliers=0;
```

```
Dmed=0;
    for i=1:pairNum
        D=sqrt(((pCfixed(i,1)-Tform_pC_Loc(1,i))^2)+((pCfixed(i,2)-
Tform_pC_Loc(2,i))^2)+((pCfixed(i,3)-Tform_pC_Loc(3,i))^2));
        Dmed=Dmed+D;
        if D<ModelSphereRad
            Inliers=Inliers+1;
        end
    end
    Dmed=Dmed/pairNum;
    if Inliers==InlierMax
        NumBestT=NumBestT+1;
        if Dmed < Dmed_best
            H_best=H;
            Af=[pair1 pair2 pair3 pair4];
        end
    end
    if Inliers>InlierMax
        InlierMax=Inliers;
        H_best=H;
        Af=[pair1 pair2 pair3 pair4];
    end

    if InlierMax > pairNum/3
        break;
    end

end


%_____END_____________________________________________________________________________
```