Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

Grado en Ingeniería Electrónica Industrial y Automática

Integración de un sistema de visión estereoscópico en un entorno de cirugía laparoscópica

Autor:

Estop Remacha, Rafael

Tutor:

Fuente López, Eusebio de la
Ingeniería de sistemas y Automática

Valladolid, Julio 2017.

Resumen

La cirugía laparoscópica, conocida también como “cirugía mínimamente invasiva” ha supuesto una revolución en el ámbito de la medicina. Sin embargo, su gran complejidad requiere del constante desarrollo de métodos que mejoren y faciliten las tareas que se llevan a cabo durante las intervenciones. En las últimas décadas, la visión artificial ha representado un papel fundamental en la realización de estas mejoras.

En el siguiente trabajo de fin de grado se expone el desarrollo de una aplicación para guiar a un robot quirúrgico que asiste al cirujano durante intervenciones de laparoscopia (HALS). Para ello se capturarán imágenes de la mano intracavitaria y éstas serán tratadas mediante visión artificial para poder determinar tanto la ubicación como la posición de la mano y así, poder guiar al robot.

Abstract

The laparoscopic surgery, is known as “minimally invasive surgery” has supposed a revolution in the medicine´s field. However, his great complexity requires the constant development of methods which improve and facilitate the tasks that are carried out during the inventions. In the last years, the artificial vision has played an important role in the development of these improvements.

In the following final project, it discusses the development of an application to guide a surgical robot assisting the surgeons during laparoscopic (HALS) interventions. For this, images of intracavitary hand will be captured and these will be treated by artificial vision to be able to determine the location and the position of the hand and finally, it will be able to guide the robot.

Palabras Clave

Laparoscopia, Visión Artificial, Estereovisión, Tiempo de vuelo, Nube de puntos

Keywords

Laparoscopy, Artificial Vision, Stereovision, Time of flight, Point Cloud

Agradecimientos

“A mi tutor Eusebio, ya que, gracias a su dedicación y su confianza en mí, ha sido posible realizar este trabajo, y en general a todos aquellos profesores que deciden ser un apoyo y no una piedra en el camino del alumno y que trabajan día a día para mejorar la calidad de la enseñanza pública.

A mi familia, a mis padres y mi hermano, que sin su constante apoyo y cariño hubiera sido imposible llegar hasta aquí.

Mario, Christian, Pablo, Adri, Diana, Alberto, Carol… Y tantos otros amigos que se quedan sin nombrar, gracias por el apoyo y los buenos ratos que pasamos y que espero que sigamos pasando mucho tiempo.

A Julián, que sé que siempre estará dispuesto a ayudarme y a escucharme, compañero de batallas.

Y, por último, a dos personas muy especiales en mi vida, a mi abuelo, cuya felicidad siempre me fascinó, de él aprendí a valorar cada momento de la vida. Y a mi tío Chus, que con su amor incondicional me enseñó a querer y a apreciar a las personas de mi alrededor, a disfrutar de viajar, y a disfrutar de la vida.”

Índice

Índice de ilustraciones

1. Introducción

El desarrollo de este Trabajo Fin de Grado se ha realizado en el departamento de Ingeniería de Sistemas y Automática de la Escuela de Ingenierías Industriales de la Universidad de Valladolid. Siendo este trabajo perteneciente a uno de los proyectos de investigación que se están realizando en el departamento.

Para la realización del mismo se han utilizado los avances realizados por compañeros que hicieron su Trabajo de Fin de Grado o Máster en cursos anteriores.

1.1. Justificación

Tras la invención de la cirugía laparoscópica o "cirugía mínimamente invasiva" el mundo de la medicina sufrió una revolución. Los estudios y análisis realizados durante los últimos años afirman que la cirugía laparoscópica ha sido uno de los mayores logros tecnológicos de finales del siglo XX, [1].

Pero todos estos logros son debidos al trabajo de muchos profesionales e investigadores de una gran cantidad de ramas del conocimiento como pueden ser la medicina, química, física, biotecnología, ingeniería... Además, el desarrollo de todo tipo de técnicas y métodos para la mejora de las intervenciones sigue estando hoy en día en auge.

Por otro lado, la visión artificial o visión por computador que comenzó a desarrollarse hace algunas décadas ha visto un avance muy grande en los últimos años gracias al aumento de la potencia de cálculo y disminución del precio que han sufrido los ordenadores. En la actualidad, es posible implantar un sistema de visión en un tiempo relativamente bajo y con unos costes inalcanzables hace algunos años.

Gracias a este desarrollo, la visión artificial ha podido incursionar en el mundo de la medicina en una gran cantidad de aplicaciones, facilitando y solventando algunos de los problemas que existían. Debido a esto, la visión artificial se presenta como uno de los grandes avances dentro del mundo de la medicina y en concreto, en el ámbito de la cirugía.

La cirugía laparoscópica asistida con la mano (HALS) es un tipo de intervención en la cual el cirujano introduce su mano en la cavidad abdominal del paciente para poder llevar a cabo la operación. No obstante, en muchas ocasiones ésta necesita ser asistida por otras herramientas ya que únicamente

dispone de una mano. En esta situación, la visión artificial se presenta como un claro candidato a solventar esta necesidad, ya que, gracias a ella, un robot quirúrgico puede determinar tanto la ubicación como la posición de la mano dentro del abdomen del paciente y poder así no solo guiar al robot sino también tomar decisiones autónomamente para ayudar al cirujano en su tarea.

1.2. Objetivos

El objetivo principal del proyecto es poder determinar con la mayor precisión y robustez posible la ubicación y posición de la mano intracavitaria durante una intervención de cirugía laparoscópica asistida por la mano. La información obtenida será enviada a un robot para que tome las decisiones oportunas y así asistir al cirujano.

Para que el robot pueda reconstruir y así determinar la posición de la mano es necesario que se le indiquen tantos parámetros de la misma como sea posible, teniendo en cuenta que cuanta más información se le indique mejor será dicha reconstrucción, pero mayor tiempo y coste computacional supondrá obtenerla. Debido a esto, se ha buscado un compromiso entre estos factores y se ha determinado que los parámetros que se le indicarán al robot serán:

- Orientación respecto del eje de coordenadas de los dedos pulgar, índice y corazón.
- Posición en coordenadas XYZ de los extremos de los dedos pulgar, índice y corazón.

Además, para lograr alcanzar el objetivo final se han ido marcando algunos objetivos intermedios:

- Realización de un estudio de viabilidad en el funcionamiento conjunto de las librarías de procesamiento de imagen OpenCV y de procesamiento de nubes de puntos PCL.
- Aprendizaje del funcionamiento de PCL y sus funciones.
- Conversión de información de OpenCV a PCL.
- Estudio del ruido presente en las imágenes y posibles soluciones para eliminarlo.
- Métodos para la detección de los dedos.
- Optimización del código y análisis del rendimiento del mismo.

2. Introducción a la cirugía laparoscópica

El término laparoscopia proviene del griego laparo- (flanco) y -skopia (instrumento de observación) y se utiliza para describir el procedimiento mediante el cual se examina el peritoneo con un endoscopio. La Real Academia de la Lengua Española define laparoscopia como "una técnica de exploración visual que permite observar la cavidad pélvica-abdominal con un instrumento conocido como laparoscopio".

La cirugía laparoscópica es una técnica especializada para la realización de cirugías. Antes de su completo desarrollo, esta técnica se utilizaba generalmente en cirugías ginecológicas o de vesícula biliar. En la última década, el uso de esta técnica se ha ampliado a otros ámbitos como la cirugía intestinal. En la cirugía tradicional o "abierta", el cirujano realiza una sola incisión para acceder al abdomen, en cambio, la cirugía laparoscópica recurre a varias incisiones de 0.5 a 1 cm. Cada una de estas incisiones se denomina "puerto" y en ellos se insertan instrumentos tubulares conocidos como "trócares". Durante las intervenciones, a través de los trocares se pasan instrumentos especializados y una cámara llamada laparoscopio. Al iniciar el procedimiento, el abdomen se llena a una determinada presión con dióxido de carbono para conseguir un espacio de trabajo y visibilidad. El laparoscopio captura imágenes de la cavidad abdominal a las pantallas de alta resolución del quirófano. El sistema permite que el cirujano pueda realizar las mismas operaciones que la cirugía abierta, pero con incisiones más pequeñas.

En ciertas situaciones, el cirujano puede recurrir al uso un tipo de puerto especial que es lo suficientemente grande como para insertar una mano. Cuando se utiliza un puerto de este tipo la técnica quirúrgica se llama laparoscopia "asistida con la mano". El puerto necesario para realizar esta operación es más grande que las demás incisiones de laparoscopia, pero resulta más pequeña que la incisión para una cirugía abierta.

Las publicaciones y estudios realizados en las dos últimas décadas indican que la cirugía laparoscópica constituye uno de los mayores avances tecnológicos de finales del siglo XX y comienzos del XXI. La aplicación de la cirugía de puerto único,

la cirugía robótica y la cirugía por orificios naturales, son la expresión del gran potencial que tienen las intervenciones de mínima invasión.

2.1. Historia de la cirugía laparoscópica

El ser humano siempre ha mostrado curiosidad por conocer los órganos del cuerpo. Este hecho se puede comprobar desde la Antigua Grecia, donde Hipócrates (460-370 a.C) desarrolló espéculos para poder realizar exploraciones anales. Otros instrumentos parecidos dedicados a propósitos similares se han encontrado en diferentes lugares y civilizaciones como por ejemplo en Pompeya (70 d.C)

Ilustración 1 Espéculo romano

El siguiente paso lo dio Abulcasis, cirujano de la medicina árabe, que recurrió al principio de la refracción para poder iluminar la vagina y poder así estudiar el cuello uterino, [2].

Pero no fue hasta 1805 en el que Philipp Bizinni (1773-1809) presentó el "lichtleiter", una herramienta que conducía la luz hacia el interior del cuerpo y conseguía obtener imágenes mediante el uso de espejos y lentes. Años más tarde, en 1853, Antonie Jean Desormaux (1815-1870), perfeccionó dicho instrumento para que pudiera ser utilizado en humanos y no solo en animales. Con él consiguió estudiar la vejiga, el útero y el cuello uterino. Además, este instrumento vio potenciadas sus posibilidades cuando en 1880, el descubrimiento de la bombilla permitió generar la luz en la punta del endoscopio de manera más controlada y evitando las quemaduras que provocaba la lámpara de queroseno que se utilizaba

hasta el momento. En este aspecto se puede ver la importante contribución necesaria de diferentes áreas del conocimiento además de la medicina para el desarrollo de este tipo de técnicas.

Durante las primeras décadas del siglo XX numerosos avances fueron realizados en el ámbito de las inspecciones abdominales y torácicas, hasta que en 1929 Heinz Kalk consiguió desarrollar un sistema de lentes de hasta 135º de visión que empleaba para el diagnóstico de enfermedades del hígado y vesícula biliar. Para ello utilizaba 2 trocares, uno para realizar pequeñas punciones y otro para el tubo laparoscópico.

Pero fue en la década de los 60 el momento de mayor desarrollo en la cirugía laparoscópica. El ginecólogo e ingeniero Kurt Semm realizó gran cantidad de inventos dedicados a la cirugía laparoscópica, [3]. Desarrollo el insuflador automático que media la presión del gas intraabdominal, consiguió montar la fuente de luz de manera externa evitando las quemaduras, inventó el cable de fibra óptica que se emplea actualmente e ideó el sistema de lavado de cavidades, así como todo tipo de instrumental laparoscópico. Además, en 1982 consiguió realizar por primera vez en la historia la intervención de apendicectomía mediante laparoscopia.

En 1986 se introduce por primera vez una cámara en la cavidad que permitía observar y ayudar de manera más cómoda ya que hasta entonces el único que podía ver la cavidad era el cirujano, impidiendo a los ayudantes realizar su función. De esta manera se extendió el uso de cámara y pantallas cada vez de resolución y calidad mayores haciendo la cirugía más segura y sencilla.

2.2. Cirugía laparoscópica asistida con la mano (CLAM)

Las técnicas desarrolladas para las intervenciones de laparoscopia presentan ciertas limitaciones, algunas de ellas son:

- La capacidad de manipulación del espécimen enfermo está restringida.
- La ausencia retroalimentación táctil.
- Utilizando videolaparoscopia puede no ser posible acceder a visualizar por completo el campo quirúrgico.
- Las intervenciones suelen ser de larga duración.
- La preocupación por la seguridad de los pacientes.

Estas restricciones de la cirugía laparoscópica estándar fomentaron el desarrollo de la cirugía laparoscópica asistida (LAS) para el tracto gastrointestinal. Después del examen laparoscópico inicial y la preparación, la cirugía laparoscópica

asistida recurre a una pequeña incisión sobre la patología intestinal. El intestino involucrado se administra a través de la misma y se realiza la escisión del fragmento enfermo. La unión del intestino restante se realiza de forma extracorpórea con técnicas estándar y visión estereoscópica normal. Una de las principales desventajas de este método es que el neumoperitoneo[1] se pierde al realizar la incisión para llevar a cabo la resección. Después de la reanastomosis, se debe cerrar la incisión y restablecer el neumoperitoneo para completar la operación.

La cirugía laparoscópica asistida por la mano difiere en algunos aspectos respecto a la técnica LAS. Con HALS (hand-assisted laparoscopy surgery), se utiliza un aparato para mantener el neumoperitoneo cuando se introduce la mano en el abdomen del paciente a través de una pequeña incisión. Al igual que en la cirugía laparoscópica tradicional, el cirujano recurre a un laparoscopio y un monitor para visualizar la cavidad, pero además tiene la ventaja de usar su propia mano, teniendo así una retroalimentación táctil.

El brazo posee 7 grados de libertad que proporcionan un posicionamiento completo de la mano en el espacio. Es importante tener en cuenta que el pulgar y el dedo índice de la mano intracavitaria pueden ser utilizados para asegurar la hemostasia en el caso de una hemorragia intraoperatoria. Debido a que el método HALS permite el mantenimiento de la sensación táctil y utiliza la coordinación ojo-mano, esta variación de la cirugía laparoscópica resulta más sencilla de dominar para los cirujanos que fueron entrenados para la cirugía tradicional abierta.

Recapitulando, las técnicas HALS tienen el potencial de:

- Facilitar la cirugía laparoscópica.
- Acortar los tiempos de las intervenciones.
- Reducir los tiempos y la dificultad de aprendizaje.
- Mejorar la seguridad de los pacientes.
- Permiten la realización de una disección.

Inicialmente la implantación de HALS se ha limitado a unos pocos centros importantes y las investigaciones sobre de la correcta implementación de este método ha sido escasa. Sin embargo, existe la necesidad de realizar investigaciones sobre las cuestiones más básicas de las intervenciones laparoscópicas asistidas manualmente y establecer métodos de entrenamiento para evitar el desarrollo errático que tuvo la cirugía laparoscópica en sus comienzos. Es necesario el desarrollo de nuevos instrumentos y mejora en el diseño de estas técnicas.

[1] Neumoperitoneo: En medicina, presencia de aire en la cavidad peritoneal.

Algunos de los problemas existentes pueden ser aparentemente simples, como por ejemplo la altura apropiada de la mesa de operaciones y su orientación. Estos parámetros necesitan ser estudiados porque la configuración de un quirófano es diferente para una intervención HALS que para la cirugía laparoscópica estándar o para una cirugía abierta tradicional. Otro ejemplo es el problema derivado del hecho de que el cirujano tenga una mano fuera del abdomen y otra dentro, esta puede ser una posición poco cómoda y podría fomentar la fatiga del cirujano durante procedimientos largos y complejos. También necesita ser analizado si lo óptimo es que la mano de ayuda en determinadas tareas sea la otra mano del cirujano operante o la de un cirujano asistente.

Según apuntan los primeros informes, la ubicación de la mano auxiliar debe ser tomada como un puerto de la operación y triangulado con el otro puerto de la intervención laparoscópica de manera que ambos formen ángulos de azimut semejantes con el punto de toma de imágenes laparoscópicas. La ubicación de dichos puertos de la operación en ángulos iguales permite al cirujano dirigirse de la manera más cómoda al órgano objetivo. Sin embargo, si la mano auxiliar está demasiado cerca de dicho órgano, puede entorpecer la visión de éste y las técnicas de la operación. Si la mano auxiliar está demasiado lejos del órgano, la fatiga podría convertirse en un factor importante a tener en cuenta en la realización del procedimiento. Además, es resulta necesario analizar los riesgos de causar daños a otras regiones u órganos durante la intervención.

Una visión preliminar de HALS sugiere la necesidad de investigar los siguientes aspectos:

- Dispositivos o mecanismos para facilitar la introducción y retirada de la mano intracavitaria y herramientas en el abdomen.
- Creación de métodos o herramientas para poder controlar las hemorragias más fácilmente.
- Mejora del alcance efectivo de la mano operativa.
- Desarrollo de instrumentos manuales activados mediante los dedos.
- Dispositivos para permitir el cambio de mano intracavitaria de izquierda a derecha, y viceversa, de forma rápida y segura.
- Diseño de un sellado cómodo entorno a la mano operativa de manera que se minimice la fatiga muscular y entumecimiento de la misma durante las intervenciones.

Debido a que la tecnología laparoscópica asistida a mano se encuentra en una etapa temprana, resulta urgente el desarrollo de estudios adecuados para poder explotar su potencial. Teniendo en cuenta, además, que sólo unos pocos instrumentos operativos estándar pueden ser utilizados en

procedimientos de cirugía HALS, es necesario el diseño y fabricación de nueva instrumentación para estas técnicas. Es fundamental el desarrollo de un ambiente ergonómico para la práctica y ejecución de las intervenciones. Y, por último, se requiere de nuevos mecanismos para fomentar la educación quirúrgica y extender conocimiento de las posibilidades de la cirugía laparoscópica asistida manualmente.

3. Introducción de la imagen digital

3.1. Historia y desarrollo

Comenzamos este capítulo haciendo una breve presentación de la historia que se ha seguido para el desarrollo de la imagen hasta llegar a nuestros días.

Desde nuestros comienzos, la humanidad siempre ha sentido la necesidad de representar objetos, escenas o eventos mediante imágenes. Las primeras muestras de ello son las pinturas rupestres que se diferentes asentamientos distribuidos por prácticamente todo el mundo.

Quizás las más famosas sean las de la cueva de Altamira, España debido a su buena calidad y conservación, Ilustración 2. En ellas se aprecian las representaciones de animales como el bisonte o el ciervo. Además, se puede ver el gran uso de la perspectiva, el juego del volumen utilizando la forma de la cueva y de los colores.

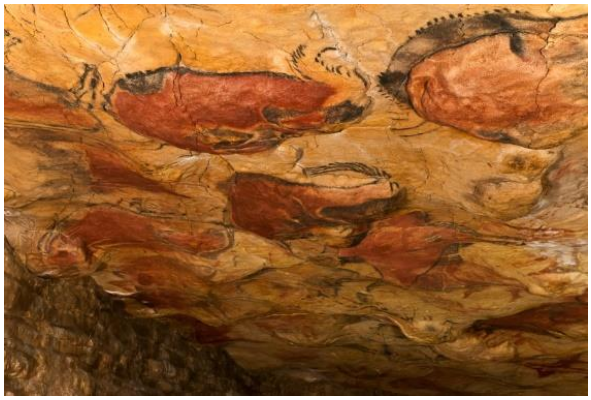

Ilustración 2 Cueva de Altamira

Con el desarrollo de la civilización, las técnicas de la pintura fueron mejorándose y poco a poco fueron apareciendo los antecedentes de las cámaras tradicionales. Un ejemplo de esto es la cámara oscura. Ilustración 3

La cámara oscura se trata de una caja o sala totalmente cerrada y opaca con un agujero en uno de sus laterales. Por dicho agujero entra la iluminación del exterior de tal manera que en la pared opuesta se representa la escena del exterior invertida. Pudiendo fijar esta escena mediante pintura. Su origen no está del todo claro, pero

se sabe que Aristóteles en el siglo IV a.C. hizo una descripción de la misma en uno de sus escritos. Con el tiempo se fue perfeccionando utilizando lentes de manera que ya no era necesaria una iluminación específica en el exterior ya que ésta era capaz de captar más luz que un simple agujero.

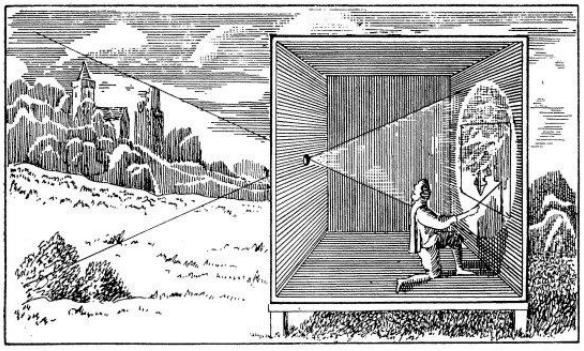

Ilustración 3 Cámara oscura

Pero no fue hasta comienzos del siglo XIX en el que el químico Joseph Nicéphore Niépce realizó la primera fotografía usando una cámara oscura y una lámina de material fotosensible. Con el tiempo se fueron mejorando las tecnologías para la toma de imágenes dando paso a la fotografía tradicional.

Y finalmente, en 1955 el científico Russell Kirsch consiguió crear la primera imagen digital de la historia Ilustración 4. Ésta tenía un tamaño de 176x176 pixeles y se obtuvo mediante un dispositivo que convertía las imágenes en matrices con valor 0 o 1. En la imagen se puede ver la bajísima resolución con la que fue tomada y los problemas derivados de ello como son la aparición de barras verticales. El problema principal al que se enfrentó Russell fue la baja capacidad de cálculo de los ordenadores de la época y su reducida memoria.

Ilustración 4 Primera imagen

Y 22 años después de esta imagen, en 1977 aparece la primera cámara de fotos digital la cual permitía la obtención directa de imágenes digitales revolucionando así la industria de la fotografía.

3.2. Fundamentos

En este documento no realizaremos un estudio detallado de la imagen ya que ese no es el propósito del mismo, pero sí que es conveniente explicar algunos fundamentos ya que serán clave para poder comprender correctamente futuros conceptos y poder así solventar algunos de los problemas que nos encontraremos.

Una imagen digital es una colección ordenada de valores que se representan ordenados en forma de filas y columnas formando una matriz. Este tipo de imágenes

pueden ser tratadas por un ordenador debido a que se forma por valores cuantificables y concretos.

3.3. Obtención

Para la obtención de imágenes digitales hoy en día existen diversos métodos como pueden ser los escáneres, dibujos vectoriales mediante ratón o tableta, etc.... Sin embargo, nos fijaremos únicamente en la obtención mediante cámara digital ya que es la que se usará posteriormente y es la que nos conviene entender en profundidad.

La obtención en una cámara digital se divide en 3 zonas: Óptica, Sensor y Digitalización.

3.3.1. Óptica

La óptica hace referencia a los mecanismos que tiene la cámara para captar la luz y conducirla de forma correcta al sensor.

Tal y como vimos en la historia de la imagen las primeras cámaras oscuras tenían únicamente un agujero por el que la luz entraba. Esto hacía que la luz que se reflejaba en la pared opuesta era muy tenue y por lo tanto la luminosidad exterior debía ser alta. Este problema se podía solventar haciendo un agujero más grande, pero tenía el inconveniente de que a medida que el agujero era mayor las imágenes proyectadas sobre la pared se hacían más borrosas.

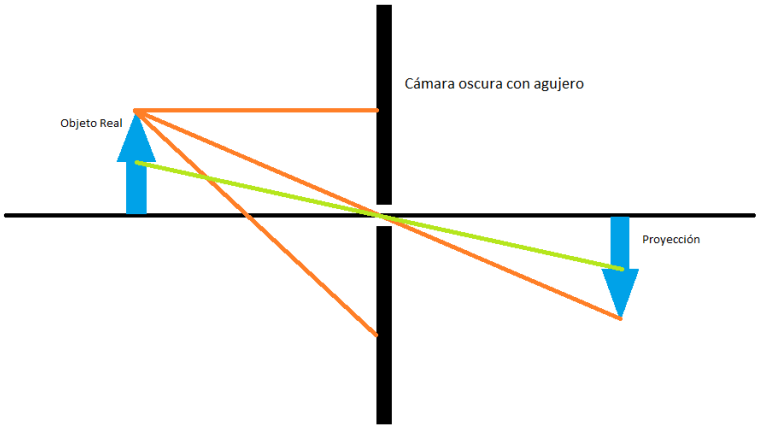

Ilustración 5 Esquema cámara oscura

Este problema se solventó con la aparición de las lentes, las cuales eran capaces de captar mucha más luz sin difuminar el resultado. Esto se puede entender fácilmente comparando la Ilustración 5 con la Ilustración 6. En la cámara oscura única luz que forma parte de la proyección es aquella que consigue pasar por el

pequeño agujero, mientras que, utilizando una lente, toda la luz que pasa por la lente contribuye a la generación de la proyección. El único problema que encontramos cuando usamos lentes es que éstas obligan a colocar la superficie de proyección a una distancia determinada ya que si ésta no está correctamente colocada los diferentes haces de luz no coincidirán en el mismo punto produciendo así una imagen borrosa y poco nítida. Esta distancia viene dada por el punto focal.

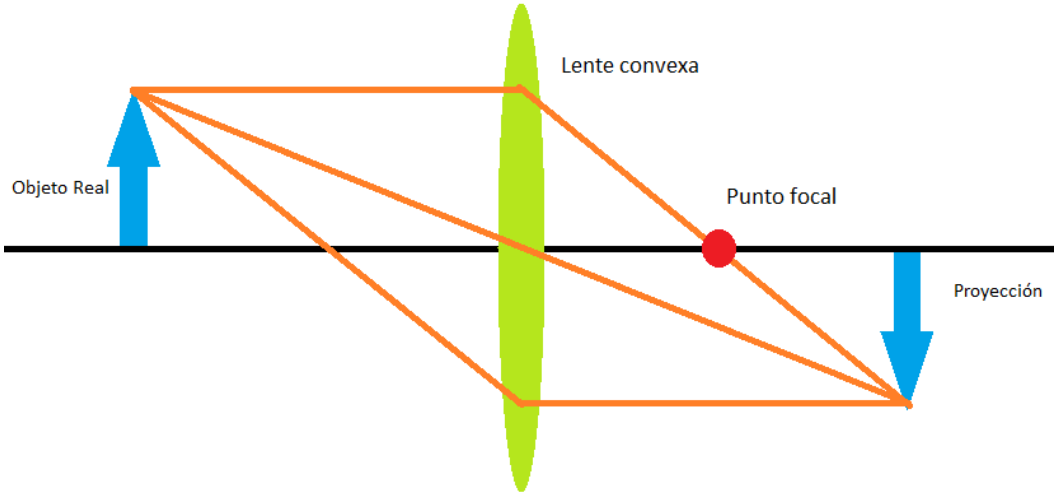

Ilustración 6 Esquema de lente convexa

La correcta elección de la focal de la lente es fundamental a la hora de usar la cámara ya que determinará el tamaño con el cual se vean los objetos en nuestras imágenes. Siendo mayores cuanto mayor sea la distancia focal.

Además de las lentes, las cámaras disponen de otras herramientas para controlar la entrada de la luz hacia el sensor, uno muy importante es el diafragma, éste consigue regular la cantidad de luz que entra hacia la lente, puedo así limitar dicha luz cuando exista una luminosidad muy alta o permitir su paso cuando sea baja.

También existen los filtros los cuales nos ayudan a limitar el espectro de luz que entra a la cámara, por ejemplo, poniendo un filtro de rojos podemos representar en una imagen únicamente los objetos que contengan rojo.

3.3.2. Sensor

El sensor es el encargado de transformar la luminosidad recibida en una señal eléctrica que pueda ser interpretada y digitalizada. El tipo, y sus características determinarán en gran medida el resultado final de la fotografía.

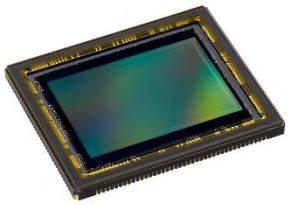

Ilustración 7 Sensor

Constructivamente un sensor consiste en chip en la que en una de sus caras se disponen pequeños componentes sensibles a la luz (pixeles) que serán los encargados de recibirla y transformarla en impulsos eléctricos.

El tamaño de los sensores puede variar de unos 10mm hasta 40mm, esto es determinante en el resultado de la imagen obtenida ya que, a mayor tamaño, mayor número de pixeles dispondrá el mismo y por tanto más información seremos capaces de captar.

El método empleado para ir captando los impulsos enviados por los píxeles es definido por la tecnología que implemente el sensor, existen muchos tipos, pero destacan los sensores CCD y en gran medida los CMOS que son los más utilizados actualmente debido a su rendimiento, bajo consumo y bajo coste de producción.

Otro parámetro que determina a un sensor es la forma del mismo, lo común es que sean rectangulares como el de la Ilustración 7 pero podemos encontrar sensores lineales para determinadas aplicaciones industriales.

Finalmente, un aspecto fundamental es si un sensor es sensible al color o no. Para conseguir que un sensor capte los colores se necesario que tenga píxeles que sean sensibles a diferentes colores, normalmente al rojo, al verde o al azul. Formando ternas de estos 3 tipos de pixeles se podrá obtener 3 imágenes diferentes, cada una correspondiente a un color para después juntarlas y formar una imagen con toda la gama cromática.

3.3.3. Digitalización

Dado que en el mundo real todo es continuo, los valores que puede tener la intensidad de luz en un punto pueden ser infinitos, por lo tanto, para obtener una imagen digital deberemos realizar antes un proceso llamado digitalización. La digitalización de una imagen consiste analizar el impulso eléctrico que proviene de los píxeles del sensor y otorgarle un valor concreto, de tal manera que repitiendo el proceso para todos los puntos del sensor obtendremos la matriz de datos perteneciente a la imagen digital de la escena Ilustración 8.

Existen diversas maneras de realizar la digitalización, un parámetro muy importante es el que determina el rango en el que pueden estar los valores otorgados, es decir, una de las formas más típicas de imágenes digitales son las que representan la luminosidad de un pixel con un valor que puede estar entre 0 y 255 representando el 0 el negro y el 255 el blanco. Todos los valores intermedios representarán valores de gris más o menos oscuros en función de si son más cercanos al 0 o al 255. El resultado de este tipo de representación es una imagen con 256 tonos diferentes de grises.

173	177	181	185	189	193	197	201
169	85	89	93	97	101	105	205
165	81	29	33	37	41	109	209
161	77	25	5	9	45	113	213
157	73	21	17	13	49	117	217
153	69	65	61	57	53	121	221
149	145	141	137	133	129	125	225
255	253	249	245	241	237	233	229

Ilustración 8 Representación imagen digital

3.3.4. Ruido en la imagen

Durante todo el proceso de obtención de las imágenes, existe la posibilidad de que se produzcan errores o que algún evento externo altere el resultado final. Estos errores e interferencias se manifiestan en la imagen como ruido y éste nos hará perder información de la imagen y hacer que pierda calidad.

Entender los tipos de ruido que se pueden presentar y las diferentes causas de los mismos es muy importante a la hora de poner en funcionamiento un sistema que haga uso de la captura de imágenes.

Distinguimos 4 tipos diferentes:

Ilustración 9 Ruido Gaussiano

- **Ruido gaussiano**: Se presenta en forma de pequeñas variaciones en los colores y las intensidades en todos los puntos de la imagen de manera aleatoria. Viene relacionado por la radiación electromagnética y resulta imposible eliminarlo por completo, aunque si se puede minimizar en gran medida su efecto. Sigue una distribución normal y por ello es bastante fácil reducirlo mediante un filtro gaussiano con tratamiento de imágenes.

- **Ruido uniforme**: Es bastante parecido al anterior ya que también es aditivo y visualmente se pueden confundir. Pero este tipo de ruido en lugar de seguir una distribución normal sigue una distribución uniforme. El origen de este ruido está en el convertidor analógico/digital de las cámaras.
- **Ruido periódico**: Se producen patrones periódicos en la imagen. Se produce debido a fuertes interferencias con máquinas eléctricas de gran potencia. Lo más recomendable es el apantallamiento de la cámara en la medida de lo posible.


Ilustración 10 Ruido periódico

- **Ruido sal y pimienta**: Este tipo de ruido es muy característico y se distingue muy fácilmente a simple vista. Consiste en la aparición del pixeles blancos y negros en puntos aleatorios de la imagen. Se produce por fallos en la transmisión de la imagen y es posible eliminarlo en gran medida mediante un filtro de la mediana.


Ilustración 11 Ruido sal y pimienta

3.4. Características

Existe una multitud de tipos de cámaras con diferentes sensores y ópticas para cada aplicación, ya sea industrial o no y una buena decisión a la hora de escoger una u otra puede resultar determinante para un correcto funcionamiento, por lo tanto, es fundamentar hacer un estudio previo de las características de nuestro entorno, necesidades de la aplicación y de la oferta de cámaras que tenemos a nuestra disposición.

Ahora que conocemos las bases de las imágenes digitales vamos a analizar algunas de sus características, algunas de ellas ya se han mencionado anteriormente, pero se detallarán un poco más.

- **Resolución:** Sirve para indicar el nivel de detalle o nitidez con el que se representa la imagen. Viene dada normalmente como el número de píxeles en una pulgada (2.54cm). Por lo tanto, a mayor resolución mayor será el detalle de la imagen y más información tendrá la misma, Ilustración 12.

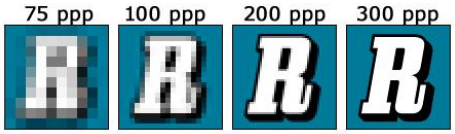

Ilustración 12 Ejemplo de resolución

- **Tamaño:** Es el número de píxeles de los que está compuesta una imagen.

- **Formato de la imagen:** Este parámetro hace referencia al método mediante el cual el sistema almacena la información de la imagen. Se distinguen dos clases, por un lado, están los métodos sin pérdida de información y por el otro lo que implican pérdida de información. Actualmente el más extendido es el formato JPEG el cual es corresponde al tipo de pérdida de información, pero consigue ahorrar mucha memoria simplificando regiones de las imágenes que tienen zonas de colores similares. Además, admite la carga progresiva haciéndolo ideal para páginas web.
- **Relación de aspecto:** Se trata de la razón entre el ancho y el alto de la imagen, este valor indicará si la imagen es cuadrada o más o menos alargada.

3.5. Iluminación

La iluminación es un factor determinante en la obtención de imágenes. Corresponde a la fuente de luz que será captada más tarde por nuestro sensor, por ello, disponer de una iluminación adecuada es muy importante.

Antes de continuar cabe mencionar que existe la posibilidad de solucionar algunos problemas de iluminación modificando parámetros de la óptica tales como la apertura del diafragma o el tiempo de exposición, pero el rango de acción de estos recursos es limitado y no siempre se podrá obtener un resultado óptimo.

Los tipos de iluminación son:

- **Direccional:** La luz es proyectada directamente hacia el objeto. Es utilizada para localización y reconocimiento de objetos. Es sencilla y muy útil en muchas aplicaciones, pero tiene el gran inconveniente de que puede generar reflejos y sombras en la imagen.
- **Difusa:** Iluminamos al objeto desde todos los ángulos para evitar así la generación de sombras.
- **A contra luz:** La fuente de luz se coloca detrás del objeto en dirección a la cámara, de este modo solo apreciaremos el contorno de la pieza sobre un fondo blanco, pero con una gran nitidez y detalle. Se utiliza para detección y conteo de piezas.
- **Estructurada:** Se proyectan puntos o un patrón de colores sobre un objeto o escena para después poder analizar su desviación y estimar así la profundidad. Este tipo de iluminación es utilizada por ejemplo por la cámara 3D Kinect de Microsoft.

Ilustración 13 Iluminación estructurada

Además, es tipo de fuente de luz determina las características de la luz recibida:

- **Lámparas incandescentes:** En desuso actualmente, se basan en calentar un filamento de Tungsteno hasta la incandescencia. Producen una luz intensa y focalizada, pero son poco eficientes y se degradan con el tiempo.
- **Tubos fluorescentes:** Producen una luz difusa, pero se degradan con el tiempo. Tienen, además el inconveniente de que parpadean según su frecuencia de alimentación, esto puede generar ruido en las imágenes según sea la frecuencia de captura y tiempo de exposición.
- **Luz LED:** Tienen una gran duración y rendimiento, se pueden agrupar muchos LED en zonas reducidas. La potencia es limitada, aunque está en desarrollo.
- **Lámparas Xenon:** Producen una iluminación blanca muy intensa. Se les puede incorporar un estroboscópico para aplicaciones de alta frecuencia.
- **Láser:** Proyectan un haz de luz concentrada, son utilizados para la luz estructurada.

3.6. Visión artificial

La visón artificial o también llamada visión por computador se trata de una disciplina científica que se conforma de métodos de obtención, de procesamiento, de análisis y comprensión de imágenes. Su objetivo es la producción de información para que ésta pueda ser tratada por una máquina. Todos estos métodos hacen uso de distintos campos como son la geometría, la estadística o la física entre otros.

La visión artificial es ampliamente utilizada en una gran cantidad de sectores como pueden ser el sector industrial, el sector de la investigación o el sector del entretenimiento. Para el caso de la visión artificial en el sector de la industria, ésta es empleada para la inspección automatizada y la orientación y guía de robots en plantas y procesos industriales haciendo uso de una combinación de métodos de análisis automático de imágenes.

Algunas de las aplicaciones más extendidas de la visión artificial pueden ser:

- Control de procesos
- Análisis de imágenes médicas o de modelado topográfico
- Lectura de matrículas o códigos de barras
- Guiado de vehículos autónomos
- Vigilancia mediante el conteo de personas
- Reconocimientos faciales
- Interacción entre hombre-máquina

Uno de los ámbitos más importantes de los que se han comentado y que, además, coincide con el tema de este documento es la visión artificial en aplicaciones médicas. Esta área se caracteriza por la extracción de información a partir de datos de imágenes con el objetivo de realizar o ayudar a realizar un diagnóstico médico. Los tipos de imágenes con los que se trabaja en este ámbito van desde imágenes de rayos X, a imágenes ultrasónicas. Algunas de las aplicaciones son la detección de tumores, ayuda al cirujano durante intervenciones, mediciones del flujo sanguíneo o análisis de la temperatura corporal.

La visión artificial hace uso tanto de imágenes en dos dimensiones como de imágenes tridimensionales, el principal motivo para utilizar un tipo de imagen u otro es si para lograr el objetivo de la aplicación es necesario conocer la posición real de los objetos, es decir, la visión en 2 dimensiones se limitará a trabajar con la información obtenida a partir de formas, colores o texturas de una imagen normal mientras que la visión 3D se centrará en las profundidades de los objetos, formas y posiciones de los mismos. Es importante entender que la visión en dos dimensiones es mucho más sencilla y rápida, por lo tanto, siempre se recurrirá a ella a no ser que para la aplicación en concreto no sea válida.

4. Imágenes 3D

4.1. Introducción

En el apartado anterior se ha hecho un estudio de las imágenes en 2 dimensiones, en este se procederá a hacer lo mismos, pero con las imágenes en 3 dimensiones.

Una imagen en 3 dimensiones se caracteriza porque es capaz de almacenar la información de las 3 coordenadas espaciales de cada punto o pixel. Esta capacidad hace que su potencial en cuanto a posibles aplicaciones en visión artificial sea muy alto. En contra, tienen el problema de que la obtención de las mismas requiere tecnologías más complejas y su tratamiento máquinas más potentes y con mayor almacenamiento. Además, su representación es más complicada que en las imágenes tradicionales ya que todos los métodos de representación de imágenes convencionales están diseñados para trabajar únicamente con 2 dimensiones como, por ejemplo, la pantalla del ordenador o el papel de una fotografía. Debido a esto, las imágenes en 3 dimensiones son principalmente utilizadas en aplicaciones de visión artificial, aunque desde hace unos años están expandiéndose a otros campos como el cine o la realizad virtual.

La visión en tres dimensiones es utilizada, además, por los humanos. Es muy común en la sociedad el pensar que somos capaces de ver en tres dimensiones gracias a que disponemos de 2 ojos y esto, es una verdad a medias. Tal y como se va a ver más adelante en los métodos de obtención, uno de ellos es el estéreo, el cual consiste en la obtención de 2 imágenes en 2 dimensiones desde 2 puntos ligeramente desplazados. Sin embargo, tanto los seres humanos hacemos uso de una gran cantidad de recursos a parte del estéreo para obtener la tercera dimensión.

Algunos de estos mecanismos son la oclusión de objetos, paralaje de movimiento, proyecciones de sombras, desenfoques…

- **Oclusión de objetos**: Resulta ser uno de los más importantes en el día a día, consiste en que si un objeto “A” tapa a otro objeto “B”, esto significa que “A” está más cerca de nosotros que “B”.
- **Paralaje de movimiento**: Este también es muy importante, consiste en que los objetos más alejados tenderán a parecer que se desplazan más despacio que los objetos más cercanos. Este hecho puede ser fácilmente apreciable cuando se va montado en un coche o tren, todos los objetos que estén cerca de vehículo aparecerán y desaparecerán rápidamente del campo de visión, mientras que los objetos más alejados como las montañas o el sol parecerán no moverse o hacerlo realmente despacio.

- **Proyecciones de sombras**: La iluminación juega un papel fundamental en la toma de imágenes en general y en la visión humana en particular. Las sombras arrojadas por objetos sobre el fondo, sobre otros objetos o sobre sí mismos nos ayudan a determinar no solo las posiciones relativas entre objetos si no también la forma de dichos objetos.
- **Desenfoques**: Tal y como se vio anteriormente, la profundidad de campo y en consecuencia las zonas enfocadas y desenfocadas es algo inevitable en la toma de imágenes. Siendo conscientes de este problema, es posible obtener información adicional de ello. Así, sabemos que aquellos objetos que se encuentren desenfocados estarán en un plano de profundidad diferente al de aquellos que aparecen enfocados.
- **Tamaños relativos**: Para nuestra visión, el cerebro no solo analiza las imágenes captadas por nuestros ojos, sino que también recurre a la memoria. Cuando vemos algo conocido, como puede ser una persona, o un coche automáticamente nuestro cerebro asume un tamaño estimando calculando aproximadamente la distancia a la cual se encuentra para que se vea de ese tamaño. Este mecanismo resulta muy útil en determinados entornos y situaciones, pero no siempre es efectivo, pudiendo dar lugar a errores de apreciación en las distancias.

La agrupación de todos estos mecanismos junto con el de estéreo y algunos otros conforman la visión humana, sin embargo, no forman un sistema perfecto, ya que existen algunas situaciones en las que dichos mecanismos se contradicen, dando lugar a percepciones erróneas. Estas situaciones raramente suceden en la vida real se pueden modificar fácilmente algunas imágenes para producir estos errores, estos son comúnmente denominados “efectos visuales”.

Es muy importante entender que, aunque para nosotros recurrir a todos estos mecanismos no nos cuesta ningún esfuerzo, y, de hecho, lo hacemos inconscientemente, la complejidad de dichos mecanismos y a dificultad en implementarlos todos correctamente en enorme. Además, es importante destacar que todos estos métodos proporcionan resultados relativos, es decir, que nunca se da un valor concreto para la profundidad o la distancia, sino las apreciaciones son de tipo: un objeto delante de otro, o, un objeto está muy lejos de mí…. Debido a estos factores, para una máquina resulta prácticamente imposible implementar estos mecanismos ya que por un lado son muy costosos y por el otro no proporcionan resultados concretos y cuantificables. Por eso, para la visión artificial no basta con una sola cámara u ojo como podría ser nuestro caso, sino que siempre necesitará implementar uno de los métodos de obtención que se comentarán más adelante.

La forma normal en la que se encuentran las imágenes 3D es la nube de puntos. Una nube de puntos se caracteriza por ser una agrupación de puntos, en la que cada uno posee sus propias coordenadas XYZ que lo ubican en el espacio, pero la principal diferencia con una imagen normal es que los puntos no tienen porqué cubrir toda el área o espacio disponibles. Es decir, en una imagen normal, si el

tamaño de la misma es de 1000x1000 píxeles, se asegura que todos y cada uno de los pixeles dentro de esa matriz van a tener un valor de color. Sin embargo, las nubes de puntos no garantizan esto, dependiendo de las tecnologías de obtención, que se verán más adelante, algunas sí que proporcionan nubes completas en las que se dan valor a todos los puntos dentro de un rango mientras que otras simplemente proporcionan puntos de una escena pudiendo dejar huecos entre ellos.

4.2. Obtención

La obtención de imágenes en tres dimensiones resulta ser bastante más complicada que la obtención de imágenes normales. En la actualidad existen diferentes métodos que se analizarán a continuación en los que cada uno tiene unas ventajas y unos inconvenientes, pero es una regla común la necesidad de realizar cálculos relativamente pesados para su obtención.

Podemos distinguir 2 tipos de tecnologías en la obtención de imágenes 3D, por un lado, están las que recurren a cámaras tradicionales y otros recursos para la poder detectar la tercera dimensión y por otro lado están aquellas que rompen con las tecnologías tradicionales y recurren a otros métodos.

4.2.1. Estéreo mediante 2 cámaras

Este es el método estaría en el grupo que hace uso de las tecnologías tradicionales ya que para llevarlo a cabo es necesario disponer de 2 cámaras iguales.

Esta tecnología es de las más extendidas y conocidas por todo el mundo debido a que fue una de las primeras y además es la tecnología que empleamos tanto humanos como gran cantidad de animales para detectar la tercera dimensión.

Su funcionamiento se basa la posibilidad de calcular la profundidad de un punto a partir de la distancia existente en su ubicación en 2 imágenes tomadas a una distancia conocida.

En la Ilustración 14 se puede ver un esquema para entender mejor el funcionamiento de esta tecnología. En el caso de arriba, solo se dispone de una cámara y ésta muestra un árbol, sin embargo, no es posible determinar la profundidad de este ya que tampoco sabemos su tamaño. En el segundo caso disponemos de una segunda cámara que ve el mismo árbol, pero desde otra posición. Superponiendo sus imágenes, ya sí que somos capaces de determinar su profundidad.

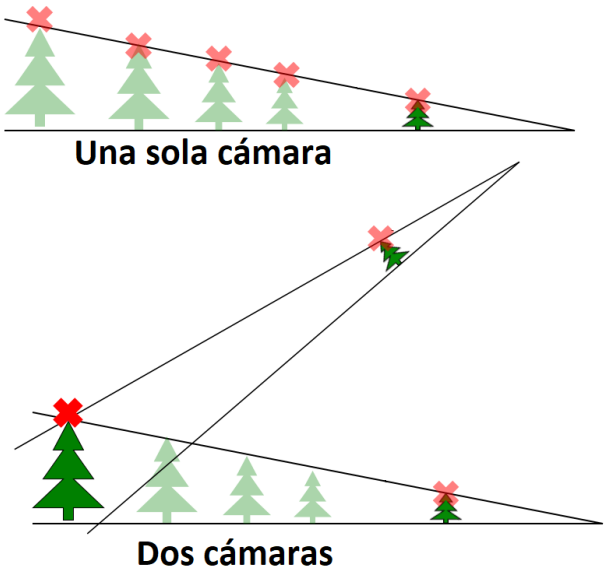

Ilustración 14 Esquema del estéreo

Para calcular las distancias a la cual se encuentran los puntos se recurre a una resolución geométrica. Debido a la restricción epipolar [4], el cálculo de las profundidades se reduce a la realización de una triangulación. Las imágenes captadas por las 2 cámaras presentan una desviación horizontal conocida como disparidad. Aplicando semejanza de triángulos se consigue obtener X_I y X_D y después se obtendrá la profundidad a partir de estos valores.

$$x_I = \frac{f}{z}\left(x + \frac{b}{2}\right)$$

$$x_D = \frac{f}{z}\left(x - \frac{b}{2}\right)$$

$$d = x_I - x_D \;\rightarrow\; z = \frac{f}{d}b$$

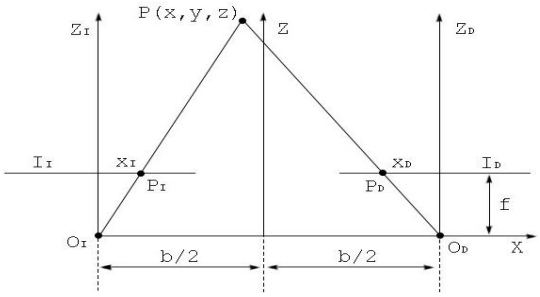

Ilustración 15 Esquema geométrico del estéreo

En las ecuaciones anteriores, “f” hace referencia a la focal de la lente, y “b” a la distancia a la cual se encuentran las cámaras.

Sin embargo, el estéreo no solo consiste en aplicar estos cálculos para cada punto ya que antes de efectuarlos en imprescindible el encontrar el mismo punto en las 2 imágenes, es decir, para poder determinar la disparidad de un punto es condición necesaria que encontremos de forma inequívoca el mismo punto en las 2 imágenes captadas. A este proceso se le denomina búsqueda de puntos homólogos o correspondencia estereoscópica y la correcta ejecución de este paso determina en gran medida la calidad de los resultados obtenidos. El encontrar los puntos homólogos es el proceso más complicado del estéreo y no siempre es posible llevarlo a cabo con éxito ya que cabe la posibilidad de que un punto se encuentre en una imagen, pero no en la otra debido a posibles oclusiones o reflejos. Además, a pesar

de utilizar cámaras iguales existe la posibilidad de que haya ligeras variaciones en los valores aportados dificultando el proceso.

Existen 2 tipos de métodos para conseguir el emparejamiento:

- **Técnicas basadas en el área**: Recurren a analizar los valores de intensidad tanto del pixel en concreto como de su vecindad para después buscar los mismos patrones en la imagen de la otra cámara. La principal ventaja de estos métodos es que hacen posible la creación de mapas de disparidad de alta densidad en la que prácticamente la totalidad de los pixeles han sido emparejados. Sin embargo, su principal debilidad es que al hacer uso de los valores de intensidad son muy sensibles a cambios en la iluminación y pueden realizar emparejamientos erróneos.
- **Técnicas basadas en las características**: Recurren a representaciones simbólicas obtenidas de la imagen de intensidad. Algunas de estas son puntos de borde aislados, cadenas de puntos de bordes o regiones delimitadas por bordes. Estos métodos son fuertes en las debilidades de los del otro tipo y débiles en las fortalezas de los otros, es decir, consiguen una gran robustez ante cambios de iluminación y son relativamente rápidos, pero consiguen mapas de disparidad pobres en densidad.

4.2.2. Luz estructurada

Como se acaba de explicar en la técnica del estéreo, con una única cámara no nos basta para obtener la profundidad, para resolver este problema el estéreo recurre a una segunda cámara, sin embargo, existe otra opción más sencilla que consiste en utilizar por ejemplo un láser en lugar de la segunda cámara, para que proyecte un haz de luz sobre la escena. De este modo, conociendo la trayectoria del haz de luz y buscando en la imagen capturada el punto de luz podremos determinar de la misma manera la profundidad de dicho punto.

En la Ilustración 16 se puede ver a grandes rasgos el funcionamiento de esta técnica. Al igual que pasaba en el estéreo es necesario dar un segundo valor para poder determinar cuál es la profundidad del punto.

Sin embargo, con este método únicamente es posible determinar la profundidad de un único punto ya que el láser solo emite un haz. Pero este problema se resuelve modificando la luz emitida.

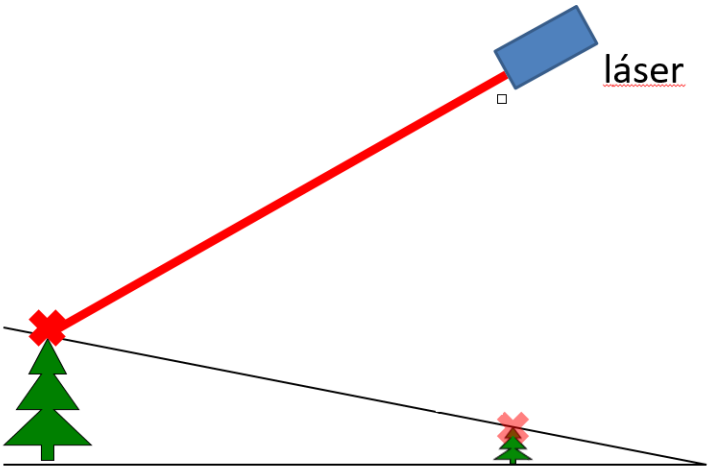

Ilustración 16 Esquema de funcionamiento de la luz estructurada

- **Haz de luz puntual**: Se trata del método más sencillo de luz estructurada. En el solo se dispone un único haz de luz que produce un punto en la escena. Este punto debe ser buscado en la imagen captada y realizar los cálculos de una manera muy parecida al del estéreo. Resulta muy útil cuando se desea conocer la posición un objeto o si la escena es totalmente estática ya que se podría ir moviendo el láser e ir captando diferentes imágenes para luego después ir reconstruyendo la escena en tres dimensiones.
- **Haz de luz lineal**: En este caso en lugar de producir un punto en la escena lo que se proyecta es una línea. Este método sirve fundamentalmente para determinar la forma de objetos ya que analizando las deformaciones de la línea que en un principio debería ser recta se puede determinar la forma del objeto sobre la que se ha proyectado la luz. Algunas aplicaciones pueden ser el control de calidad en la fabricación de piezas o la determinación del estado de las ruedas de un coche, Ilustración 17. Por último, es posible emplear varios haces de luz lineales para generar una "rejilla" y así poder obtener más información del objeto.
- **Proyección de patrones**: Se trata de la evolución del método anterior, en este caso en lugar de emplear láseres lineales se recurre a un proyector que generará diferentes patrones sobre la escena, de este modo, cuando capturemos la imagen esos patrones se habrán deformado debido a las diferentes formas y profundidades de los objetos y estas podrán ser determinadas estudiando dichas deformaciones.
- **Codificación espacial**: Este método es una mezcla entre el haz de luz puntual y la proyección de patrones y resulta muy útil y efectivo para obtener imágenes en 3 dimensiones de escenas relativamente complejas. Este método consiste en la proyección

de una gran cantidad de puntos situados de forma pseudoaleatoria, Ilustración 18. De nuevo la posición de cada punto en la imagen dependerá de los objetos que haya y de la cercanía a la cámara que tengan, de este modo, analizando la desviación de cada punto se obtiene la imagen en 3 dimensiones. Este método es implementado por una de las cámaras más extendidas y utilizadas para obtención de imágenes 3D debido a su calidad y bajo precio, esta cámara es la "Kinect" y está fabricada por Microsoft.

Ilustración 17 Inspección de neumáticos con haz de luz lineal

Ilustración 18 Generación de puntos pseudoaleatorios

Las principales características de la luz estructurada son que con ella se pueden obtener unos resultados de muy alta calidad y precisión, además, son no son dependientes de una iluminación externa ya que ya disponen de la suya propia, pudiendo llegar a ser en un espectro no visible para los humanos como es el caso de la cámara "Kinect". Sin embargo, pueden fallar cuando hay un exceso de iluminación exterior, lo que puede provocar que las proyecciones propias queden difuminadas y por lo tanto sea mucho más difícil encontrarla en la imagen.

4.2.3. Tiempo de vuelo

La última de las tecnologías que se van a analizar, las cámaras de tiempo de vuelo suponen un cambio por completo en la forma en la que se obtienen las imágenes en tres dimensiones.

Las cámaras basadas en tiempo de vuelo miden el tiempo que necesita un haz de luz en ir desde la cámara hasta la superficie de un objeto y regresar. Además, emplean luz modulada en amplitud generada a partir de matrices de diodos de luz que son capaces de emitir a una frecuencia cercana al infrarrojo, [5].

Conociendo la velocidad de la luz "c", la frecuencia de la modulación, "f_{mod}" y la correlación entre 4 señales que están desplazadas unas de otras 90º, "$r_0(0º)$", "$r_1(90º)$" "$r_2(180º)$", "$r_3(270º)$" es posible determinar tanto el retardo en fase "ϕ_i", la amplitud "a_i" y la distancia, "d_i" en cada uno de los píxeles que conforman la imagen.

$$\phi_i = artan\left(\frac{r_1 - r_3}{r_0 - r_2}\right)$$

$$a_i = \frac{\sqrt{(r_1 - r_3)^2 + (r_0 - r_2)^2}}{2}$$

$$d_i = \frac{c * \phi_i}{4\pi * f_{mod}}$$

La principal ventaja de poder calcular estos parámetros es que no solo se dispone del valor de la distancia que es lo que a priori nos interesa, sino que utilizando por ejemplo la amplitud se puede estimar la fiabilidad de los valores de distancia siendo esta mayor cuanto mayor sea el valor de amplitud.

Las principales ventajas que presentan las cámaras ToF son las siguientes:

- Tamaño relativamente contenido y compactas.
- Rango de distancias de trabajo muy amplio pudiendo llegar desde unos pocos centímetros a decenas de metros.
- No requieren de iluminación externa para su funcionamiento, pudiendo trabajar en la oscuridad.
- Consiguen resultados estables en cuanto a repetitividad y precisión
- La cantidad de "frames" o imágenes por segundo que pueden suministrar es bastante aceptable.

Sin embargo, también tienen algunas desventajas:

- La resolución conseguida por este tipo de cámaras suele ser bastante baja.
- Cuando en la escena hay objetos que se mueven a una velocidad relativamente alta pierden mucha precisión pudiendo dar valores erróneos.

- La precisión en la medida de profundidades no es muy alta.
- Algunos tipos de material y algunos colores afectan a su funcionamiento dando lugar a errores en la medida.
- En ambientes muy iluminados, sobre todo con luz natural pueden presentar fallos de medida.

5. Punto de partida, herramientas y puesta en marcha.

5.1. Punto de partida

Este trabajo de fin de grado parte del avance realizado por compañeros de cursos anteriores en esta misma materia, por lo tanto, es necesario hacer un análisis de la situación de la que partimos.

Principalmente se parte del trabajo realizado por Carlos Castedo Hernández en su trabajo de fin de grado [6]. El objetivo del mismo era la obtención de una imagen 3D de una mano mediante la técnica del estéreo utilizando dos cámaras.

Tal y como se ha explicado en el apartado 4.2.1, el estéreo consiste en la comparación de las 2 imágenes de una escena captadas desde puntos ligeramente diferentes. De modo que si encontramos un mismo punto en las 2 imágenes y encontramos la distancia existe entre las 2 imágenes y conociendo la distancia a partir de la cual se han capturado las imágenes podremos calcular la distancia a la cual se encuentra dicho punto.

El proceso que se sigue es hacer esta comparación para todos y cada uno de los puntos de las imágenes dando lugar a un mapa de disparidades a partir del cual se pueden obtener las coordenadas XYZ de todos los píxeles de la Ilustración 19.

Carlos Castedo en su trabajo diferencia 2 modos de trabajo, uno en el cual realiza el mapa de disparidades directamente a toda la imagen y otro en el que se realiza antes una búsqueda por color para eliminar así de la imagen las zonas menos relevantes. Como, por ejemplo, el fondo de a escena o la palma de la mano.

Ilustración 19 Mapa de disparidades

Por último, utilizando este mapa de disparidades, mediante funciones de OpenCV se consigue obtener una matriz de puntos con coordenadas cartesianas XYZ las cuales constituyen la nube de puntos de la escena.

Tal y como veremos a continuación se han utilizado ambos métodos con diferentes resultados y conclusiones.

5.2. Software

Durante todo este documento, se viene hablando reiteradamente tanto de OpenCV como de PCL y a continuación se procederá a realizar una explicación y análisis de los mismos. Ambos son librerías de código abierto las cuales contienen funciones pensadas para trabajar con imágenes y nubes de puntos. Ambas son utilizadas comúnmente en aplicaciones de visión artificial debido a su versatilidad, facilidad de uso y eficiencia.

5.2.1. Ubuntu (Linux)

Ubuntu [7] es un sistema operativo que está basado en GNU/Linux. A pesar de ser de código libre, pertenece a *Canonical* la cual se financia a través de medios vinculados al sistema operativo como el soporte técnico. El hecho de que sea de código libre permite que exista una gran comunidad de desarrolladores que proporcionan un soporte al sistema y a otras variaciones del mismo como: Ubuntu MATE, Kubuntu, Xubuntu o Ubuntu Studio.

Existen diversas versiones de Ubuntu para escritorio, Canonical publica una nueva cada 6 meses, garantizando soporte técnico para 9. Además, cada 2 años publica una versión LTS (Long Term Support) en las cuales se garantizan 5 años de soporte técnico con actualizaciones y mejoras periódicas.

El origen de Ubuntu es una variación del código base de *Debian,* el objetivo del mismo es el de conseguir una distribución más sencilla para los usuarios finales

mejorando algunas características originales y creando otras nuevas como la instalación y gestión de programas.

Una de las características más importantes de Ubuntu es el uso de la interfaz de usuario. En sus comienzos hacían uso del escritorio GNOME el cual posee un panel inferior para visualizar las ventanas y un panel en la parte superior para los menús y la información del sistema, pero desde la versión 11.04, Canonical lazó su propia interfaz denominada Unity, la cual fue diseñada para mejorar al máximo el área de interacción del escritorio de Ubuntu tal y como vemos en la Ilustración 20.

En cuanto a las características de Ubuntu, en sus últimas versiones, Ubuntu soporta arquitecturas tanto de 32 como de 64 bits. Además, es capaz de actualizar todas las aplicaciones del equipo a la vez haciendo uso de repositorios lo que potencia en gran medida su uso.

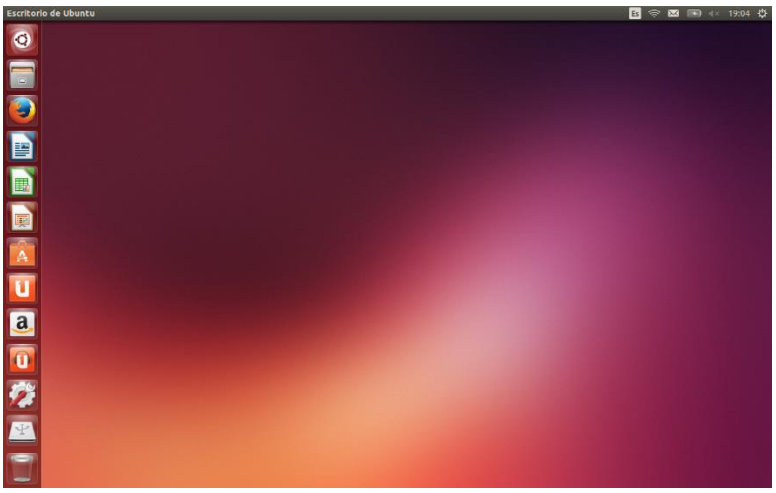

Ilustración 20 Escritorio Ubuntu

Ubuntu no solo se trata de un sistema operativo de escritorio o para servidores si no que ha conseguido extenderse hasta los smartphones, las tabletas y las televisiones. Estas otras versiones se caracterizan por ser también de código libre y encontrarse en estos momentos en desarrollo lo que limita enormemente su expansión de uso.

Los requisitos que Ubuntu necesita que una máquina tenga para poder ser instalado en ella han ido variando ligeramente entre versiones, pero estas son una referencia:

- Procesador x86 a 700MHz.
- Memoria RAM de 512Mb.
- Espacio libre en el disco duro de 5GB.
- Tarjeta gráfica y monitor con resoluciones de 1024x768.

Como se puede ver, los requisitos no son muy amplios lo que facilita en gran medida su uso en gran cantidad de dispositivos, incluyendo tarjetas como *RaspberryPi*.

Para el caso concreto de este trabajo se ha recurrido a la última versión LTS de Ubuntu ya que es la que mayor tiempo de soporte técnico nos garantiza actualmente. Se trata de la versión 16.04 LTS que se lanzó el 21 de abril de 2016 y tendrá soporte hasta abril de 2021. Esta versión trae consigo algunos cambios respecto a versiones anteriores como son la mejora del centro de Software, introducción del calendario de GNOME o mejoras en la integración con el gestor de archivos.

5.2.2. OpenCV

OpenCV [8] es una librería de funciones orientada a visión por ordenador en tiempo real que fue desarrollada originalmente por Intel. Se trata de una librería multi-plataforma (Windows, GNU/Linux, MAC OS X) y además utiliza la licencia BSD, lo que permite ser utilizada libremente tanto para propósitos comerciales como para investigación.

Contiene más de 500 funciones que abarcan una gran cantidad de áreas en la visión artificial como calibración de cámaras, visión por estéreo, reconocimiento facial… Su programación está realizada en lenguaje C++ y algunas de sus características son:

- Son compatibles con The Intel Processing Library (IPL) y utiliza The Integrated Performance Primitives (IPP) para aumentar el rendimiento en el caso de que el sistema disponga de ellas.
- No utiliza librerías numéricas externas.
- Tiene interfaces para otros lenguajes y entornos de programación como Python, Java o Matlab/Ovtave.

El modo en el que OpenCV se estructura se puede dividir de la siguiente manera:

- Estructuras y operaciones básicas.
- Procesamiento y análisis de imágenes.
- Análisis del movimiento y seguimiento de objetos.
- Análisis estructural.
- Reconocimiento de objetos.
- Reconstrucción tridimensional.
- Calibración de cámaras.
- Interfaces gráficos y adquisición de vídeo.

Para concluir, el uso de OpenCV en aplicaciones de visión artificial facilita enormemente la programación y la optimización del código en comparación con otras herramientas orientadas al mismo propósito. Además, posee una documentación extensa y una gran comunidad que permiten un rápido aprendizaje y que mantienen a OpenCV en continuo desarrollo.

5.2.3. PCL

Point Cloud Library (PCL) [9], es una libraría de código abierto de algoritmos para el procesamiento de nubes de puntos y tratamiento de geometrías en 2 y 3 dimensiones. Sus funciones tratan una amplia gama de áreas en la visión artificial, enfocándose principalmente en el espacio tridimensional. Algunas de éstas pueden ser filtrado, reconstrucción de superficies, segmentación…

PCL fue presentada en marzo de 2010 y su primera versión fue lanzada en mayo de 2011. Al igual que OpenCV se trata de una herramienta multi-plataforma y de código abierto que trata de expandirse buscando la mayor versatilidad posible. Puede ser utilizada tanto en Windows como en GNU/Linux como en MAC OS X. Además, su uso es gratuito y libre. Actualmente, en mayo de 2017 la versión más actualizada es la 1.8.0.

Una ventaja muy importante de PCL es que está desarrollada en pequeñas librarías que pueden ser compiladas por separado. Esto hace que para sistemas de poca potencia computacional o almacenamiento sea más llevadero la utilización de sus funciones.

PCL hace uso de otras librerías como son las librerías “Boost” o las “Flann”. Además, existe la posibilidad de mejorar el rendimiento de sus funciones haciendo uso de la GPU del sistema lo que potencia enormemente las posibilidades de utilización.

Para concluir, PCL al igual que OpenCV consigue facilitar la labor de investigación y desarrollo de aplicaciones de visión artificial, aportando un lenguaje de alto nivel con un aprendizaje relativamente sencillo y con una documentación perfectamente ordenada y extensa [10]. Así como en el ámbito de la visión en 2D OpenCV presenta ventajas frente a PCL, en 3D PCL supone un gran avance y por ello se ha decidido recurrir a estas librerías.

5.3. Herramientas para el desarrollo

Acabamos de ver las herramientas software que se van a utilizar en el desarrollo de este trabajo, sin embargo, también se hará uso de otras herramientas hardware y a continuación se realizará un estudio de las mismas.

5.3.1. Cámaras Logitech C310

Se tratan de cámaras web de calidad baja pero que servirán para poder analizar los avances que se vayan consiguiendo. Haremos uso de 2 cámaras idénticas colocadas sobre un soporte metálico, Ilustración 21, que las dejará fijas y alineadas entre ellas para conseguir la mejor calidad posible en el estéreo.

Ilustración 21 Cámaras C310

Las prestaciones de las cámaras son:

- Compatibilidad con sistemas Windows vista, 7, 8 y 10.
- Captura de fotos de hasta 5 megapíxeles.
- Captura de video HD de hasta 1280 x 720 píxeles.
- Tecnología Logitech Fluid Crystal.
- USB 2.0.

Como se analizará más adelante el uso de estas cámaras es únicamente para el desarrollo del trabajo ya que en una hipotética aplicación real de laparoscopia estas cámaras no serían válidas por su tamaño, resolución y óptica.

5.3.2. Cámara IFM O3D303

Ilustración 22 Cámara IFM O3D303

Se trata de una cámara de tiempo de vuelo o ToF (Time of flight) de la empresa alemana Ifm, [11]. Tal y como se explicó en el apartado 4.2.3, el tiempo de vuelo consiste en la obtención de la tercera dimensión a partir de contar el tiempo que tarda un haz de luz en ir y volver de la cámara al objeto en cuestión.

Debido a causas externas no se ha podido utilizar esta cámara en el desarrollo de este trabajo, pero sí que se pudo hacer un análisis de la misma y de las posibilidades que tendría para su uso.

Las características principales de esta cámara son:

- Distancias de operación de 30cm a 8m.
- Frecuencia de obtención de imágenes 25 Hz.
- Ángulos de apertura de 60ºx45º.
- Resolución de 176 x 132 píxeles.
- Conexión vía Ethernet.

- Temperaturas de funcionamiento de -10º a 50º.

Las principales ventajas que el uso esta cámara tiene frente a las Logitech son esencialmente que no sería necesario realizar el estéreo con todo lo que ello supone, tiempo de computación, tener únicamente una cámara, sin calibraciones… Además, la tecnología de esta cámara hace que no sea necesaria la iluminación externa ya que ella misma produce los haces de luz. Esto supone una clara ventaja frente al estéreo ya que en una cirugía laparoscópica no se puede garantizar siempre una correcta iluminación en toda la escena.

En contra, esta cámara en concreto tiene algunos problemas que se debería solventar para poder ser utilizada en nuestra aplicación. Tiene el problema del tamaño, ya que a pesar de utilizar una sola cámara frente a 2 su tamaño es muy grande, pero el principal problema que tiene es que no está realmente diseñada para la toma de imágenes de forma continua sino de forma muestreada. Esto hace que si utilizamos el modo continuo la cámara elevará rápidamente su temperatura pudiendo llegar al punto en el que se debe parar para no poner en riesgo su funcionamiento. Por último, para utilizar esta cámara es necesario el uso de unos drivers concretos para que la cámara y PCL se comuniquen correctamente.

A pesar de estos inconvenientes el uso de esta cámara o de alguna similar con su misma tecnología puede suponer un gran avance respecto al estéreo.

5.3.3. Guantes

Tal y como se explica en el trabajo de fin de carrera de Carlos Castedo [6], para la realización del estéreo se distinguen 2 modos de funcionamiento y, en uno de ellos se hace uso de un guante negro con algunos dedos recortados para conseguir el mapa de disparidades perteneciente únicamente a esos dedos que están al aire. Esto lo consigue filtrando por color y en el desarrollo de este trabajo se hará uso de su código y por lo tanto del guante, Ilustración 23.

Ilustración 23 Guante

5.3.4. Otros

Para terminar, se comentará brevemente otros objetos y herramientas que se utilizarán:

- **Soporte de cámaras**: Se trata de un soporte en el que colocaremos nuestras cámaras, tiene altura regulable y una cuadrícula en su zona baja. Nos ayudará a comprobar la veracidad de las coordenadas que nos proporcione el estéreo ya que podremos ver a qué altura real están las cámaras.
- **Panel de calibración**: Se trata de una cuadrícula en forma de tablero de ajedrez cuyas medidas son conocidas y que nos valdrá para calibrar las cámaras y conseguir así unos resultados más fiables. El tamaño de cada cuadrado es de 18mm de lado y hay 9 esquinas de ancho por 5 de alto. Estos valores son los que nos pedirá el código de calibración.

5.4. Puesta en marcha

Antes de comenzar con el desarrollo y análisis del código vamos a ver los pasos previos que se deben realizar para tener todas nuestras herramientas en orden y dispuestas para el proceso.

5.4.1 Software

Comenzamos realizando la instalación de Ubuntu 16.04LTS en la máquina que vamos a emplear tanto para el desarrollo del código como para las pruebas del mismo. Una vez Ubuntu está instalado y actualizado podemos proceder a la instalación de algunos de los programas que necesitaremos para la realización del código, en este caso, se decidió utilizar Code::Blocks IDE [12], se trata de un programa con una interfaz de usuario que incluye un compilador y nos facilita en gran medida la tarea de escritura, compilación y prueba del código.

Continuamos con la instalación tanto de OpenCV como de PCL, no importa cuál de las 2 se instale primero, pero lo que sí es importante es que en el momento que la primera sea instalada se compruebe exhaustivamente su correcto funcionamiento ya que este tipo de instalaciones pueden dar lugar a errores y fallos en el funcionamiento que más tarde pueden ser muy complicados de solucionar.

Tras cada instalación es recomendable la configuración del compilador de Code::Blocks para que sea capaz de encontrar las librarías tanto de OpenCV como de PCL para ello habrá que indicarle la ruta de cada una de las librerías. Existe un problema de compatibilidad entre OpenCV y PCL, más concretamente si en las rutas del compilador le indicamos las de OpenCV2 y PCL el programa será incapaz de compilar debido a esta incompatibilidad, por ello, la solución adoptada en este caso es la de eliminar las rutas a OpenCV2 y así eliminar este problema, tiene el inconveniente de que perdemos algunas de las funcionalidades de OpenCV, pero en este caso ninguna que sea necesaria para nuestra aplicación.

Es importante mencionar la importancia de la correcta ejecución de estas instalaciones, ya que suponen un paso muy importante para poder continuar en el desarrollo del código.

Para finalizar con la parte del software es importante comprobar el correcto funcionamiento de las cámaras y de la compatibilidad de las mismas. En nuestro caso, existe un problema entre las cámaras Logitech C310 y Ubuntu debido a que éstas no están preparadas para trabajar con este sistema operativo. Estas cámaras capturan tanto imagen como sonido, ya que tienen integrado un micrófono, pero debido a trabajar con Ubuntu la captura de sonido no será posible (no afecta al desarrollo del trabajo) y además, cuando éstas toman una imagen y el programa está lanzado desde el terminal del Ubuntu lanzan un mensaje en el mismo con la frase: "Corrupt JPEG data: 2 extraneous bytes before marker 0xd1" esto es debido a que al no tener unos drivers específicamente diseñados para Ubuntu la comunicación entre la máquina y las cámaras no es perfecta. A pesar de estos problemas las imágenes capturadas por las cámaras son correctas y perfectamente útiles para ser utilizadas en la aplicación.

5.4.2. Hardware

En cuenta al hardware, lo que vamos a hacer es preparar nuestro soporte, con las cámaras y disponerlo todo de manera que la conexión de las cámaras al ordenador sea lo más cómoda posible.

Para ello fijamos las cámaras al soporte tratando que la unión no permita el movimiento de las cámaras ante posibles golpes o movimientos pero que la misma no sea permanente Ilustración 25.

Finalmente, para facilitar el proceso de calibración de las cámaras, el panel de calibración se fija a un tablero el cual nos servirá para mantenerlo plano y facilitar su uso, Ilustración 24.

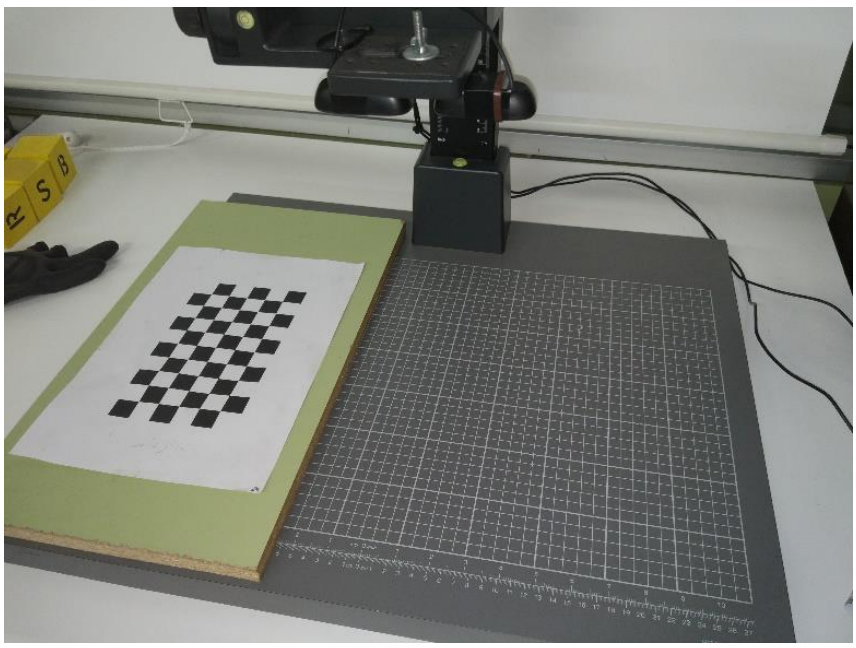

Ilustración 24 Tablero de calibración y soporte

Ilustración 25 Cámaras en el soporte

6. Desarrollo de la aplicación

Comenzamos así el desarrollo de la aplicación. Para abarcar todo el proceso seguido se va a dividir este apartado en diferentes secciones que explicarán paso a paso qué problema se presenta, las posibles soluciones y la solución escogida. Además, durante la ejecución del proyecto se abordó el problema principal desde diferentes puntos de vista para encontrar así el mejor método y, conseguir así, alcanzar los objetivos propuestos. Explicaremos cada una de ellas para que posteriormente se haga un análisis del resultado arrojado y así valorar sus ventajas y desventajas.

6.1. Conversión a nube de puntos de PCL

Algo que tienen en común todas las propuestas realizadas en este trabajo es que todas ellas comienzan con el paso de la información obtenida por el procesamiento del estéreo, el cual se encuentra en forma de mapa de disparidades, a una nube de puntos con la cual PCL podrá trabajar y aplicar sus funcionalidades sobre ella.

Para llevar a cabo esta tarea recorreremos el vector que posee las coordenadas XYZ calculadas por el estéreo, este vector es llamado “Ima3D” por Carlos Castedo en su código. Mientras vayamos recorriéndolo, iremos insertando cada punto en la nube de puntos hasta haber introducido todos.

Es importante destacar que Carlos Castedo en su trabajo diferenció 2 modos de trabajo en los cuales en uno de ellos realizaba el estéreo a toda la imagen directamente y en otro aplicaba antes un filtro de color para eliminar ciertas partes de la imagen y mantener otras. Estos 2 modos de trabajo serán utilizados más tarde en los diferentes enfoques que se le han dado al problema. Sin embargo, esto no afecta al paso de la información a las nubes de puntos ya que sea cual sea el modo de trabajo que estemos utilizando el vector “Ima3D” tendrá la información de aquellos puntos que sean importantes para nosotros, ya sean todos los de la imagen o solo los que hayan pasado el filtro de color.

Este proceso tiene varios aspectos que deben ser tenidos en cuenta antes de realizarse, por un lado, debemos pensar si realmente es interesante introducir absolutamente todos los puntos calculados o podemos excluir algunos, ya sea porque no son necesarios o porque no es necesario tener demasiados.

Si vemos todos los puntos calculados para una escena cualquiera, vemos que la resolución que arroja el proceso está relacionada con la de las cámaras. Aunque

para el procesado de imágenes en 2 dimensiones la resolución de estas cámaras puede llegar a ser incluso baja, se puede ver que en el procesado 3D un exceso de información lastra en gran medida el rendimiento del mismo, debido a que los cálculos que realiza son más y más complejos en comparación con los procesos de imágenes tradicionales.

Además, tras el procesamiento del estéreo se crea una región de puntos en la imagen en la cual no se han podido calcular las coordenadas XYZ debido a que una de las 2 cámaras no veía esa región. Esto hace que al pasar la información tal cual a PCL se genere la misma zona en la nube de puntos esta vez en lugar de representarse de color negro se representa a una altura que está fuera de la realidad tal y como vemos en la Ilustración 27. Si no eliminamos esta región, PCL la tratará como información válida y ésta podrá perjudicar tanto al rendimiento como al resultado final del proceso. Por ello, se decide incluir en la nube únicamente aquellos puntos que se encuentren en un rango de valores.

Ilustración 26 Nube de puntos guardando todos los puntos.

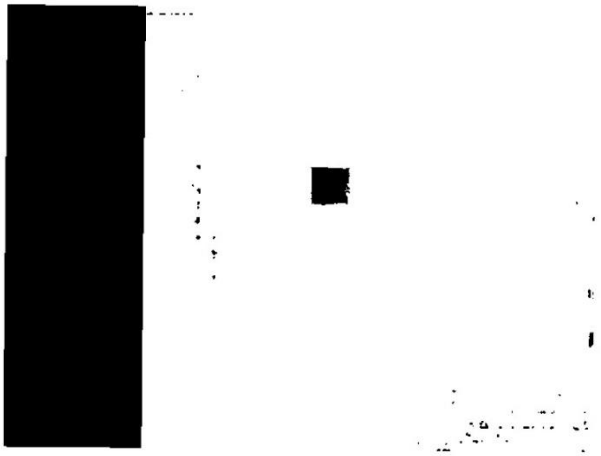

Ilustración 27 Representación de la región sin calcular.

Este proceso de criba de puntos nos crea un nuevo problema en cuanto a la ejecución del código se refiere, y es que cuando creamos una nube de puntos debemos indicarle a PCL de antemano qué tamaño va a tener dicha nube, esto hace que nos veamos obligados a realizar un primer recorrido para contar cuántos puntos pasarán los criterios y una segunda para introducirlos en la nube. Esto es algo muy poco eficiente en cuanto a tiempo de cálculo se refiere, pero supone una mejora frente a no realizar dicha criba.

Por otro lado, es importante analizar qué tipo de nube de puntos se va a utilizar para que almacene la información, PCL dispone de varios tipos haciendo distinción principalmente en el tipo de punto que la nube va a almacenar. Existen los Puntos XY los cuales solo tienen 2 coordenadas en el espacio, los puntos XYZ que tienen 3, los XYZI los cuales a parte de las tres dimensiones tienen un parámetro más que hace referencia a la intensidad pudiendo así crear una nube de puntos en escala de grises. También están los puntos XYZRGB que al igual que los anteriores ubican puntos en el espacio tridimensional pero además les otorgan color tal y como vemos en la Ilustración 29.

Ilustración 28 Ejemplo nube de puntos XYZ

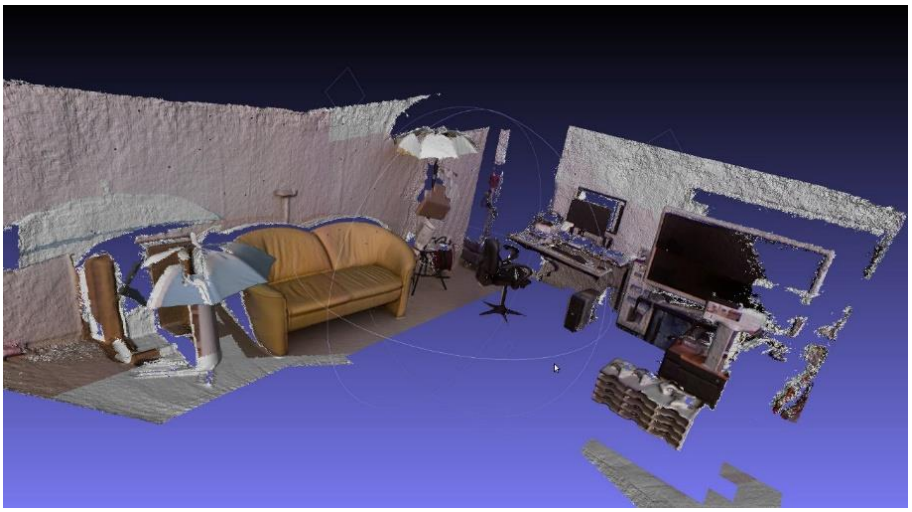

Ilustración 29 Nube de puntos XYZRGB

Dado que en el proceso para conseguir el estéreo se ha convertido la imagen a escala de grises y que más tarde la función que nos proporcionaba las coordenadas de los puntos de la imagen no nos da la información del color, lo más oportuno es utilizar un tipo de punto XYZ. Esto, además, resulta lógico ya que como se vrá más adelante el uso del color en la nube de puntos no va a ser necesario en ningún proceso planteado por lo tanto añadirlo solo provocaría un aumento del uso de memoria y tiempo de ejecución.

Otro aspecto importante a tener en cuenta es la ubicación del sistema de referencia de OpenCV y las unidades que utiliza, ya que si pasamos directamente la información a PCL todos estos parámetros se mantendrán constantes. La respuesta es que el sistema de referencia lo sitúa en el objetivo de la cámara izquierda, dejando el eje z como la profundidad en la dirección de las mismas. Por otro lado, las unidades que utilizan son las décimas de milímetro ($10^{-4}m$). Cabe la posibilidad de mantener esta información tal y como la proporciona OpenCV, pero finalmente se decidió modificar las unidades a milímetros para que la representación en PCL sea más sencilla de ver y analizar. Para ello se deberá dividir por 10 las coordenadas de todos los puntos que se deseen añadir a la nube.

Una vez se han valorado todos los aspectos que se acaban de comentar, se puede proceder al volcado de la información y así conseguiremos una nube de puntos que nos permite comenzar a utilizar las funciones propias de PCL con total libertad.

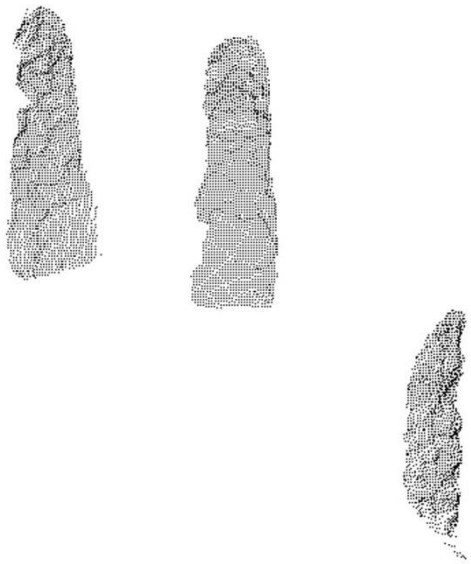

Ilustración 30 Nube de puntos con filtro por color

En la Ilustración 28 e Ilustración 30 se pueden ver las nubes de puntos obtenidas en los 2 modos de trabajo del estéreo. Por un lado, en la Ilustración 28 el estéreo es completo a toda la imagen y se pueden ver tanto la mano como la mesa. Por otro lado, en la Ilustración 30 se ha aplicado un filtro de color que elimina de la imagen todo lo que no sea del color de la piel. En el momento de capturar la imagen se tenía puesto el guante del apartado 5.3.3 por ello lo único que se aprecia en la

nube de puntos son 3 dedos. A diferencia de la otra nube de puntos, en ésta solo tenemos información que es válida para la aplicación ya que son los dedos los que nos dan la información que necesitamos y no el resto de la mano. Además, se ha conseguido eliminar en gran medida la influencia del ruido en la nube lo que será de gran ayuda al trabajar con ella.

A continuación, se desarrollará el problema utilizando los 2 modos de obtención del estéreo.

6.2. Desarrollo a partir de la nube de puntos completa.

Esta fue la primera de las alternativas que se llevaron a cabo, las razones de escogerla fueron diversas.

- Resulta más sencillo y rápido hacer el estéreo a toda la imagen y no tener que pasar el filtro de color a las 2 imágenes captadas por las cámaras lo que haría tener menos coste computacional y así reducir el tiempo de procesado por imagen.
- En este modo de trabajo no es necesario emplear un guante que deje los dedos al descubierto lo que sería un gran beneficio para llevar a la práctica la aplicación.
- Tener toda la información de la escena puede servir para obtener información adicional en caso de que fuera necesaria, como la posición de otras herramientas de trabajo.
- Se podría obtener la información de los 5 dedos ya que todos podrían ser visualizados, sin tener en cuenta que unos se tapen a otros en determinadas posiciones de la mano.

Tal y como se ha comentado en el apartado anterior, se comenzó observando diferentes aspectos:

- Se creaba una zona a gran altura correspondiente a la zona en la que el estéreo no podría realizarse.
- La cantidad de puntos de la nube podría ser demasiado grande para unos cálculos rápidos.
- Había mucho ruido en la nube de puntos.

6.2.1. Filtrado por rango de la nube de puntos

La zona negra que se crea en el proceso del estéreo se procesa como parte de la nube de puntos, este área posee una gran cantidad de puntos y se pudo comprobar que afectaba en gran medida tanto a los tiempos de lectura y escritura de la nube de puntos debido a su tamaño como a los resultados de diferentes funciones que se aplicarán posteriormente. Por ello es totalmente necesario eliminarla de la nube de puntos.

Para ello, existían 2 opciones, eliminarla antes de generar la nube de puntos que es la opción que se comentó en el apartado anterior o eliminarla mediante mecanismos de PCL. Eliminarla antes traía consigo la ventaja de que la lectura y escritura de la nube de puntos se mejoraba, pero a cambio nos veíamos obligados a realizar un bucle para calcular previamente el número de puntos que se escribirían en la nube y otro para introducirlos. El balance de este proceso era bueno, pero poco eficiente en cuanto a programación se refiere.

La otra opción, es la de eliminarlo mediante mecanismos de PCL, esta resulta mejor en cuanto a programación ya que PCL tiene una función específica para ello, pero no mejora los tiempos de lectura de la nube de puntos. La función específica de PCL para conseguir esta tarea es un filtro llamado “PassThrough Filter”. Este filtro consiste en crear un objeto de filtrado al cual le damos los parámetros a partir de los cuales decidirá si un punto es aceptado o rechazado. En este caso los parámetros serían los valores máximo y mínimo que el punto podría tener en su coordenada Z, que es la que determina la altura a la que se encuentra.

Los valores que insertemos han de estar cuidadosamente calculados ya que si nos pasamos o nos excedemos en los mismos se correrá el riesgo de no eliminar la zona o de eliminar áreas importantes de la nube. Para ello, se estima que la tanto la mano como la mesa se encuentran a no más de 60cm de las cámaras. Este valor es razonable ya que la zona que queremos eliminar tiene un valor de altura de 1.4 metros de altura por lo tanto los valores finales que introducimos al filtro son que los valores Z de cada punto han de estar entre 0 y 600mm.

El resultado que obtenemos con este filtro es el esperado, todos los puntos que superan los 600mm en su coordenada Z son eliminados de la nube quedando el resto intactos. Además, se comprueba que el tiempo de computación requerido para llevar a cabo el filtrado es bastante corto, lo que lo convierte en una buena opción para solventar el problema.

6.2.2. Cantidad de puntos en la nube.

Determinar cuántos puntos son necesarios que tenga la nube de puntos para que los resultados sean correctos, pero también se obtengan de manera rápida y eficaz será un problema constante a lo largo de todo el desarrollo del trabajo.

Como ya se ha explicado anteriormente tener una gran cantidad de puntos facilita el encontrar la solución correcta en determinadas funciones de PCL, sin embargo, tener un exceso de ellos puede lastrar en gran medida otras tareas que han de llevarse a cabo.

Para hacerse una idea de la cantidad de puntos que se pueden llegar a manejar en una sola nube de puntos, si abrimos la nube proporcionada por el sistema de estereovisión, el número de puntos de la escena asciende a 307.200 puntos. Este dato puede ser visualizado en el archivo .pcd que se crea al guardar una nube de puntos tal y como se ve en la Ilustración 31.

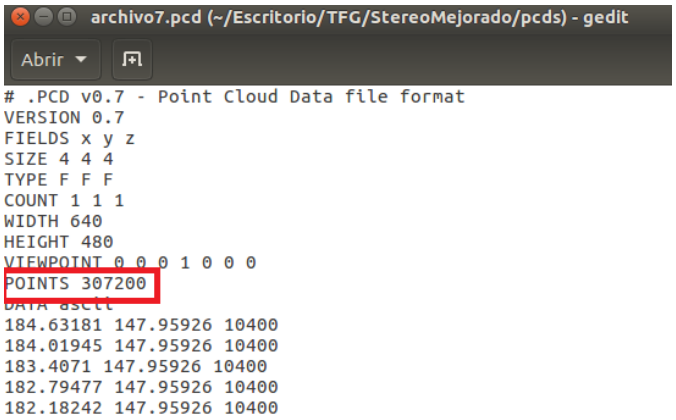

Ilustración 31 Representación de un archivo .pcd

En cambio, si investigamos un poco y comparamos con otras escenas nubes de puntos capturadas mediante otras tecnologías vemos que la cantidad de puntos que se suelen manejar para este tipo de aplicaciones raramente superan los 100.000 puntos.

Para modificar la cantidad de puntos en una nube de puntos disponemos de diversas opciones, a continuación, se explicarán 2 de ellas.

La primera de ellas consiste en la posibilidad de reducir la nube eliminando puntos de manera aleatoria. Éste proceso de hecho, se podría realizar antes de escribirlos en la nube de PCL junto con el proceso que se comentó en el apartado anterior de eliminar la zona negra. Para llevarlo a cabo, bastaría con generar un número aleatorio cada vez que fuéramos a escribir el punto en la nube y si este número quedara por debajo de un valor fijado el punto quedaría descartado.

Así, por ejemplo, si generamos números aleatorios entre 0 y 1 y fijamos la frontera para aceptar un punto en 0.5, al final del proceso habríamos eliminado aproximadamente la mitad de los puntos. Modificando la frontera podemos hacer que la criba sea más exhaustiva o no.

Este proceso plantea ventajas y desventajas, por un lado, al realizar el proceso durante la escritura de la nube reducimos desde el principio el tamaño de la misma minimizando los tiempos de escritura y lectura. Sin embargo, por el otro lado, la eliminación de puntos de manera aleatoria tiene el inconveniente de que cualquier punto puede ser eliminado pudiendo así perder información importante o pudiendo dejar regiones de la nube vacías si todos los puntos son borrados.

La segunda haría uso de una herramienta de PCL, se trata de un filtro llamado “VoxelGrid Filter”. Este filtro consiste en crear una “caja” de unas determinadas dimensiones para después ir pasando esa caja por todos los puntos de la nube. En cada momento, todos los puntos existentes en la caja serán simplificados a uno solo que se encuentre en el centroide de la caja. De este modo, se consigue simplificar agrupaciones de puntos mientras que aquellos puntos aislados se mantendrán prácticamente intactos.

El resultado de aplicar el filtro se puede ver en la Ilustración 32 y la Ilustración 33. Para el filtrado se ha creado una caja cúbica de 3mm de lado.

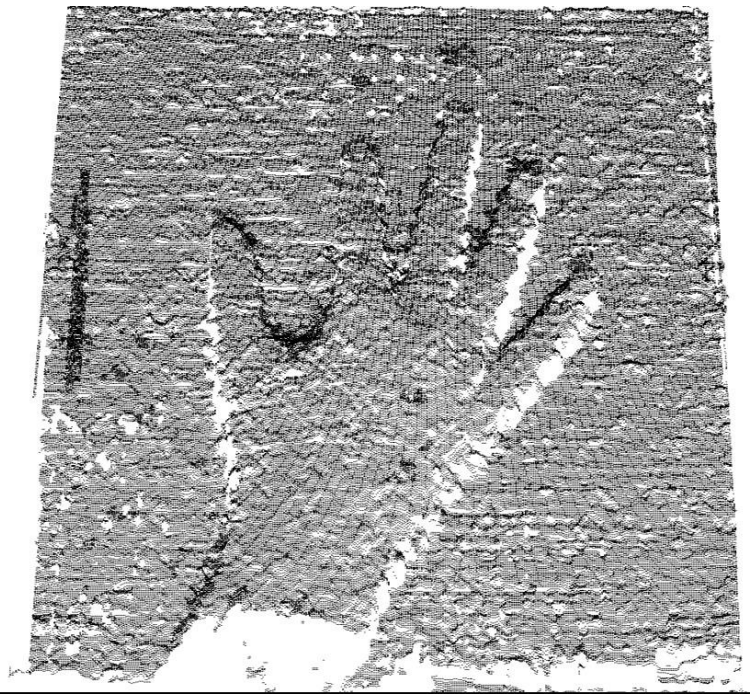

Ilustración 32 Nube con todos los puntos

Ilustración 33 Nube reducida

Este tipo de filtro tiene la principal ventaja de que no elimina información importante ya que respetará todas las formas de la nube y las zonas en las que los puntos estén algo más aislados.

El tamaño de dicha caja determinará en qué medida la reducción será mayor o menor. A mayor tamaño de caja, mayor cantidad de puntos eliminaremos y por tanto más reduciremos la nube. Además, este filtro tiene la ventaja de que la caja no ha de ser cúbica, si no que podemos darle diferentes tamaños en las direcciones de los ejes XYZ consiguiendo así diferentes criterios de eliminación entre ejes. Por ejemplo, tal y como vemos en Ilustración 34, se ha dado mayor peso a la eliminación de puntos en el eje vertical mientras que en el horizontal se han respetado los puntos originales.

Ilustración 34 Reducción asimétrica

Finalmente, se ha optado por utilizar el filtro de PCL utilizando una caja cúbica ya que no nos interesa diferenciar entre ejes para la reducción de la nube de puntos.

6.2.3. Tratamiento del ruido

Tal y como se explicó en el apartado 3.3.4 de este documento, la existencia de ruido en una imagen puede afectar negativamente a determinadas operaciones que se realicen sobre ella, y lo mismo pasa con las nubes de puntos tridimensionales. En este caso el ruido en lugar de manifestarse como variaciones de intensidad o de color se manifiesta con la existencia de puntos en la nube en zonas en las que no hay ningún objeto. Estos puntos suelen denominarse “Outliers” y pueden estar tanto aislados como agrupados y su origen puede justificarse mediante diferentes motivos.

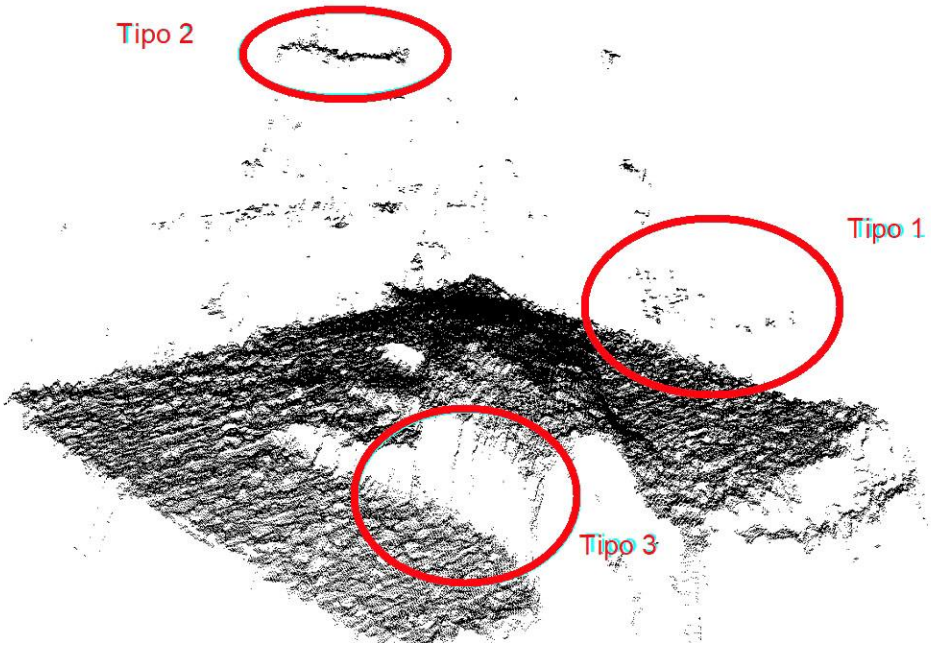

Ilustración 35 Tipos de ruido

En la Ilustración 35 se representan los diferentes tipos de ruido que se nos presentan en nuestras capturas. Se han distinguido 3 tipos y a continuación se analizarán:

- Tipo 1: Puntos aislados o en grupos, pero relativamente dispersos. Se trata del tipo de ruido menos perjudicial y el más sencillo de eliminar, su origen está en errores en cálculo de la disparidad mientras se realizaba el estéreo. Es más común cuanto más homogénea sea la superficie que las cámaras están capturando debido a que esto hace que sea más complicado el encontrar los puntos homólogos en las 2 imágenes, pero en gran parte es inevitable.
- Tipo 2: Grandes agrupaciones de puntos. Son poco comunes, pero realmente complicados de eliminar por completo. Su origen está, entre otras posibles causas, en reflejos que se pueden producir en determinadas superficies. Si la iluminación no está correctamente dispuesta, puede darse el caso de que se produzcan reflejos y éstos sean captados por las cámaras. El problema de esto es que los reflejos no estarán en la misma ubicación en las 2 imágenes ya que las cámaras están ligeramente separadas una de la otra, Ilustración 36. Esta distancia entre reflejos produce errores en el emparejamiento de puntos homólogos y por lo tanto unos valores de disparidad alterados. Este tipo de ruido se debe evitar todo lo posible modificando la iluminación.

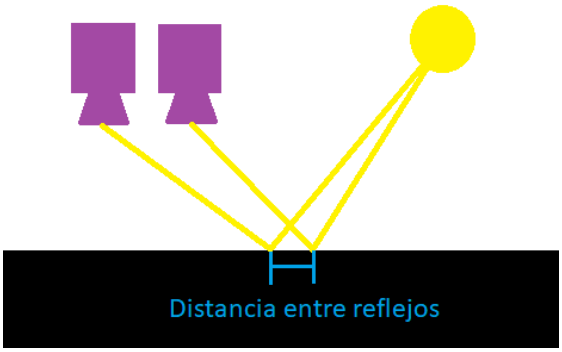

Ilustración 36 Disparidad de reflejos

- Tipo 3: Puntos entre bordes de objetos. Este tipo de ruido a diferencia de los anteriores no es debido a errores en el estéreo, si no, que es un tipo de ruido que es común a todas las tecnologías actuales de captura 3D, en la Ilustración 37 se puede ver un ejemplo del mismo tipo de ruido en una captura con una cámara de tiempo de vuelo (ToF). Consiste en la existencia de puntos entre los bordes de objetos que están a diferentes profundidades. Su existencia se debe a que resulta imposible para las máquinas determinar a qué profundidad se encuentran dichos puntos debido a que una de las cámaras no "ve" esa zona debido a que el objeto la ocluye. Debido a que este tipo de ruido es tan común en visión 3D, PCL posee filtros destinados a su eliminación.

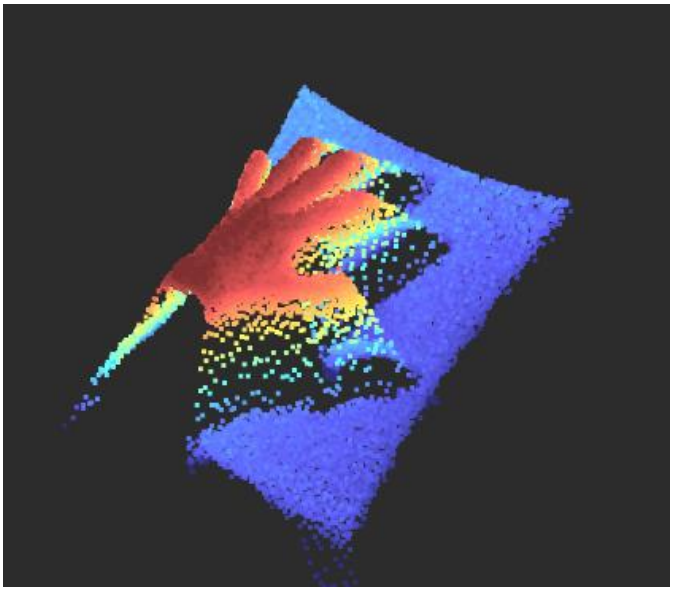

Ilustración 37 Ruido de borde en tecnología ToF

Ahora que ya se conoce los tipos de ruido existentes en las nubes de punto que es obtendrán en la aplicación, es necesario analizar las diferentes herramientas de las que PCL dispone para eliminar o minimizar dicho ruido.

- Filtro de eliminación estadística

El primero de los filtros destinados a la eliminación del ruido que se van a analizar consiste en analizar la distancia media que tiene un punto con el resto de puntos de su vecindario más próximo. Analizando punto a punto estas distancias se puede determinar qué puntos son debidos al ruido y cuáles no, dado que los que más distancia tengan con sus vecinos tienen más posibilidades de ser outliers. PCL denomina a este filtro como “StaticalOulierRemoval filter”.

Para ejecutar este filtro debemos darle la nube de puntos a analizar y la nube de puntos en la que volcará el resultado, estas nubes pueden ser la misma o no. Además, tenemos que darle el número de vecinos con los que tiene que medir la distancia por cada punto y por último la desviación estándar a partir de la cual se considerará a un punto como Outlier.

Ilustración 38 Resultado del filtro de eliminación estadística

Si ejecutamos este filtro sobre nuestra nube de puntos los resultados son los que se pueden ver en la Ilustración 38. Como se puede observar, si se ajustan correctamente los parámetros el filtro es capaz de eliminar totalmente el ruido quedando únicamente algo en los bordes (tipo 3). Es importante tener en cuenta que para realizar este ejemplo se ha aplicado un filtro muy estricto el cual medía la distancia entre muchos puntos para conseguir el mejor resultado posible. Pero en realidad a la hora de la verdad este resultado no podrá ser tan bueno ya que el tiempo de cálculo empleado llega casi a los 5 segundos.

La realidad es que si se espera que el filtro aporte resultados aceptables los tiempos de computación son claramente inaceptables para una aplicación real.

- Filtro de eliminación por condición de radio

De nuevo el objetivo es determinar si un punto es outlier o no, para ello este filtro cuenta cuántos puntos vecinos existen dentro de una hipotética esfera generada en torno a un punto concreto. El tamaño de la esfera es proporcionado por nosotros, así como el número de vecinos que debe haber dentro de la esfera para que el punto no sea considerado como outlier. En la Ilustración 39 vemos un ejemplo en 2 dimensiones, en ella si ponemos que el número de vecinos mínimo es 2, entonces el punto amarillo y el verde quedarían determinados como outliers y eliminados de la nube de puntos.

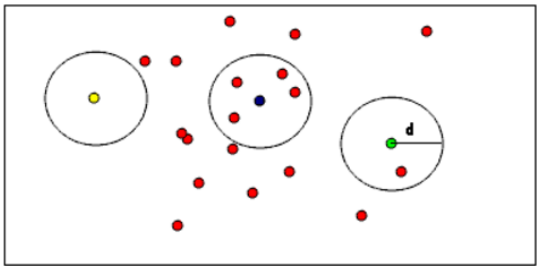

Ilustración 39 Ejemplo de funcionamiento de filtro de radio

Si aplicamos este filtro el resultado es el que se puede ver en la Ilustración 40. De nuevo la eliminación de los puntos aislados es muy buena y de las agrupaciones pequeñas también, pero las más grandes son un problema debido a que al estar los puntos tan juntos es complicado que se detecten como outliers. La solución es ajustar los parámetros de radio de la esfera y de mínimo número de vecinos, pero ajustarlo a la perfección es prácticamente imposible y si queremos quitar los grupos más grandes también estaremos quitando puntos pertenecientes a la mano o a la mesa y eso no es recomendable.

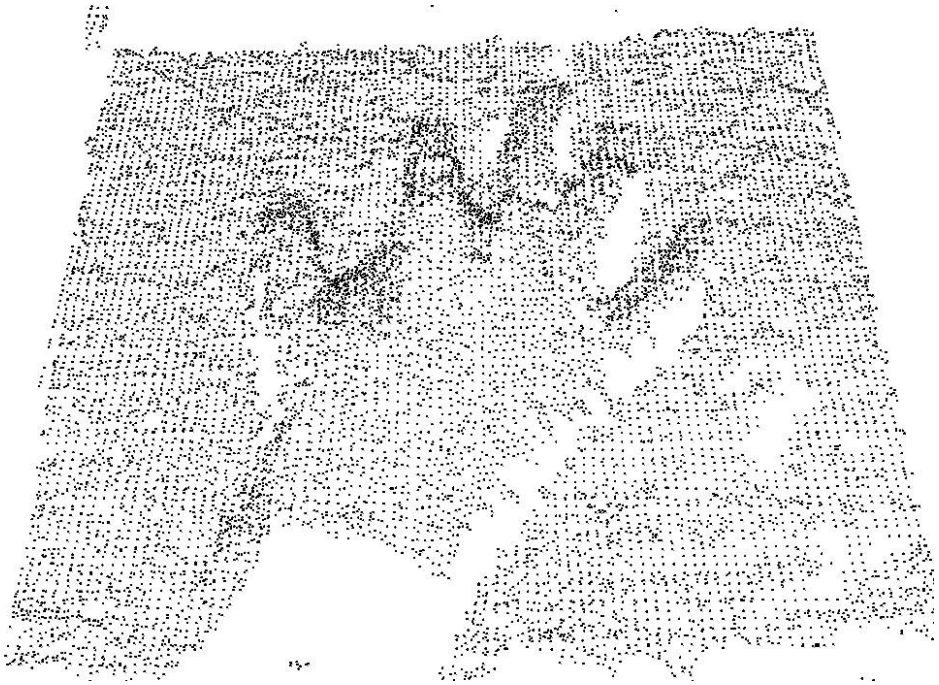

Ilustración 40 Resultado de filtro de eliminación por radio

- **Filtro de eliminación de puntos de borde de objetos.**

Se trata de un filtro dedicado únicamente a eliminar el ruido que se ha denominado de tipo 3. Este filtro se denomina “ShadowPoints” en PCL.

Este filtro se basa en las sombras dejadas por un objeto sobre otro para encontrar los conjuntos de puntos correspondientes al ruido. El rendimiento de este filtro es muy bueno, pero solo es eficaz en imágenes que contengan mucho ruido de este tipo y en este caso no es realmente necesario.

Tras probar detenidamente los filtros mencionados y comparar tanto los resultados obtenidos como los tiempos de computación se puede concluir que si lo que se desea es eliminar por completo todo el ruido de la imagen o minimizarlo al máximo, el mejor filtro es el de eliminación estadística junto con el filtro de borde de objetos. A pesar de ello, para los pasos sucesivos no es realmente necesario que se limpie tanto la imagen ya que la influencia que pueda tener el ruido que quede en el resultado final es muy baja o nula. Debido a esto se ha concluido que con realizar un filtro por radio con sus parámetros debidamente ajustados conseguimos un resultado muy válido empleando un tiempo de computación bajo si lo comparamos con el resto de opciones que se han planteado.

6.2.4. Eliminación del fondo mediante búsqueda de planos

Durante los procesos anteriores se ha fijado la atención en eliminar el ruido de la imagen, sin embargo, algo que es muy importante es poder eliminar los puntos referentes al fondo. Éstos tienen un gran peso ya que ocupan una gran área de la imagen.

Para ello vamos a hacer que el sistema busque conjuntos de puntos que conformen un plano. Esto lo conseguimos con la función “Plane model segmentation”.

Esta función necesita que se le proporcionen diferentes datos para que funcione correctamente. Por un lado, necesita que se le indique qué tipo de modelo ha de buscar, planos, cilindros, esferas… Y, además, el método de búsqueda que ha de emplear, de todas las opciones que ofrece destaca la de RANSAC, [13]. Por otro lado, esta función hace uso tanto de la nube de puntos en la que se va a hacer la búsqueda como de una nube extra la cual almacena los valores referentes a las normales de cada punto. Para obtenerla, se recurre a la función “NormalEstimation”.

Por último, es necesario indicar la tolerancia que puede admitir a la hora de estimar si un punto pertenece al modelo o no y el número de iteraciones máximas que se pueden realizar hasta determinar que el resultado es correcto.

Al ejecutar esta función sobre la nube de puntos, ésta nos devuelve tanto un vector donde se almacenan los puntos pertenecientes al modelo como los parámetros que definen al modelo buscado, es decir, para el caso del plano devuelve

4 valores que definen un plano en el espacio. Estos cuatro valores hacen referencia a la fórmula general de un plano:

$$Ax + By + Cz + D = 0$$

De este modo, mediante el vector de puntos podremos eliminarlos de la nube o guardarlos en una nueva si fuese necesario. Además de poder ubicar el plano perteneciente al fondo de la escena y así se podría comprobar que la estimación de profundidades es correcta midiendo la distancia real de las cámaras a la mesa y comparando con la distancia proporcionada por el programa.

Tras el proceso de eliminación del fondo la nube de puntos queda tal y como se ve en la Ilustración 41 y en la Ilustración 42. Tras todos los procesos se ha conseguido una mano relativamente bien conservada (hay algunas zonas en las que se han perdido puntos de ella) aislada de todo lo demás. Aunque hay algunas agrupaciones de puntos pertenecientes al ruido, ya se ha explicado antes que eliminarlos resulta realmente complicado, se espera que su interferencia en acciones sucesivas no sea demasiado grande.

Ilustración 41 Mano filtrada y sin fondo 1

Ilustración 42 Mano filtrada y sin fondo 2

6.2.5. Filtrado mediante búsqueda de objetos

Existen otras opciones para conseguir una nube de puntos libre de ruido. Todas las opciones que se han planteado anteriormente tenían en común que trataban de buscar los puntos referentes al ruido y eliminarlos. Pero en cambio ahora se va a seguir la lógica inversa, es decir, se va a tratar de detectar aquellos objetos que nos interesen y aislarlos en una nueva nube de puntos.

El principal problema de este razonamiento es, por un lado, encontrar correctamente los objetos presentes en la imagen y por el otro, poder determinar cuál o cuáles de ellos son los que son los que debemos aislar.

Para PCL un objeto es una agrupación de puntos en la cual existe una continuidad espacial y una densidad de puntos suficiente. A diferencia de la visión humana la cual está adaptada para encontrar e identificar una gran multitud de objetos, independientemente de que estén girados, parcialmente ocluidos o sean del tamaño que sean, para una máquina el buscar objetos en una tiene una gran carga computacional y si además hay que tratar de identificarlos ésta se dispara. En este caso nos vamos a limitar a encontrar tantos objetos como podamos para más tarde determinar cuál es el correspondiente a la mano mediante otros parámetros.

Para la detección de objetos, PCL tiene la función “EclideanClusterExtraction”. Esta función consigue la detección de objetos a partir de un algoritmo relativamente sencillo. Consiste en recorrer todos los puntos de la nube buscando los puntos vecinos que se encuentren dentro de un radio dado por el usuario, si existe algún vecino que cumpla esa distancia máxima procede a analizar los vecinos de éste y continua así mientras va almacenando en una lista aquellos puntos que han cumplido la restricción. Una vez no hay más vecinos que analizar se comprueba el tamaño de dicha lista para saber de qué tamaño es el objeto encontrado, si este tamaño está entre unos valores lógicos el conjunto de puntos se determina como un solo objeto y es almacenado para continuar con el resto de la imagen. De este modo, el usuario puede controlar en qué medida el algoritmo debe ser más o menos riguroso a la hora de determinar puntos como vecinos, si se indica un radio pequeño la función será más estricta y tenderá a encontrar más objetos de pequeño tamaño mientras que si el radio proporcionado es grande cabe la posibilidad de que tome objetos que estén relativamente juntos como uno solo.

Para la ejecución de la función, es necesario un paso previo que consiste en la creación de lo que se denomina “Kd-tree”. Se trata de un método de ordenación y agrupación de puntos en el espacio. Su objetivo es el minimizar los tiempos de búsqueda de los vecinos más próximos de cada punto.

El término “k” hace referencia al número de dimensiones de las que dispone el sistema, que en el caso de la aplicación son 3. Se trata de un árbol binario (cada nodo tiene 2 hijos) en el cual en cada nodo se almacena la información de un punto. La construcción del árbol, añadir o eliminar elementos del mismo o conseguir un árbol balanceado son cuestiones que se alejan del propósito de este trabajo, pero

existe una gran investigación en este ámbito, [14]. En la Ilustración 43 se puede ver un ejemplo de un árbol que almacena los puntos de un sistema bidimensional. Se puede ver como cada nivel del árbol hace referencia a un eje y los 2 hijos de cada nodo son ordenados teniendo en cuenta si su valor es superior o inferior al del padre.

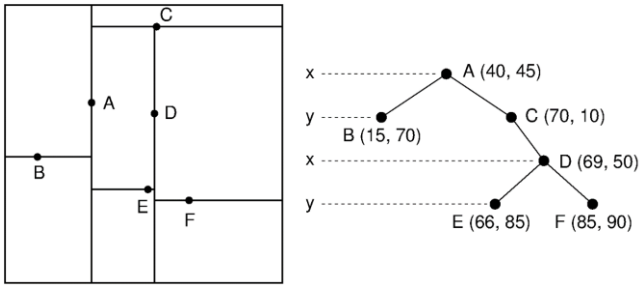

Ilustración 43 Ejemplo de Kd-tree de 2 dimensiones

Este tipo de árboles son muy utilizados en computación ya que para la búsqueda de puntos en grandes conjuntos son realmente eficientes ya que el tiempo extra dedicado a crear y mantener el árbol es inferior al que se ahorra durante la búsqueda.

Una vez el Kd-tree ha sido creado y los parámetros de radio y los rangos de tamaño han sido proporcionados la función podrá realizar la búsqueda. De nuevo es importante mencionar que a mayor tamaño de la nube de puntos más costosa será la operación. Esto refuerza aún más la necesidad de realizar la reducción de tamaño que se ha comentado anteriormente.

Cuando el proceso ha finalizado, la función nos proporciona un vector en el cual se almacenan los índices de los puntos correspondientes a cada objeto encontrado. De modo que con este vector podremos extraer los objetos y volcarlos en una nueva nube libre de ruido y de otros objetos.

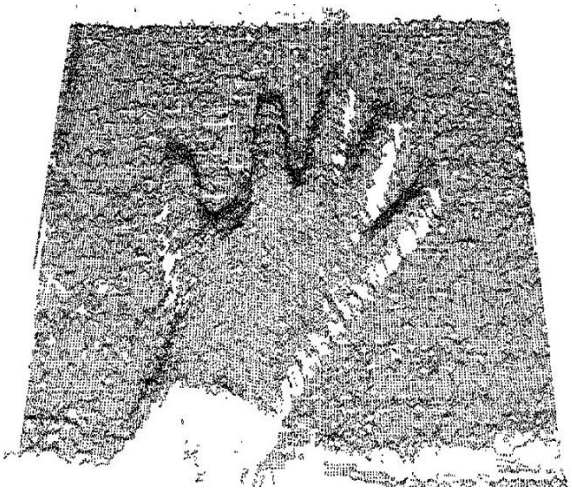

Ilustración 44 Escena original

El funcionamiento de esta se puede comprobar en las siguientes ilustraciones. Partimos de la situación representada en la Ilustración 44 y esta será la nube a partir de la cual se aplicará la función. A esta nube únicamente se le ha reducido su tamaño

mediante el filtro “VoxelGrid” explicado en el apartado 6.2.2, la razón de este paso previo es aliviar la carga computacional que supone la búsqueda de objetos. Antes del filtro la nube tenía 307200 puntos y después tenía 74424 puntos, el tamaño del filtro era 1.5x, 1.5y, 1.5z. Una vez realizada la reducción se ejecuta la función y a continuación se mostrarán los objetos encontrados.

Los parámetros proporcionados a la función son una tolerancia de 1.5, el tamaño máximo del objeto de 80000 puntos y el tamaño mínimo de 1000. Como se puede apreciar, el tamaño máximo es mayor que el tamaño de la propia nube lo que hace que jamás se limite un objeto por exceso de tamaño. Sin embargo, el tamaño mínimo sí que es muy importante ya que, si no se ajusta bien, podrían tomarse como objetos algunas de las agrupaciones de ruido, alterando por completo el buen funcionamiento del programa.

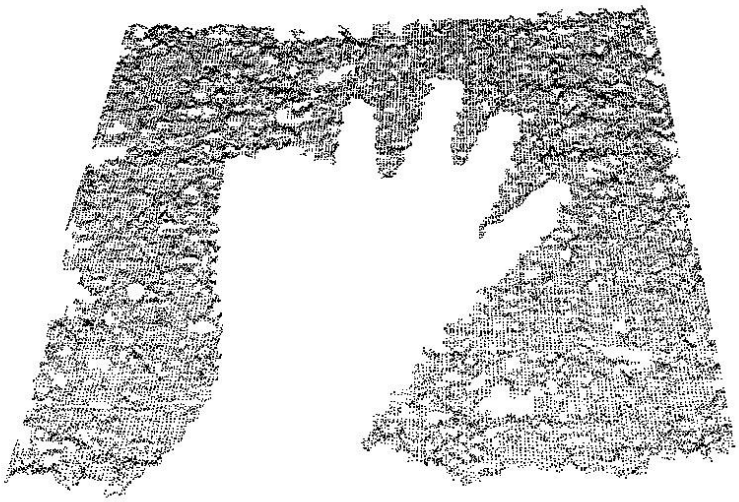

Ilustración 45 Objeto encontrado: Mesa

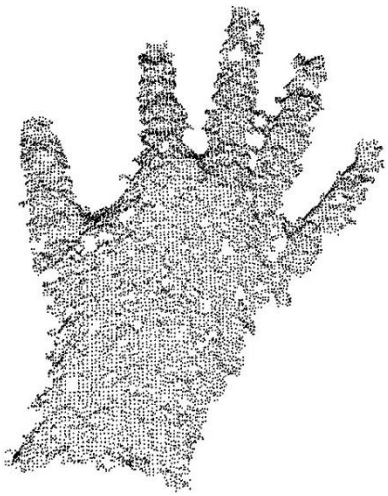

Ilustración 46 Objeto encontrado: Mano

Una vez encontrados los objetos, el problema está en saber distinguir cual es el que corresponde a la mano y cual al fondo. No es posible hacer una distinción por tamaños ya que estos variarán según la mano esté más alejada o más cerca de las cámaras o se vea mas zona del brazo o menos. Lo que se debe comprobar es la profundidad de los 2 objetos ya que es algo lógico e invariable que la mano siempre se va a encontrar por encima del fondo, por lo tanto, el procedimiento sería el siguiente:

- **No se encuentra ningún objeto**: No se contempla salvo algún error en el estéreo. Se deberán tomar las medidas oportunas.
- **Se encuentra un solo objeto**: La mano no se encuentra en la escena y solo se está encontrando la mesa, por lo tanto, no se debe hacer nada hasta que se reciba otra imagen.
- **Se encuentran 2 objetos**: Uno es la mano y otro el fondo, se comprueba la profundidad y el que menor valor tenga será el de la mano, se vuelcan los puntos correspondientes a este objeto en una nueva nube.
- **Se encuentran 3 o más objetos**: Indica que o bien se ha producido un error o se hay demasiado ruido en la imagen.

Los resultados obtenidos con esta técnica son realmente buenos y resulta ser la mejor en cuanto a eliminación de ruido de todas las que se han analizado, sin embargo, puede presentar algunos problemas de robustez y los tiempos de computación también son bastante altos.

6.2.6. Detección de dedos

Tras todos los procedimientos explicados anteriores, sea cual sea el camino seguido de todos los que se han planteado, se ha conseguido tener una nube de puntos en la cual se encuentran únicamente los puntos correspondientes de la mano. Ahora es el momento de tratar de determinar dónde están los dedos, su orientación y las puntas de los mismos.

- **Búsqueda de cilindros.**

En la búsqueda de un método para aislar cada dedo del resto de la mano para estudiarlo por separado, se pensó en la idea de que se podría estimar un dedo como un cilindro, debido a esto, se planteó este método.

Para llevarlo a cabo, se recurre a la misma función que se utiliza en el apartado 6.2.4 pero en este caso en lugar de buscar planos se buscarán cilindros. El cambio de modelo en la búsqueda altera ligeramente los parámetros que la función necesita para realizar su tarea correctamente y por ello se analizarán a continuación.

El método de búsqueda continúa teniendo las mismas opciones que antes, las cuales se pueden consultar en la documentación de PCL [15] en el apartado de “method_types.h”. Este parámetro determina el algoritmo de búsqueda que se utilizará para encontrar el modelo indicado. La elección de uno u otro puede

modificar considerablemente el tiempo empleado para la búsqueda y los resultados que la misma proporcione. Por ello se decide recurrir al mismo método de antes, RANSAC [13], debido a que es un método muy estudiado y con una gran garantía de funcionamiento.

Se continúa indicando el modelo que se quiere buscar que en este caso es un cilindro y aquí es donde residen las diferencias con la búsqueda de planos ya que para buscar un cilindro no vale con saber su forma, sino que también sus dimensiones. Por ello, debemos indicar el rango en el cual puede estar el diámetro del cilindro que en el caso de los dedos de la mano un rango lógico podría ser de 5mm a 15mm.

Por último, al igual que en el proceso anterior si indican el número máximo de iteraciones, la tolerancia para determinar un punto como perteneciente o no al modelo y el peso que tendrá el valor de la normal.

Además, al ejecutar la función al igual que antes nos devolverá el vector que almacena los puntos pertenecientes al cilindro, pero esta vez al no tratarse de un plano si no de un cilindro los parámetros que lo definen pasan de 4 a 7 y siguen la siente lógica:

- Los 3 primeros datos son las coordenadas XYZ que ubican un punto en el espacio.
- Los siguientes 3 datos definen la orientación de una recta que pasa por el punto anterior, la cual será el eje del cilindro.
- El último punto determina el radio del cilindro encontrado.

La gran cantidad de parámetros que se deben proporcionar a la función complica en gran medida la posibilidad de optimizar el valor de éstos para que el resultado sea el mejor posible. Realizar un cambio en uno de ellos puede suponer un aumento demasiado grande del tiempo de computación o variaciones en los resultados arrojados por la función. Aunque éste no será el principal problema al ejecutar este método tal y como se verá más adelante.

El transcurso del método es cíclico y consta de las siguientes fases:

- Cálculo de las normales
- Búsqueda del mejor cilindro
- Almacenamiento de los parámetros del cilindro encontrado
- Volcado de los puntos pertenecientes al cilindro a una nueva nube de puntos

Y este proceso se realizará tantas veces como dedos se desee encontrar. De modo que, en una situación ideal, tras 5 iteraciones, nos encontraríamos con una nube de puntos sin los dedos y 5 nubes de puntos que contengan cada una un dedo aislado, aparte de un vector en el que se abría almacenado la ubicación y orientación de cada dedo.

Sin embargo, existe un problema al ejecutar este ciclo. Y es que el número de dedos que las cámaras son capaces de ver varía mucho en una situación real, ya que, si se está moviendo la mano, los dedos pueden taparse unos a otros o que puede ser que éstos estén recogidos en la mano y por lo tanto no sean reconocidos como cilindros. Este problema se podría solucionar imponiendo una restricción de tamaño a los cilindros encontrados de manera que las iteraciones se detendrían si el cilindro encontrado fuese demasiado pequeño lo que indicaría que el cilindro encontrado no pertenece a un dedo correctamente estirado y así únicamente tendríamos la información de aquellos dedos visibles y viables de ser encontrados.

Pero en este razonamiento existe un problema que hace que el método sea totalmente inviable y es que, durante la búsqueda de cilindros, a pesar de ajustar lo mejor posible los parámetros de la función, los puntos correspondientes al resto de la mano influyen demasiado en ella. Haciendo que los cilindros encontrados no correspondan a los dedos si no a partes del brazo y de la mano. Esto se puede apreciar en la Ilustración 47, las cuales van en orden progresivo a media que el programa va encontrando cilindros.

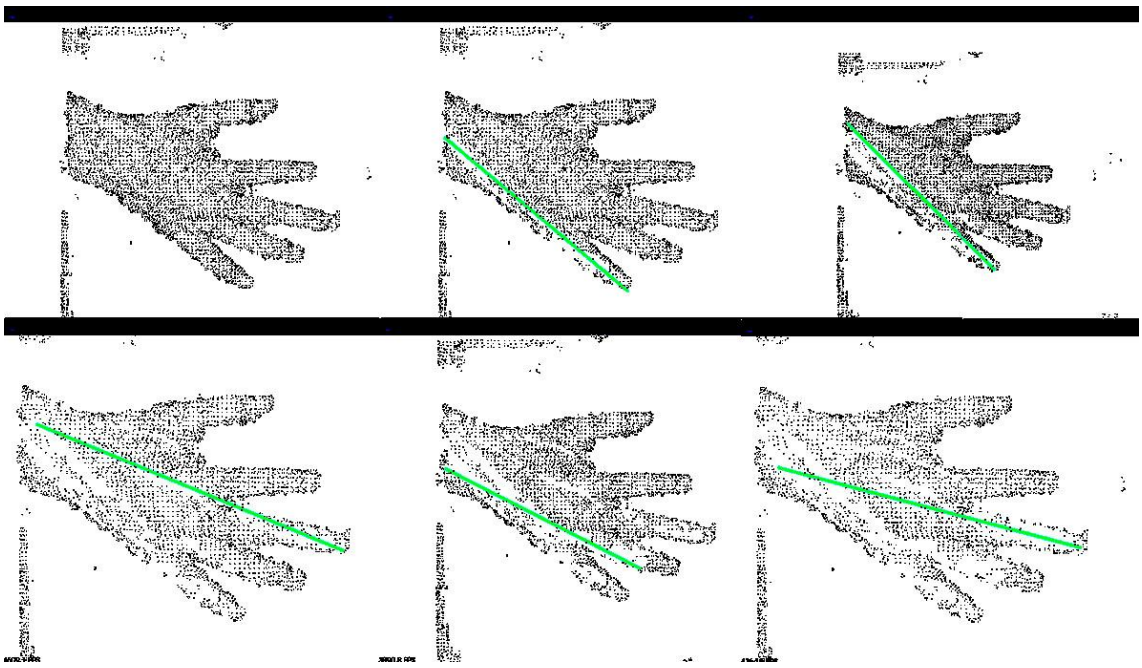

Ilustración 47 Búsqueda de cilindros en mano completa

El progreso de búsqueda va de izquierda a derecha y de arriba abajo. En las imágenes se ha remarcado el cilindro encontrado en cada uno de los pasos para facilitar su análisis. El principal problema es que la mano tiene demasiado peso en cuanto a puntos se refiere y altera por completo la búsqueda de cilindros en los dedos. En algunas posiciones en las que el dedo está alineado con la mano el programa sí que es capaz de encontrarlo como es el caso de esta posición con el dedo meñique, sin embargo, cuando el dedo está opuesto a la mano, como pasa con el pulgar, no se le considerará como un cilindro válido porque estará encontrando otros con mayor cantidad de puntos.

Este problema ha resultado imposible de solventar debido a que resulta muy difícil eliminar el resto de la mano y dejar los dedos sueltos mediante mecanismos de PCL, debido a esto esta ruta se consideró infructuosa y se comenzó a plantear otras posibles soluciones.

6.3. Desarrollo a partir del estéreo parcial por color

Debido a los infructuosos resultados conseguidos con los métodos anteriores, se decide retomar el problema con un nuevo punto de vista. En lugar de tomar todos los puntos de la escena para tener toda la información posible, se prueba a utilizar la herramienta que creó Carlos Castedo en [6] la cual consiste en aplicar un filtro de color en las imágenes antes de aplicar el estéreo para poder así eliminar toda información no relevante de la escena.

Este proceso tiene a priori las siguientes ventajas:

- Únicamente tendremos los puntos correspondientes a los dedos lo que facilitará la obtención de información de los mismos.
- La influencia de ruido se reducirá enormemente.
- La cantidad de puntos con la que se trabajará será mínima.

Mientras que en contra tenemos los siguientes puntos:

- Obligación de utilizar un guante adaptado para la aplicación del filtro de color.
- Los tiempos de computación para el estéreo serán algo superiores.
- Tener únicamente los dedos limitará la posible obtención de información adicional en el futuro.
- El filtro de color es muy sensible a cambios de iluminación.

Para este nuevo método se está empezando prácticamente desde el principio y muchas de las herramientas de PCL comentadas anteriormente no serán necesarias o útiles. Así, que se analizarán los problemas que se vayan presentando en el desarrollo de este método, así como las posibles soluciones y las ventajas e inconvenientes de cada una.

Comenzamos por lo tanto configurando el filtro de color, el funcionamiento de este filtro y de cómo se guardan los parámetros de color está detallado en [6]. Dado que el guante que se va a utilizar (ha sido analizado en el apartado 5.3.3) es de color negro y deja a la vista 3 dedos, el color que debemos introducir al programa para que haga el filtro es el color de la piel. Esto obviamente plantea el problema de que este color debe ser calibrado casi para cada persona debido a que cambios en la tonalidad de la piel pueden afectar en gran medida al resultado del filtro.

Una vez se ha calibrado el color se puede proceder a comprobar la ejecución del estéreo. En la Ilustración 48 se ve que efectivamente el mapa de disparidades calculado por el estéreo solo muestra los 3 dedos que el guante permite ver.

Ilustración 48 Mapa de disparidades de los dedos

Es momento ahora de proceder a escribir la información en un formato que PCL entienda y con el que pueda trabajar. Para ello se procederá de la misma forma que se hizo en el apartado 6.1. De nuevo se escoge un tipo de punto XYZ, ya que el estéreo no nos da valores ni de intensidad ni de color. También se modifica la escala de los datos pasando de décimas de milímetro ($10^{-4}m$) a milímetros ($10^{-3}m$). Por último, se comprueban las coordenadas de cada punto para determinar si debemos de introducirlo en la nube o no, ya que aquellas zonas de la imagen que han sido eliminadas por el filtro de color tendrán el valor “infinito” en sus coordenadas XYZ. Por lo tanto, solo se escribirán en la nube los puntos que tengan valores normales.

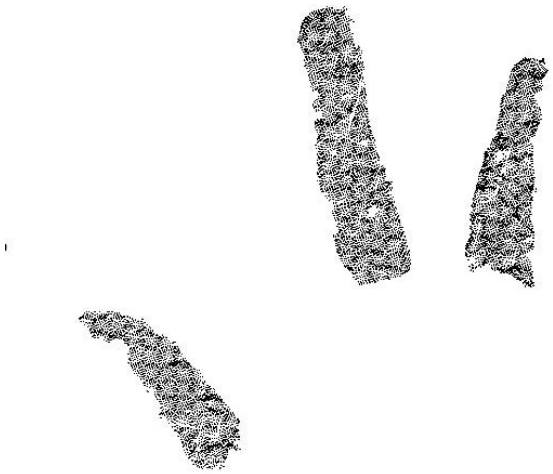

Ilustración 49 Nube de puntos de los dedos

6.3.1. Búsqueda de cilindros

Para este método se ha decidido volver a utilizar la búsqueda de cilindros para detectar los dedos. Esta decisión se toma debido a que, como ya se analizó en el método anterior, el problema por el que no funcionaba era que los puntos del resto de la mano influían demasiado en el resultado, sin embargo, en este momento solo tenemos los puntos correspondientes a los dedos por lo tanto el resultado que se obtenga debería ser mucho mejor.

Para efectuar la búsqueda de cilindros se recurre a la función de PCL "sac_segmentation" y para que funcione correctamente se debe calcular primero la dirección normal de cada punto y proporcionarle el peso que tendrán estas normales en el procedimiento, el tamaño máximo y mínimo del radio del cilindro, la tolerancia y el número máximo de iteraciones, además del método de búsqueda utilizado.

Todos estos datos son mantenidos del método anterior y son:

- Método de búsqueda: RANSAC.
- Peso de las normales: 0.3.
- Rango de valores para el radio: 5mm - 10mm.
- Tolerancia: 2.
- Número máximo de iteraciones: 1000.

Al igual que la vez anterior estos parámetros han sido ajustados por ensayo error, tratando de optimizar la calidad de los resultados obtenidos y el tiempo empleado para obtenerlos.

El proceso de búsqueda consiste en un bucle de acciones que se repetirá tantas veces como sea necesario hasta haber encontrado todos los dedos presentes en la imagen.

Comenzamos calculando las normales de la nube para después ejecutar la función de búsqueda de cilindros. Esta función nos devolverá un vector en el cual se encuentran los índices de los puntos estimados como cilindro y otra variable en la cual se encuentran los 7 parámetros que definen el cilindro encontrado. Toda esta información es muy importante para nuestra aplicación así que es fundamental almacenarla. Para ello se dispone de dos vectores, uno de ellos almacenará los parámetros de todos los cilindros encontrados y el otro almacenará las nubes de puntos correspondientes a cada dedo.

Además, una vez encontrado un cilindro resulta necesario eliminar de la nube original aquellos puntos que se han detectado en la búsqueda ya que si no son eliminados en la siguiente búsqueda volverán a ser los mismos los que vuelvan a ser considerados como el mejor cilindro. Una vez realizada la eliminación de los puntos se debe comenzar el bucle de nuevo.

Por último, es necesario imponer una condición de parada por la cual el ciclo de búsqueda se detiene. Por un lado, sabemos que el máximo de dedos que el

programa ha de encontrar son 3 ya que el guante solo deja al aire 3 dedos. Por lo tanto, el número máximo de iteraciones al bucle son 3. Pero, por otro lado, no siempre se podrán ver los 3 dedos y no siempre un dedo, aunque se vea podrá ser detectado como tal, si éste tiene una posición demasiado rara. Por ello se debe imponer una condición más para saber cuándo no quedan más dedos por detectar, esto lo conseguimos comprobando la cantidad de puntos que forman el cilindro encontrado, de este modo, cuando se encuentre un cilindro especialmente pequeño será un indicativo de que ya se han encontrado todos los existentes y que en ese momento solo hay ruido en la imagen por lo tanto habrá que detener el bucle.

A continuación, se muestra el resultado del proceso de búsqueda de cilindros, a diferencia de la vez anterior, el proceso funciona perfectamente y es capaz de detectar uno por uno los dedos presentes en la imagen, además, gracias a las condiciones de salida del bucle, este método es capaz de saber cuántos dedos hay en la imagen y de no gastar recursos cuando la mano no está y no hay nada que buscar.

Por último, gracias a haber almacenado los parámetros de cada cilindro, se tiene la posibilidad de realizar una representación en la cual se muestre todos los dedos juntos y el cilindro detectado en cada uno de ellos. Esta representación es para comprobar el correcto almacenamiento de los datos ya que, para la aplicación real, no serviría para nada.

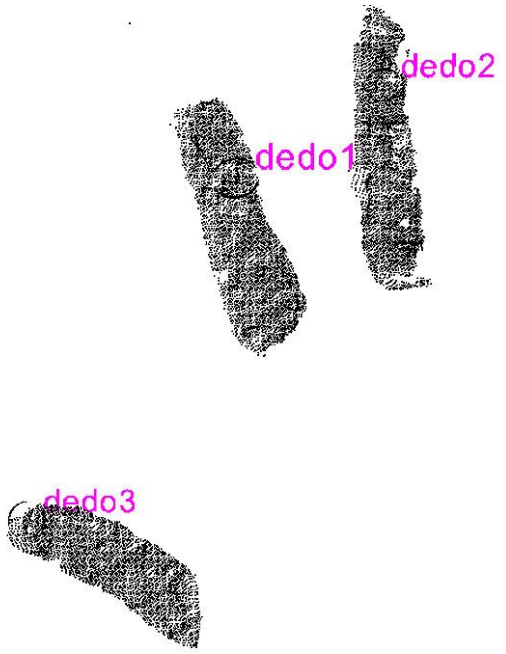

Ilustración 50Resultado de la búsqueda de cilindros

Como se puede observar en la Ilustración 50, la búsqueda resulta ser totalmente exitosa. Para facilitar la visualización de los resultados se ha añadido el texto para identificar a cada uno de los dedos. Es importante anotar que la numeración que da el programa a los dedos no sigue ningún orden específico, sino que solo hace referencia al orden en el que se han encontrado los cilindros. Por lo

tanto, tendrán números más bajos aquellos dedos que tiendan a parecerse más a un cilindro y tengan más puntos. Por eso, en el caso particular de la imagen mostrada, los dedos índice y corazón al ser más largos y grandes son los primeros en ser encontrados mientras que el pulgar resulta ser el último.

6.3.2. Búsqueda de extremos de dedos

Una vez se ha conseguido aislar cada uno de los dedos y se dispone de la orientación de los mismos, así como su posición, es momento de tratar de determinar cuáles son las posiciones de las puntas.

Determinar estas posiciones, aunque a priori puede resultar sencilla una vez se tiene el dedo aislado ya que bastaría con dar la posición del punto más alejado, resulta ser un problema debido a que al realizar el filtro de color se eliminó la información de dónde se encontraba la mano y por eso resulta muy complicado determinar qué extremo corresponde a la falange distal y cuál la falange proximal, es decir, cuando obtenemos uno de los 2 extremos del dedo no podemos saber si corresponde a la punta del mismo o al nudillo.

Para poder solucionar este problema existe la posibilidad de detectar, a partir de las orientaciones de todos los dedos dónde confluyen y en ese punto estaría la mano tal y como se ven en la primea imagen de la Ilustración 51. Sin embargo, este método resulta ser totalmente erróneo en determinadas posiciones de la mano debido a que el dedo pulgar puede oponerse al resto de la mano invalidando este método por completo. Además, se tendría un gran problema en el caso de solo detectar un dedo ya que sería imposible estimar la posición de la mano.

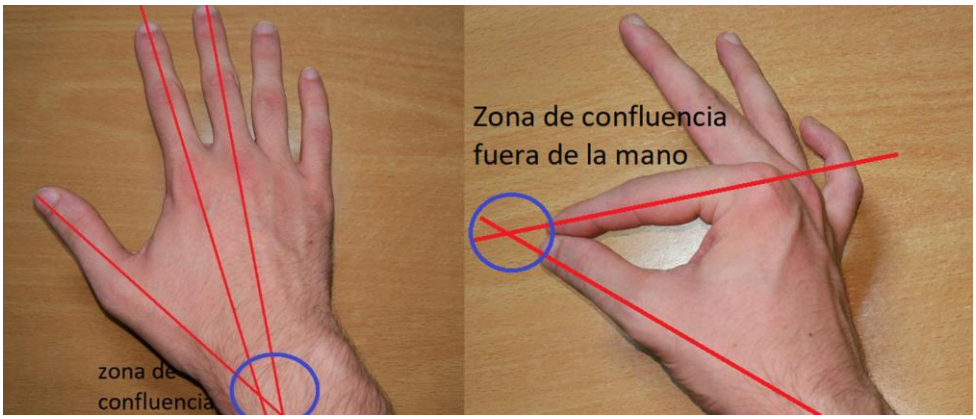

Ilustración 51 Zonas de confluencia de las direcciones de los dedos

Por lo tanto, es necesario recurrir a otro razonamiento para determinar cuál de los 2 extremos del dedo es el que corresponde a la punta. Finalmente, se decide asumir que la mano solo puede encontrarse en un rango de orientaciones de manera que la punta del dedo siempre será aquella que más coordenada Y tenga.

Este razonamiento puede parecer muy limitante y poco seguro ya que en el momento en el que la mano salga de ese rango de orientaciones los resultados obtenidos serían totalmente erróneos. Sin embargo, no se debe olvidar que se está

realizando una aplicación para una cirugía laparoscópica, es decir, que la mano que se está monitorizando se encuentra realizando una operación dentro de la barriga del paciente, por lo tanto, la capacidad de movimiento que tendrá el cirujano será muy limitada.

Gracias a esta suposición, es posible determinar las puntas de los dedos fácilmente, ya que nos bastará con estudiar la coordenada Y de todos los puntos de cada dedo y devolver la posición de aquel punto cuyo valor mayor sea. Existe la posibilidad de analizar también la coordenada X para aumentar ligeramente este rango de manera que se escogería el punto cuya suma de valor X e Y sea mayor. Por ejemplo, en el caso de la Ilustración 52 el punto elegido sería el azul ya que tiene mayor coordenada X e Y que el punto verde.

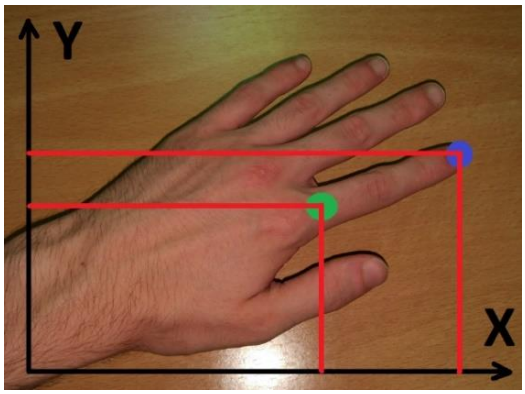

Ilustración 52 Ejemplo de búsqueda de la punta del dedo

Si pasamos este razonamiento al código y lo ejecutamos el resultado es el que se puede observar en Ilustración 53. Los puntos determinados como puntas han sido representados mediante esferas para facilitar su visualización. Además, tras varias pruebas con diferentes posiciones de la mano vemos que la búsqueda de los extremos de los dedos es muy rápida y efectiva dentro de los rangos de movimientos establecidos.

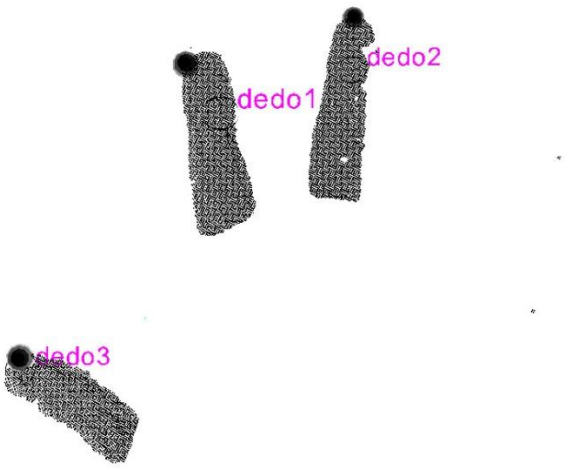

Ilustración 53 Dedos encontrados con sus puntas identificadas

6.4. Desarrollo del código

Después de buscar, probar y analizar las diferentes vías planteadas anteriormente para conseguir el objetivo planteado, se ha conseguido encontrar un método que lo consigue. Por lo tanto, es momento de realizar un código bien estructurado que implemente dicho método, que consiga realizar las tareas en el menor tiempo posible y con los mínimos recursos posibles.

Para ello, se deberá partir del código proporcionado por [6] y continuar la programación. A continuación, se irá detallando algunas de las cuestiones más importantes para conseguir el código final, el cual se encuentra en el anexo de este documento, Código .

6.4.1. Implementación de OpenCV y PCL en un solo proyecto

Realizar un proyecto que sea capaz de utilizar funciones tanto de OpenCV como de PCL independientemente no resulta algo sencillo de conseguir. Esto es debido a que entre las 2 librerías existen una serie de incompatibilidades que producen errores y fallos de compilación en el proyecto.

Una de estas incompatibilidades se da cuando en las rutas de compilación del compilador se incluyen las librerías de PCL y las de OpenCV2 que son una agrupación de librerías dentro de OpenCV. Esta incompatibilidad hace que cuando se trata de compilar el código, sea cual sea su contenido, ocurra un fallo durante el proceso y éste no consiga terminar. Debido a este problema es necesario que se eliminen de las rutas de compilación las librerías de OpenCV2 haciendo que algunas de las funciones implementadas en dichas librerías no puedan ser utilizadas en el proyecto, sin embargo, las funciones a las que se pierde el acceso no son utilizadas ni necesarias para la aplicación.

Otra de las incompatibilidades se da cuando se trata de mostrar por ventana una nube de puntos, mediante la libraría “pcl/visualization/cloud_viewer.h” de PCL, en un proyecto donde hay librerías activas de OpenCV. A diferencia del anterior esta incompatibilidad sí que puede entorpecer el desarrollo del código definitivo ya que hará más complicado la comprobación de los resultados obtenidos, sin embargo, en lo que se refiere al código final no afecta ya que lo que se desea obtener es el valor de las coordenadas de los puntos y no su representación por pantalla.

6.4.2. Ajuste de parámetros para el estéreo

Para la realización del estéreo, tal y como se ha argumentado anteriormente, se ha decidido utilizar el modo que aplica el filtro de color a las imágenes para que solo se muestren los puntos pertenecientes a los dedos.

Para que el proceso se lleve a cabo de manera correcta, son necesarios unos pasos previos:

- **Calibración de las cámaras**: La calibración de las cámaras sirve para diferentes cosas, por un lado, la mayoría de las cámaras presentan abombamientos en las imágenes debido a sus ópticas. Estas deformaciones de las imágenes se denominan comúnmente "ojo de pez" y hacen que las ubicaciones y formas de objetos se vean perturbadas en las imágenes obtenidas. La calibración consigue eliminar este abombamiento consiguiendo una mayor precisión en el estéreo. Pero, por otro lado, el calibrado sirve para poder determinar la colocación de las cámaras entre sí, tanto la distancia como las posibles desviaciones que pudieran tener en su alineación. De este modo, se consigue tener toda la información necesaria para llevar a cabo el proceso del estéreo correctamente.
  La calibración se lleva a cabo tomando imágenes del panel de calibración, apartado 5.3.4, en diferentes posiciones y ángulos. Comparando las deformaciones en todas las imágenes el algoritmo es capaz de determinar todos los valores necesarios para realizar las correcciones comentadas.
- **Calibración de los parámetros del estéreo**: Cuando se realiza el proceso de búsqueda de punto homólogos y cálculo de disparidades, es posible modificar los resultados obtenidos interviniendo en los parámetros que lo definen. Estos parámetros son muchos y no corresponde a este documento tratar cada uno de ellos, pero es importante entender que si no se ajustan correctamente el resultado obtenido puede ser totalmente erróneo. Estos parámetros se deberán variar en función de la distancia de las cámaras a la mano y al fondo, a la iluminación que exista en la escena, y las diferentes texturas que se estén captando.
- **Determinación del color**: Dado que se utiliza un filtro de color, resulta imprescindible indicar a la máquina qué color se desea filtrar. Para ello es necesario indicar el color o bien mediante valores RGB o bien HSV además de un rango y unas tolerancias. Tras diferentes pruebas, se ha decidido utilizar la metodología del RGB ya que ha resultado más robusta ante las condiciones lumínicas de las que se disponían.

Los resultados de las calibraciones y los valores de los parámetros que se han ajustado son almacenados en una carpeta dentro del proyecto de modo que cuando se ejecute el código principal, éste pueda acceder a la información de los calibrados y así ejecutar sus funciones correctamente.

6.4.3. Calibración de las funciones de PCL

En los apartados anteriores se ha explicado detalladamente todas las funciones que se han utilizado y las que se utilizarán en el programa final, sin embargo, resulta muy importante realizar un ajuste preciso de todos los valores de las funciones para que el resultado proporcionado por cada una de ellas sea el correcto. Para ello, se han ido probando una a una, comprobando tanto las entradas

como las salidas para poder así encontrar el valor óptimo y éstos han sido los resultados:

- **Almacenado de puntos en PCL**: Como se vio en el apartado 6.1, existe la posibilidad de modificar la escala en la cual se encuentran las coordenadas de los puntos. Sin embargo, para el código final se ha decidido mantenerla para no perder precisión en los valores almacenados, de este modo, las coordenadas con las que se trabajará estarán en décimas de milímetro (10^{-4})
- **Filtro de reducción de puntos (VoxelGrid Filter)**: Se realiza una reducción simétrica en los 3 ejes con un tamaño de plantilla de 2.0, esto significa que se reducirán los puntos que estén a una distancia menor de 0.2 milímetros entre sí.
- **Búsqueda de cilindros (Cylinder_Segmentation)**: Se buscan cilindros que tengan valores de radio entre 5mm y 7mm, la tolerancia admitida es de 0.8, los pesos de las direcciones normales son 0.5 y el número máximo de iteraciones es 200.

6.4.4. Implementación de hilos y multithreading

Uno de los principales problemas a la hora de implementar en un solo código toda la aplicación era que, a la hora de la ejecución, el algoritmo debía hacer demasiados cálculos y tareas para obtener la información de una sola imagen o "frame". Esto hacía que, si se deseaba ejecutar el programa en tiempo real, la frecuencia a la cual se devolvía la información de la mano era tremendamente baja además de que se iban acumulando retardos en las imágenes captadas y por lo tantos la información que suministrando estaba muy atrasada.

Para solventar este problema se planteó la utilización del multiprocesamiento o "multithreading" que consiste en hacer que la máquina realice tareas en paralelo para mejorar la eficiencia de los programas. El multiprocesamiento se basa en que los procesadores actuales están compuestos por diferentes núcleos, los cuales son independientes unos de otros a la hora de realizar cálculos, de este modo, hacen que uno o más núcleos del procesador se dediquen a una serie de tareas y otros a otras.

La implementación de hilos en C++ resulta relativamente sencilla gracias a la libraría <thread>. Ésta consigue generar, eliminar y coordinar diferentes hilos a los cuales se les encomendará la realización de una función del código, permitiendo que todos trabajen en paralelo.

En el caso concreto del código realizado se ha decido recurrir a la librería "boost/thread" ya que había sido necesario instalarla junto con PCL y ésta presenta muchas ventajas de rendimiento frente a la librería tradicional, sin embargo, el uso de la otra librería no modificaría el resultado final de la aplicación.

A pesar de la relativa facilidad que presenta C++ para incluir los hilos en el código, es indispensable incluir un sistema de control cuando se trabaja con hilos. El problema fundamental cuando hay diferentes hilos operando en un programa es que 2 hilos no pueden acceder al mismo tiempo a una zona de memoria, ni para escribir ni para leer, ya que si esto ocurre los datos leídos o escritos pueden verse corrompidos.

Para evitar este problema, C++ dispone de diferentes recursos para el control de acceso a recursos. Uno de estos es el semáforo y trabaja de la siguiente manera. Cuando se dispone de un recurso al que se le debe controlar el acceso se crea un semáforo, éste puede estar abierto o cerrado. De este modo se hace que cada función, proceso o hilo que deseen acceder al recurso comprueben el estado del semáforo, de manera que si está abierto se le permitirá acceder al recurso cerrando el semáforo y si éste ya está cerrado el proceso deberá esperar a que se habrá o continuar con otras tareas que no requieran de dicho recurso. Por último, es fundamental que cuando un proceso termina de utilizar el recurso libere el semáforo para que otro proceso puede acceder. Sin embargo, cuando se implementan semáforos se debe ser muy cuidadoso porque es realmente sencillo crear situaciones en las cuales el programa se queda colgado debido a que 2 hilos están esperando a que se libere un semáforo que el otro tiene bloqueado y viceversa.

Para nuestra aplicación se ha utilizado la libraría <mutex> la cual es un tipo de semáforo. Tiene la principal ventaja de que son muy simples de implementar y muy eficientes. Los mutex tienen 2 posiciones posibles, abiertos o cerrados y se controlan mediante las funciones “lock()” y “unlock()”.

Para implementar los hilos en el código es necesario primero analizar qué secciones de código deben de ejecutarse obligatoriamente en orden secuencial y cuáles no. Tras este análisis se puede concluir que hay 2 secciones claramente diferenciadas, por un lado, está la sección encargada de realizar el estéreo y por otro lado la que se encarga de generar la nube de puntos y estudiarla. Por lo tanto, lo lógico es implementar 2 hilos para que cada uno se encargue de una sección.

Además, es necesario también buscar aquellas variables o recursos a los que ambas secciones tienen que acceder, en este caso, es la imagen “ima3D” que es la imagen en formato “Mat” de OpenCV que almacena las coordenadas de los puntos obtenidos por el estéreo. Por lo tanto, tal y como se ha explicado anteriormente es necesario generar un Mutex para que controle el acceso a esa variable de manera que no se pueda acceder a ella desde los 2 hilos a la vez.

Por último, existe la necesidad de coordinar los 2 hilos, esto es debido a que para que el que generar las nubes de puntos pueda trabajar, antes se debe haber calculado un estéreo. Por eso, se genera un segundo mutex que, en lugar de controlar el acceso a un recurso, hará esperar al hilo de PCL según se comience a ejecutar el programa hasta que el estéreo haya generado al menos una imagen de coordenadas.

6.5. Estudio de viabilidad de cambio de cámaras por tecnología ToF

Durante todo el desarrollo de la aplicación se han utilizado las cámaras analizadas en el apartado 5.3.1. Éstas se tratan de cámaras web que fueron diseñadas y fabricadas para un uso totalmente distinto al que se le están dando en esta aplicación. Y esto se han notado claramente en muchas de las acciones que se han analizado anteriormente debido principalmente a los siguientes factores:

- Resolución y calidad de imagen bajas
- Predisposición a generar ruido en las imágenes
- Cambios en la iluminación afectan de diferente manera a una y otra a pesar de ser en principio iguales
- No disponen de drivers específicos para Linux/GNU

Debido a esto, durante el desarrollo de la aplicación se planteó la posibilidad de cambiar las cámaras por una cámara de tiempo de vuelo, 4.2.3. Este cambio tan radical tanto de cámara como de tecnología estaba basado en que realmente a pesar de modificar las cámaras por unas algo mejores seguirían existiendo algunos problemas asociados al estéreo:

- Sensible a cambios de iluminación o reflejos
- Precisión de los datos calculados por el estéreo relativamente baja
- Entre las 2 cámaras siempre existirán ligeras diferencias que afectarán al resultado final
- Necesidad de calibraciones relativamente complejas cada poco tiempo

La cámara que se planteó fue la que se analizó en el apartado 5.3.2, esta cámara es bastante compacta y está diseñada para trabajar en entornos industriales lo que hace que presente una gran robustez frente a ruidos e interferencias externas.

Se dispuso de la cámara durante algunas semanas y fue suficiente para poder analizar su funcionamiento en diferentes situaciones y ver si era factible la sustitución de las cámaras Logitech.

Para estudiar su funcionamiento se trataron de diferenciar 2 modos de trabajo, uno mediante el software propio de la empresa Ifm y otro en el que se trabajaría directamente con PCL. Sin embargo, para trabajar con PCL era necesaria la instalación de los drivers específicos para Linux y PCL y, debido al poco tiempo del que se dispuso de la cámara no fue posible la puesta en marcha de la misma en este ambiente.

En contra, utilizando el software propio de la cámara la facilidad de instalación y puesta en marcha son mucho mejores. El programa puede ser instalado en Windows y la cámara se conecta al ordenador vía Ethernet, por lo tanto, para que

ésta sea detectada por el ordenador en necesario ajustar las direcciones IP de los dispositivos.

Una vez realizadas la instalación y los ajustes de IP, el sistema está listo para capturar, almacenar y visualizar imágenes. Además, el software dispone una interfaz de usuario, Ilustración 54, que tiene una gran cantidad de parámetros fácilmente ajustables para modificar la obtención y representación de las imágenes.

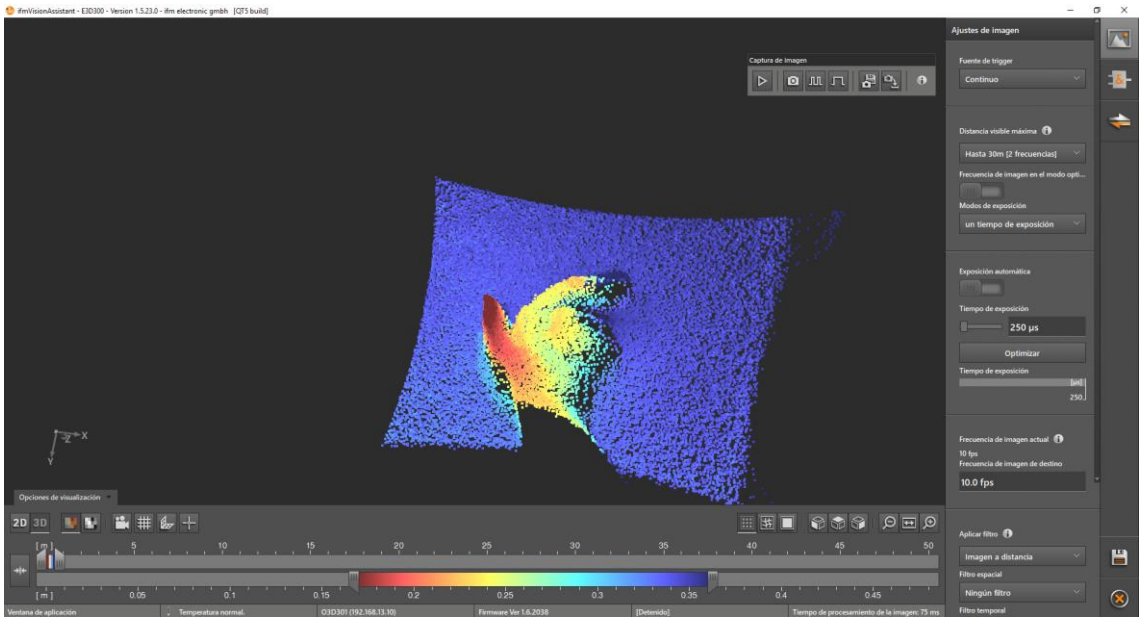

Ilustración 54 Interfaz de usuario del cámara

Por un lado, en la zona horizontal inferior se encuentran 2 barras con deslizables que nos ayudarán a modificar la representación de las imágenes, modificando las escalas a las cuales los colores que representan las distancias cambian. Gracias a esto se podrá ajustar la visualización según la escena esté más lejos y sea más amplia o a distancias más pequeñas y cerradas. También hay una serie de botones para modificar por ejemplo la representación en color o en blanco en negro, o en 3 dimensiones o en 2.

Por otro lado, en la zona lateral están los parámetros de captura de imágenes, en ella encontramos opciones como el tiempo de exposición, la cantidad de imágenes que se desean por segundo o frecuencia a la cual se emiten los pulsos. Este último parámetro es muy importante si hay más de una cámara en funcionamiento cerca ya que si 2 cámaras están trabajando a la misma frecuencia éstas pueden interferirse una a la otra y así empeorar significativamente los resultados obtenidos.

A continuación, se muestran algunas de las capturas que se realizaron con la cámara de escenas similares a las que se han tomado durante el desarrollo de la aplicación.

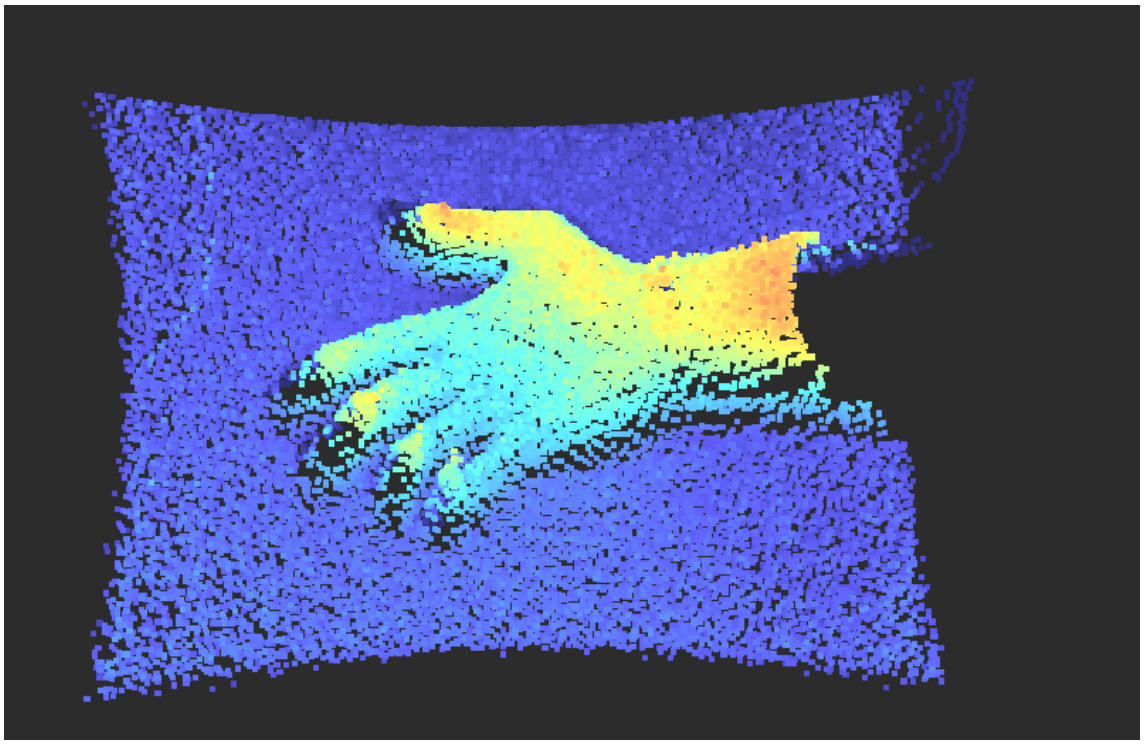

Ilustración 55 Captura imagen cámara ToF 1

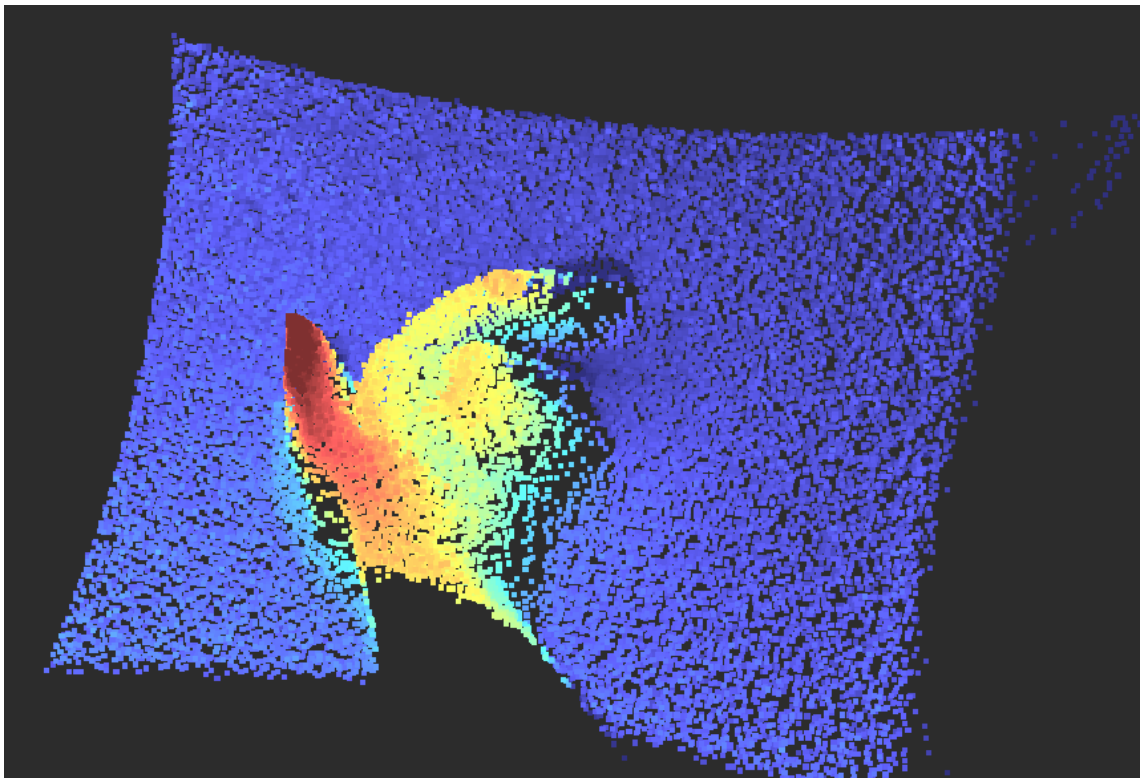

Ilustración 56 Captura imagen cámara ToF 2

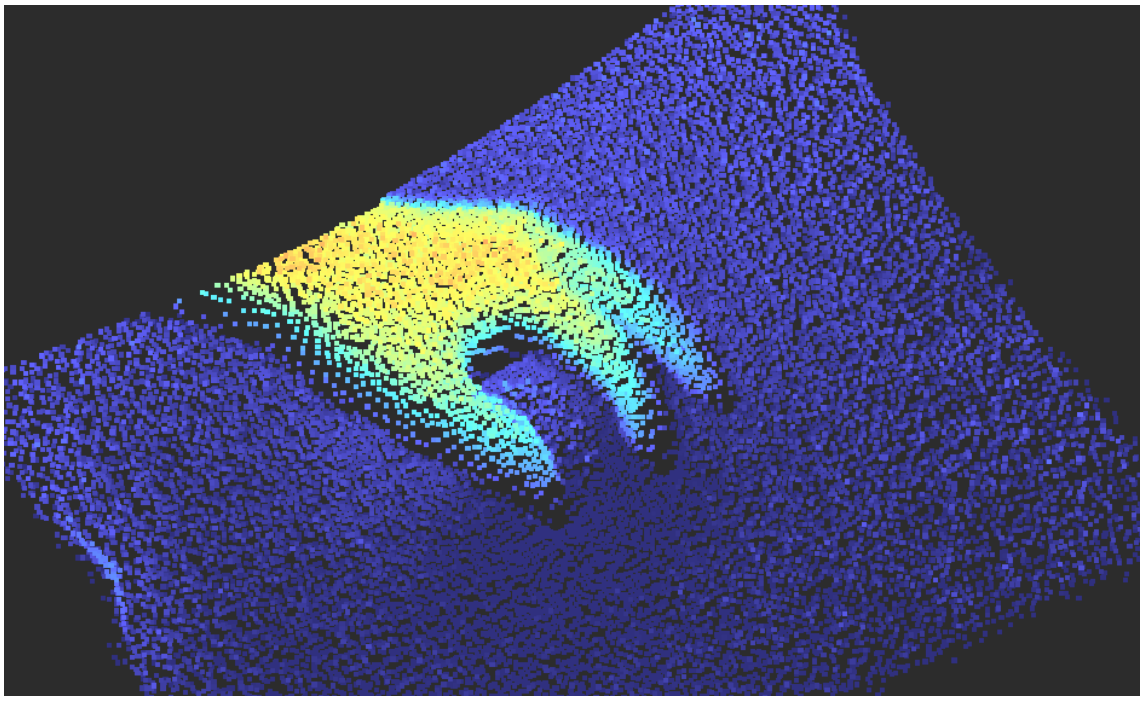

Ilustración 57 Captura imagen cámara ToF 3

Si las analizamos detenidamente y las comparamos cualquiera de las nubes de puntos que se han mostrado durante todo el documento se puede asegurar que los resultados obtenidos por la cámara ToF son mucho mejores que por el estéreo. Esto se puede asegurar debido a:

- La tecnología ToF presenta mayor precisión en las medidas de profundidad.
- El estéreo genera mucho ruido mientras que esta tecnología apenas tiene ruido en sus imágenes.
- Consigue determinar la profundidad en casi todas las superficies probadas mientras que el estéreo tiene problemas en las superficies más homogéneas.
- Las capturas anteriores se realizaron sin tener en cuenta la iluminación mientras que para la obtención de todas las imágenes con estéreo fue un aspecto que se debía controlar correctamente.
- La frecuencia máxima a la cual la cámara ToF puede suministrar imágenes es mayor que en el estéreo.
- El rango de distancias al cual la tecnología ToF trabaja es bastante más amplio al conseguido con las cámaras Logitech.

Sin embargo, esta cámara presenta también algunas desventajas:

- La cámara ToF tiene presenta problemas de sobrecalentamiento cuando trabaja en modo continuo.
- El tamaño de la cámara, a pesar de ser más compacta que muchas la mayoría de cámaras de su tecnología continúa siendo demasiado grande para una aplicación médica.

- La resolución y por tanto la cantidad de puntos conseguida por esta cámara es menor que la que consigue el estéreo.

Finalmente, tras las pocas pruebas que se pudieron realizar durante el tiempo que se tuvo la cámara, todo apunta a que salvando algunos de los problemas comentados, el cambio de las cámaras Logitech por la cámara Ifm O3D303 supondrían una clara mejora para la aplicación.

7. Análisis de resultados

En este apartado se analizarán los resultados obtenidos por las diferentes propuestas expuestas en el apartado anterior y se tratará de buscar los motivos por los cuales se consiguen dichos resultados y no otros.

7.1 Resultados a partir de la nube de puntos completa

Para este modo de trabajo ya se hizo un breve análisis de sus resultados, si recordamos del apartado 6.2, se debían realizar toda una serie de pasos previos para limpiar de ruido la imagen y obtener únicamente los puntos correspondientes a la mano. Una vez realizados estos pasos se continuaba buscando cilindros para tratar de encontrar la ubicación y orientación de los dedos.

Sin embargo, tal como se vio también en el apartado anterior los puntos pertenecientes al resto de la mano influían demasiado en el resultado final haciendo que los dedos no fuesen siempre encontrados correctamente.

A continuación, en la Ilustración 58, se muestra el resultado final de este proceso, este ejemplo representa bastante bien el comportamiento general de este método y es que consigue detectar aquellos dedos que están alineados con el resto de la mano y priorizando los más grandes (índice, corazón y anular).

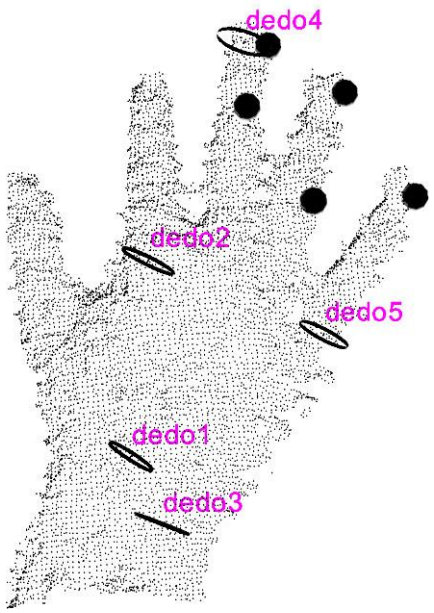

Ilustración 58 Búsqueda de cilindros en mano completa

Además, otro aspecto muy importante a analizar es el tiempo de computación que necesita el ordenador para llevar a cabo todos los cálculos. Estas pruebas han sido realizadas con un ordenador portátil de prestaciones medias, esto quiere decir que si fueran ejecutadas en una máquina más potente los tiempos se verías reducidos, pero con ellos podremos hacernos una idea del rendimiento del código.

Finalmente, tras muchas pruebas en diferentes posiciones y distancias los tiempos medios de computación por imagen son de aproximadamente 10 segundos. Obviamente, estos tiempos de computación son totalmente inadmisibles para una aplicación de este tipo en la que el objetivo es el guiado de un robot. La cantidad de información aportada es demasiado baja para que pueda realizar correctamente su tarea.

Los tiempos de computación tan altos se deben fundamentalmente a los siguientes factores:

- Lectura y trabajo con demasiados puntos.
- Demasiadas operaciones para aislar la mano.
- Búsqueda de cilindros con toda la mano demasiado lenta.

Cabe la posibilidad de mejorar el código, eliminando algunos pasos intermedios o sustituyéndolos por otros más eficientes, con el fin de reducir el tiempo de computación. Sin embargo, debido a que la calidad de los resultados tampoco es buena se ha decido que es rentable hacerlo ya que otros métodos pueden conseguir mejores resultados y menores tiempos de computación más fácilmente.

7.2. Resultados de la nube de puntos parcial.

Analizados los resultados arrojados por el método anterior se decidió buscar otra vía que consiguiera mejorar los resultados y, además, reducir los tiempos de computación sustancialmente. Y, tal y como se ha explicado en el apartado 6.3 se desarrolló este segundo método el cual hacía uso de un filtro de color para reducir la cantidad de puntos con los que se trabajaba.

Vamos por lo tanto ahora a estudiar los resultados aportados por este método y la calidad de los mismos.

A continuación, se muestran pares de imágenes con el mapa de disparidad de la escena calculado por el estéreo y los resultados obtenidos al poner a prueba el método en las diferentes posiciones de la mano. En las imágenes se muestran los cilindros encontrados y su numeración de por orden de búsqueda y las coordenadas de los puntos extremos detectados.

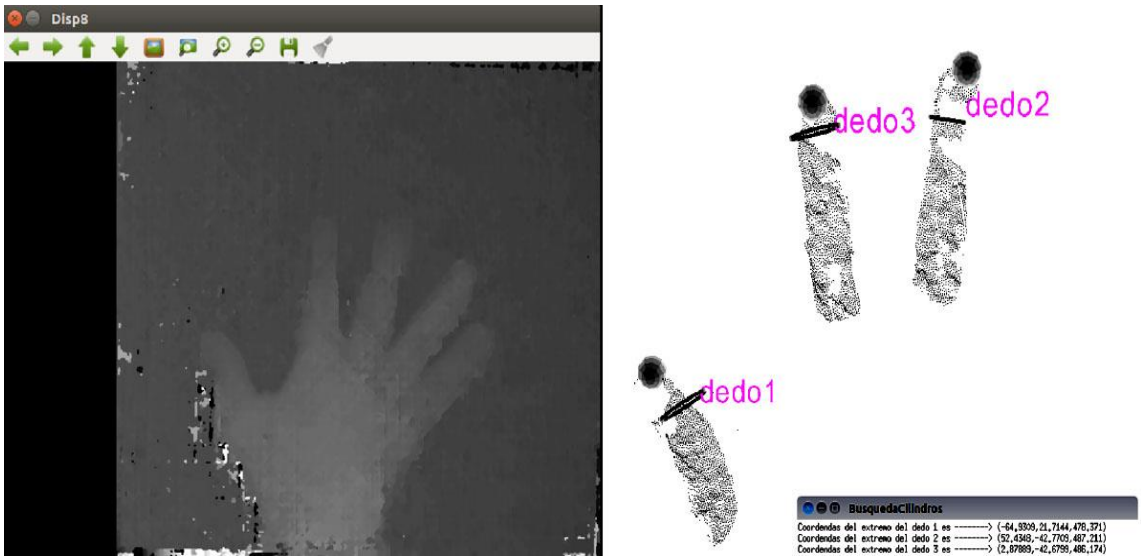

Ilustración 59 Resultados búsqueda 1

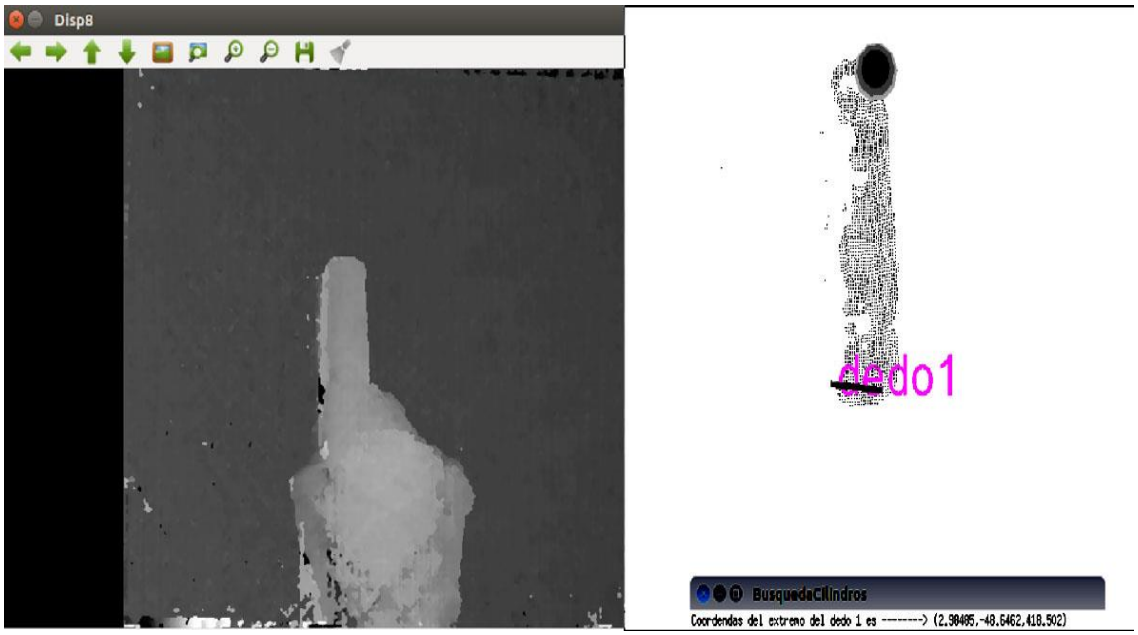

Ilustración 60 Resultado búsqueda 2

Si se analizan detenidamente las imágenes se puede ver que a pesar de que el mapa de disparidades obtenido por el estéreo no tiene toda la calidad que se desearía, la nube de puntos en PCL resulta ser muy aceptable ya que en ella no están las zonas de ruido del mapa de disparidades gracias al filtro de color.

También se puede ver que los resultados de búsqueda de dedos y de extremos son realmente buenos incluso para algunas situaciones complicadas como pueden ser las siguientes.

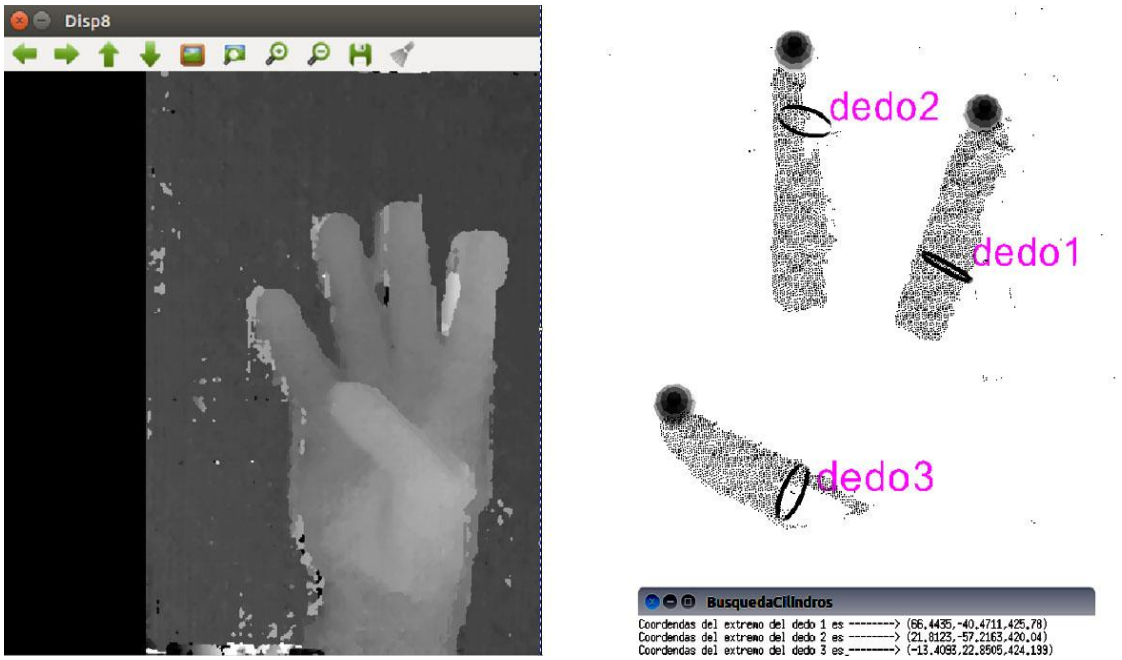

Ilustración 61 Resultado búsqueda 3

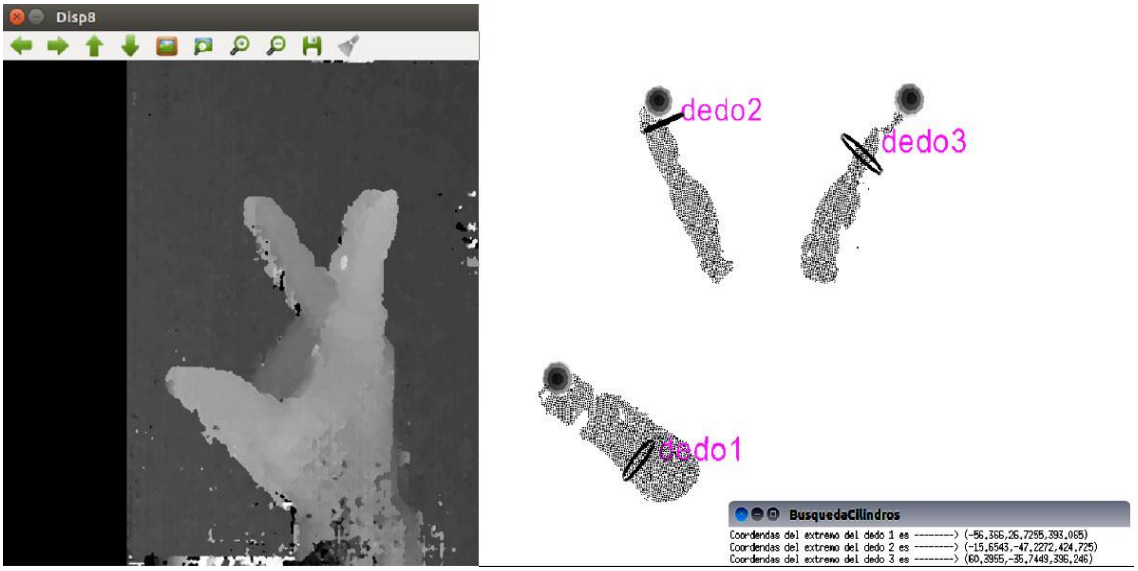

Ilustración 62 Resultado de búsqueda 4

Sin embargo, existen otras posiciones algo en las que a pesar de detectar correctamente los dedos y determinar su orientación, hay ligeras imprecisiones a la hora de determinar los extremos de los dedos como son el caso de las 2 situaciones siguientes.

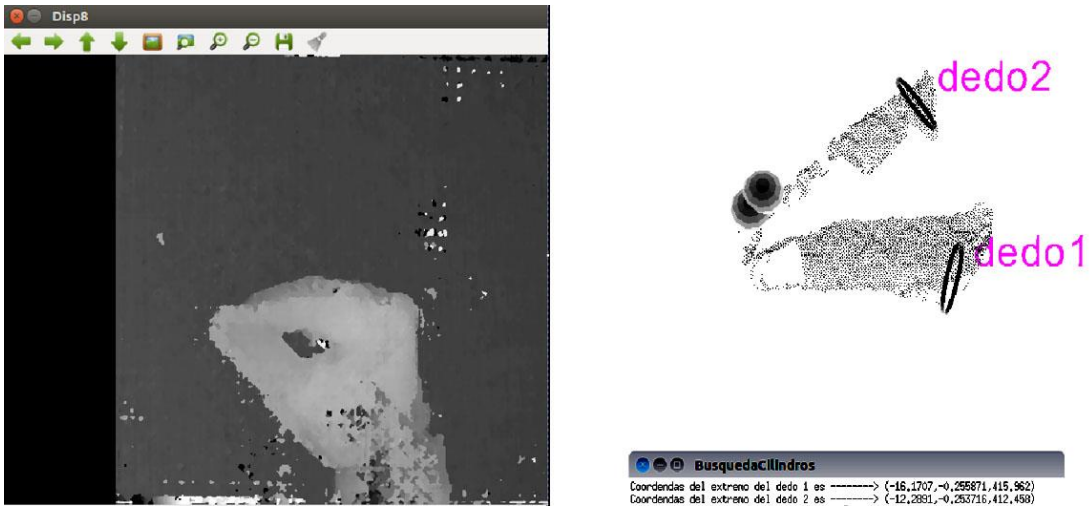

Ilustración 63 Resultado de búsqueda 5

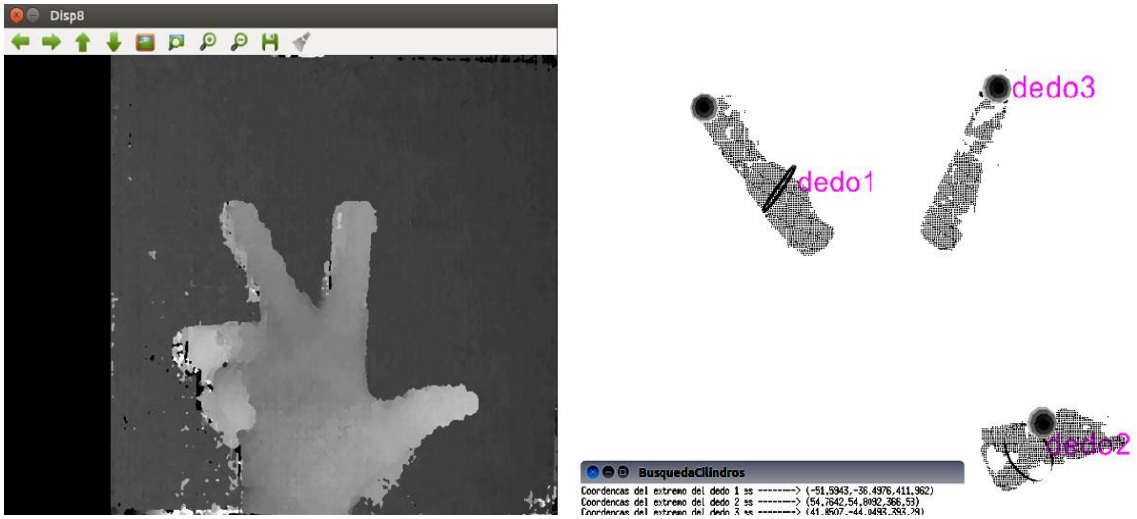

Ilustración 64 Resultado de búsqueda 6

Estos errores, aunque no son graves ya que la desviación que suponen frente al valor real no es muy grande, se deben a que se han superado los límites de orientación de la mano estimados en el apartado 6.3.2. El problema reside en que al realizar estas pruebas sin las restricciones que se tendrían durante una operación podemos rebasar dichos límites sin problemas.

Estas situaciones se muestran para ver qué pasaría en este tipo de casos en las que la mano adopta situaciones extremas, pero en una situación real muy difícilmente van a ocurrir. Aun así, se puede ver que el programa continuaría dando resultados mínimamente aceptables a pesar de todo.

Por otro lado, es necesario también estudiar la calidad de las medidas realizadas por el estéreo, ya que a pesar de que la ubicación de los extremos de los dedos y de las orientaciones de los mismos se realiza correctamente sobre la imagen, debemos saber que precisión presentan estos valores al llevarlos de vuelta a la realidad.

Como ya se ha explicado anteriormente, los valores de las coordenadas están en milímetros y el eje de coordenadas se presenta en el foco de la cámara izquierda, orientando el eje Z hacia la mano, y los ejes X e Y tal y como se ve en la Ilustración 65.

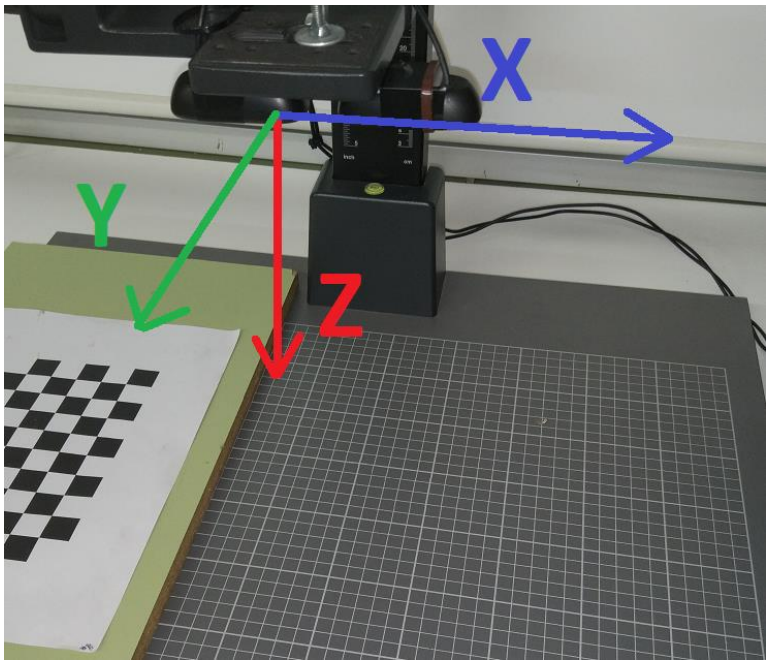

Ilustración 65 Sistema de ejes del estéreo

Estas mediciones y estimaciones de errores fueron realizadas y arrojaron un error relativo máximo del 1.5%, [6]. Esto quiere decir que, si la medida de coordenadas en la Z es de 450mm, el error máximo que el sistema puede haber cometido es de 6.75mm. Y esto es para la coordenada Z que es la que mayores valores toma ya que para la coordenada X e Y sus valores normales rondan en 100 y -100 por lo tanto los errores cometidos no superarán los 1.5mm

7.3. Rendimiento del código final

Las pruebas anteriores se han realizado en ejecuciones controladas, utilizando capturas previamente realizadas y dando el tiempo necesario a la máquina para funcionar. Pero una vez comprobado el correcto funcionamiento en esas circunstancias es momento de ejecutar el código final y comprobar su funcionamiento en tiempo real.

Para que el programa trabaje correctamente en tiempo real es totalmente imprescindible tener unas condiciones de iluminación buenas, en las que haya suficiente luz, pero ésta no genere excesivos reflejos ni sombras. Además, no se recomienda el uso de iluminación de fluorescentes ya que generan parpadeos en las imágenes que bajan la calidad de los resultados finales. Una vez la iluminación se ha establecido se debe proceder a realizar cuidadosamente las calibraciones tanto del estéreo como del color.

Se procede a poner en marcha el código y en el terminal del sistema se representará la cantidad de dedos que se han detectado y las coordenadas de sus extremos. Tal y como se muestra en Ilustración 66.

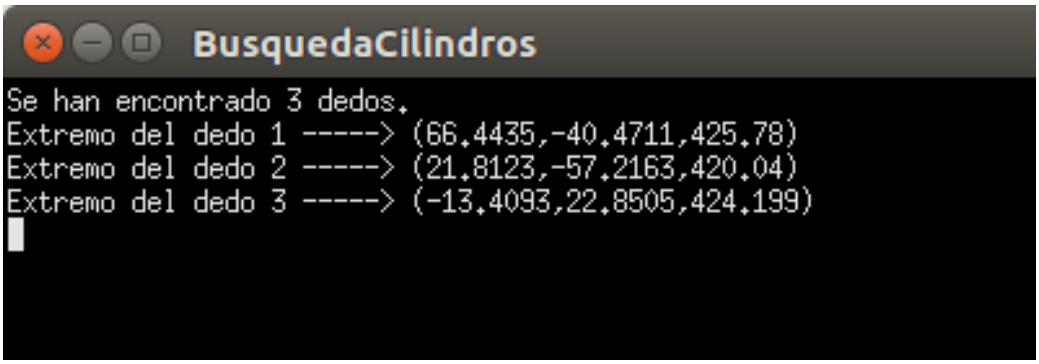

Ilustración 66 Terminal del código final

Los parámetros fundamentales a analizar de este código son, por un lado, la calidad de los resultados obtenidos, es decir, cómo de bien detecta los dedos y si hay situaciones en los que no los encuentra bien o detecta de más. Y, por otro lado, la eficiencia en cuanto a tiempo de computación y la frecuencia a la cual es capaz de analizar imágenes.

En cuanto a los resultados, su calidad es muy parecida a la que presentó la búsqueda estática analizada en el apartado anterior. Esto se debe a que el funcionamiento de búsqueda es exactamente el mismo y por lo tanto su comportamiento no tiene porqué variar a pesar de estar ejecutándose en tiempo real. Esto se traduce en que la máquina es capaz de determinar cuántos dedos hay y su ubicación en la gran mayoría de las posiciones y situaciones a excepción de momentos en los que se produce un movimiento rápido de la mano o de los dedos ya que las imágenes captadas no serán de la mejor calidad y por lo tanto el estéreo no será capaz de mantener su precisión. Sin embargo, estas situaciones de movimientos bruscos raramente se van a dar en una situación real y además son rápidamente solventadas en el momento en el que la mano vuelve de nuevo a su movimiento normal.

Por último, en cuanto a la velocidad de procesado los resultados son aceptables también. El único problema que presentan es que la frecuencia a la cual el sistema actualiza la información relativa a la mano varía dependiendo de la cantidad de puntos que el sistema tiene que analizar, es decir, en la situación en la que no está la mano, el sistema detecta rápidamente que no está y por lo tanto la frecuencia de muestreo aumenta significativamente, sin embargo, en las situaciones en las que se presentan los 3 dedos el sistema tarda algo más en determinar las posiciones de los 3 y la frecuencia cae.

Debido a esto, la velocidad de procesamiento varía desde unas 15 imágenes por segundo cuando la mano no se encuentra en la imagen hasta un mínimo de 3 por segundo cuando todos los dedos se muestran representados en las nubes de puntos.

La velocidad de ejecución de la última versión supone una mejora muy grande respecto a la del planteamiento inicial, la cual es 10 veces más lenta. Esto es debido a diversos motivos:

- Disminución considerable de la cantidad de puntos con los que se trabaja.
- Eliminación de pasos intermedios para reducción del ruido y acondicionamiento de la nube de puntos.
- Implementación de hilos.

8. Conclusiones

Durante el desarrollo de este Trabajo Fin de Grado se ha desarrollado un sistema de visión 3D destinado a ayudar y asistir al cirujano durante una intervención laparoscópica asistida por la mano. Para ello, se obtiene la tercera dimensión a partir de dos cámaras aplicando los conocimientos sobre estéreo aprendidos durante la carrera y, más tarde, convertir la información en nubes de puntos con las cuales pudiéramos trabajar y calcular toda la información necesaria.

Las herramientas empleadas han sido fundamentalmente OpenCV y PCL, las cuales nos permiten realizar una gran cantidad de operaciones y procesos sobre imágenes tanto en dos como en tres dimensiones. Estas herramientas han sido instaladas en Linux y utilizadas y compiladas mediante Code::Blocks. Todo este software, aunque complejo de implementar correctamente, forma un potente sistema de trabajo en el cual se ha podido efectuar la totalidad de la aplicación sin grandes problemas.

Se trataba de conseguir un programa que cumpliese los objetivos que se habían fijado inicialmente y que, además, permitiese el posterior desarrollo por parte de otros compañeros. Por ello, se ha continuado con el desarrollo modular del proyecto, agrupando las diferentes funciones y códigos en programas aislados que puedan ser ejecutados independientemente pero compartiendo todos recursos y configuraciones.

En cuanto a los resultados finales se puede decir que se han cumplido totalmente los objetivos, mediante el algoritmo generado, el programa es capaz de detectar correctamente los 3 dedos principales de la mano, y devolver la información referente a la orientación de los mismos y las coordenadas de sus extremos. Además, la velocidad de procesamiento conseguida se acerca mucho a la frecuencia de muestreo requerida por un robot para su guiado. Es preciso remarcar que la velocidad de ejecución podría aumentarse significativamente sin más que incrementar la potencia computacional sin un coste excesivo puesto que en el proyecto se ha empleado un ordenador portátil de gama media.

Por último, durante la realización del proyecto se han explorado otras líneas para el desarrollo futuro del proyecto, como por ejemplo la tecnología de tiempo de vuelo. Las pruebas realizadas han mostrado que su implementación podría ser muy beneficiosa para este sistema, ya que mejora sustancialmente el tiempo de procesado y la precisión de los resultados. Sin embargo, también se puede concluir que la estereovisión tiene un gran potencial, consiguiendo muy buenos resultados por un precio mucho menor al de cualquier otra tecnología 3D.

9. Líneas futuras

Una vez realizado trabajo y cumplidos los objetivos propuestos es momento de seguir buscando vías por las cuales continuar desarrollando el proyecto. Durante el proceso de desarrollo se han planteado diferentes problemas, de los cuales algunos han conseguido ser solventados pero otros no. De estos problemas no resueltos surgen la mayoría de las propuestas que se presentan a continuación.

- **Implementación de un sistema de iluminación controlado**: Tal y como se ha ido comentando durante todo el desarrollo del trabajo, la iluminación juega un papel crucial en la captura de imágenes y en la realización del estéreo. Debido a la ausencia de un sistema de iluminación adecuado nos hemos visto dependientes de la iluminación propia del lugar en el que se realizaban las pruebas, siendo éste muy poco recomendable para este tipo de aplicaciones de visión artificial.
- **Traspaso de Linux a Windows**: La decisión de en qué sistema operativo trabajar resulta muy complicada ya que de ésta dependerá tener unos problemas u otros durante el desarrollo de la aplicación. Además, una vez tomada, resultará realmente complicado y costoso revertirla. Para la realización de este trabajo se decidió utilizar Linux, más concretamente la distribución de Ubuntu, debido a las ventajas que presentaba frente Windows. Éstas eran principalmente una instalación más sencilla de OpenCV y PCL. Sin embargo, una vez realizadas se puede confirmar estas ventajas no eran tan significativas. Por otro lado, las desventajas propias de trabajar con Linux, como la necesidad de su instalación en una partición del disco o en una máquina virtual, o el trabajar vía terminal, seguían estando ahí. Por estos motivos, se recomienda el uso de Windows para futuros trabajos en este ámbito.

En cuanto a lo que se refiere al código y a posibles mejoras en él se propone lo siguiente:

- **Evitar la utilización del filtro por color trabajando con toda la mano**: Tal y como se ha explicado en este documento, la utilización del filtro de color ha sido necesaria para conseguir alcanzar los objetivos propuestos, sin embargo, tomar esta decisión ha supuesto la aparición de otros problemas. Para implementaciones futuras, se recomienda el uso de la nube de puntos correspondiente a toda la mano para así poder obtener la mayor información posible y evitar todos los problemas derivados del uso del filtro de color. Para ello, se propone la utilización

de la herramienta de PCL “NARF KeyPoints” la cual consiste en realizar un estudio de la nube de puntos para así obtener aquellos puntos más relevantes de la misma y poder así obtener la información de manera rápida y robusta. De este modo, además, se conseguiría evitar la utilización del guante.

- **Combinación del sistema de visión con un guante inteligente**: Los guantes inteligentes son guantes que son capaces de determinar la posición de la mano gracias a una serie de sensores dispuestos en ellos. Se plantea la posibilidad de implantar el uso de uno de estos guantes junto con el sistema de visión desarrollado y así combinando la información aportada por los dos sistemas se conseguiría un nivel de precisión y robustez mucho mayor.
- **Reconocimiento gestual**: Esta propuesta surge de la base de que el objetivo del robot quirúrgico es ayudar y asistir al cirujano durante las intervenciones y para ello se plantea la posibilidad de implantar un sistema de reconocimiento de gestos por el cual el robot sabría qué es lo que el cirujano desea que haga sin necesidad de interfaces. El reconocimiento gestual podría partir de algo sencillo, pudiendo reconocer unas pocas órdenes simples para más tarde ir mejorando el sistema y que pudiese reconocer órdenes más complejas.

Bibliografía

[1] *S. A. Antoniou, G. A. Antoniou, A. I. Antoniou y F.-A. Granderath, «Past, Present, and Future of Minimally Invasive Abdominal Surgery,» Journal of the Society of Laparoendoscopic Surgeons, vol. 19, nº 3, 2015.*

[2] *P. Ricci , R. Lema , V. Solà, J. Pardo y E. Guiloff , «Desarrollo de la cirugía laparoscópica: Pasado, presente y futuro. Desde Hipócrates hasta la introducción de la robótica en laparoscopia ginecológica,» Rev Chil Obstet Ginecol, vol. 73, nº 1, pp. 63-75, 2008.*

[3] *L. Grzegorz S. , «Kurt Semm and the Fight against Skepticism: Endoscopic Hemostasis, Laparoscopic Appendectomy, and Semm's Impact on the "Laparoscopic Revolution",» Journal of the Society of the Laparoendoscopic Surgeons, vol. 2, nº 3, pp. 309-313, 1998.*

[4] *R. Hartley y A. Zisserman, Multiple View Geometry in Computer Vision, Cambridge University Press, 2004.*

[5] *P. Gil, T. Kisler, G. J. García, C. A. Jara y J. A. Corrales, «Calibración de cámaras de tiempo de vuelo: Ajuste adaptativo del tiempo de integración y análisis de la frecuencia de modulación,» Revista Iberoamericana de Automática e Informática industrial, nº 10, pp. 453-464, 2013.*

[6] *C. Castedo Hernández, «Sistema de visión estereocópico para e guiado de una robot quirurgico en operaciones de cirugía laparoscópica HALS,» Valladolid, 2017.*

[7] *«Ubuntu,» Canonical Group Ltd, [En línea]. Available: https://www.ubuntu.com/.*

[8] *«OpenCV,» [En línea]. Available: http://opencv.org/.*

[9] *«Point Cloud Library,» [En línea]. Available: http://pointclouds.org/.*

[10] *«PCL Documentation,» [En línea]. Available: http://pointclouds.org/documentation/.*

[11] *«Ifm Electronic S.L.,» [En línea]. Available: https://www.ifm.com/ifmes/web/home.htm.*

[12] *«Code::Blocks IDE,» [En línea]. Available: http://www.codeblocks.org/.*

[13] M. A. Fischler y R. C. Bolles, «Random sample consensus,» Communications of the ACM, vol. 24, nº 6, pp. 381-395, 1981.

[14] F. P. Miller, A. F. Vandome y J. McBrewster, Kd-Tree, VDM Publishing, 2009.

[15] «PCL Documentation,» [En línea]. Available: http://pointclouds.org/documentation/.

Anexos

Código

```
/*********************************************************************************************/
/************************TRABAJO DE FIN DE GRADO**********************************************/
/**INTEGRACIÓN DE UN SISTEMA DE VISIÓN ESTEREOSCÓPICO EN UN ENTORNO DE CIRUGÍA LAPAROSCÓPICA**/
/*************GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA************************/
/*********************************************************************************************/
/*****************************RAFAEL ESTOP REMACHA*********************************************/
/******************Tutor del proyecto: EUSEBIO DE LA FUENTE LÓPEZ******************************/
/*********************************************************************************************/
/**El presente código ha sido realizado a partir del código de Carlos Castedo que se *********/
 /**encuentra en su trabajo fin de Grado ******************************************************/
 /*********************************************************************************************/

 ///Para la correcta compilación y ejecución de este código es necesario la instalación tanto
 ///de OpenCV como de PCL.

 ///El código ha sido divido en 2 secciones las cuales se ejecutan paralelamente a partir de hilos.

 ///Para el correcto funcionamiento del programa, el código hace uso de la información que hay
 ///en las carpetas "Data" e "Include" ubicadas en el directorio del proyecto.

 ///Antes de ejecutar el código es necesario realizar previamente la calibración de las cámaras,
 ///del estéreo y del color. Para ello se encuentran los programas en el direcctorio del proyecto


 #include <stdio.h>
 #include <stdlib.h>
 #include <time.h>
 #include <string.h>
 #include <iostream>
 #include <vector>
 #include <string>
 #include <boost/thread.hpp>
 #include <mutex>
 #include "../Include/config.h"

 #include "opencv2/core/core.hpp"
 #include "opencv2/calib3d/calib3d.hpp"
 #include <opencv2/highgui/highgui.hpp>
 #include <opencv2/imgproc/imgproc.hpp>

 #include <pcl/io/pcd_io.h>
 #include <pcl/point_types.h>
 #include <pcl/ModelCoefficients.h>
 #include <pcl/io/pcd_io.h>
 #include <pcl/point_types.h>
 #include <pcl/filters/extract_indices.h>
 #include <pcl/filters/passthrough.h>
 #include <pcl/features/normal_3d.h>
 #include <pcl/sample_consensus/method_types.h>
 #include <pcl/sample_consensus/model_types.h>
 #include <pcl/segmentation/sac_segmentation.h>
 ///#include <pcl/visualization/cloud_viewer.h> /**Problemas al utilizarla con OpenCV**/
 #include <pcl/filters/voxel_grid.h>

 using namespace std;
 using namespace cv;

 void estereo();
 void busqueda();

 ///Definición del tipo de punto utilizado por las nubes de puntos.
 typedef pcl::PointXYZ PointT;

 ///Declaración de las variables globales a las que acceden los 2 hilos.
 Mat ima3D;
 unsigned dedos_encontrados;

 ///Declaración de los mutex que controlan el acceso a las variables globales
 std::mutex mutex_ima3D;
 std::mutex mutex_coordinacion;

```

```
72 int main()
73 {
74
75       boost::thread hilo_estereo (estereo);
76       boost::thread hilo_busqueda (busqueda);
77
78       hilo_estereo.join(); ///En el caso de que el hilo 1 termine su ejecución debido a un error
         en la apertura de los ficheros de
79       ///calibración o de las cámaras se finaliza el programa sin esperar al hilo 2.
80
81       return 0;
82 }
83       ///Función llamada por el hilo 1, lee lo ficheros necesarios para la calibración y
         realiza el estéreo en bucle guardando la información en la variable globar Ima3D.
84 void estereo()
85 {
86       mutex_coordinacion.lock();
87       /// Lectura fichero calibracion
88       Mat map_r1, map_r2; //pixel maps para x e y para rectificar imagen Dcha
89        Mat map_l1, map_l2; //pixel maps para x e y para rectificar imagen Dcha
90       Mat Q; //necesaria para remapeo 3D
91
92       bool salida=false;
93
94       string fileCalib = "../Data/Calibracion.yml";
95       FileStorage fs_cal(fileCalib, FileStorage::READ);
96       fs_cal["Q"] >> Q;
97       fs_cal["map_l1"] >> map_l1;
98       fs_cal["map_l2"] >> map_l2;
99       fs_cal["map_r1"] >> map_r1;
100      fs_cal["map_r2"] >> map_r2;
101
102      fs_cal.release();
103
104       // Comprobacion lectura valores
105      if(map_r1.empty() || map_r2.empty() || map_l1.empty() || map_l2.empty()) ///En caso de error
         en la apertura de los ficheros de calibración se finaliza el hilo sin tomar imágenes.
106      {
107              cout << "ERROR al cargar fichero de calibracion" << endl;
108              salida=true;
109      }
110
111      /// Lectura fichero rango colores
112      int hH,hS,hV,lH,lS,lV,hR,hG,hB,lR,lG,lB;
113      bool modeHSV = true;
114      string fileFiltro = "../Data/ColorDetecta.xml";
115      FileStorage fs_fil(fileFiltro, FileStorage::READ);
116      fs_fil["hH"] >> hH;
117      fs_fil["hS"] >> hS;
118      fs_fil["hV"] >> hV;
119      fs_fil["lH"] >> lH;
120      fs_fil["lS"] >> lS;
121      fs_fil["lV"] >> lV;
122      fs_fil["hR"] >> hR;
123      fs_fil["hG"] >> hG;
124      fs_fil["hB"] >> hB;
125      fs_fil["lR"] >> lR;
126      fs_fil["lG"] >> lG;
127      fs_fil["lB"] >> lB;
128      fs_fil.release();
129
130      if(hH == 0 && hS == 0 && hV == 0 && lH == 0 && lS == 0 && lV == 0){
131              modeHSV = false;
132      }
133
134      /// Lectura fichero parametros stereoSGBM
135      int minDisparity, numDisparity, SADWindowSize, P1, P2, disp12MaxDiff, preFilterCap,
         uniquenessRatio,
         speckleWindow, speckleRange;
136
137      string fileSGBM = "../Data/ParametrosStereo.xml";
138      FileStorage fs_BM(fileSGBM, FileStorage::READ);
139
140      fs_BM["minDisparity"] >> minDisparity;
141      fs_BM["numDisparity"] >> numDisparity;
142      fs_BM["SADWindowSize"] >> SADWindowSize;
143      fs_BM["P1"] >> P1;
144      fs_BM["P2"] >> P2;
145      fs_BM["disp12MaxDiff"] >> disp12MaxDiff;
146      fs_BM["preFilterCap"] >> preFilterCap;
147      fs_BM["uniquenessRatio"] >> uniquenessRatio;
148      fs_BM["speckleWindow"] >> speckleWindow;
149      fs_BM["speckleRange"] >> speckleRange;
150      fs_BM.release();
151
152       ///Setup adquisicion imagen
153      VideoCapture capIzda(1), capDcha(2);
154      capIzda.set(CV_CAP_PROP_FRAME_HEIGHT, FRAME_HEIGHT);
```

```
155     capIzda.set(CV_CAP_PROP_FRAME_WIDTH, FRAME_WIDTH);
156     capDcha.set(CV_CAP_PROP_FRAME_HEIGHT, FRAME_HEIGHT);
157     capDcha.set(CV_CAP_PROP_FRAME_WIDTH, FRAME_WIDTH);
158
159     Mat element = getStructuringElement( MORPH_ELLIPSE, Size(TAM_STREL,TAM_STREL));
160
161     while(!salida) ///Bucle infinito mientras todo funcione correctamente.
162     {
163             /// Captura de imagenes
164             capIzda.grab();
165             capDcha.grab();
166
167             Mat frameIzda, frameIzdaRect, frameDcha, frameDchaRect;
168                     capIzda.retrieve(frameIzda);
169             capDcha.retrieve(frameDcha);
170
171             if(frameIzda.empty() || frameDcha.empty()){ ///En caso de error en la toma de
                imágenes se sale del bucle y finaliza el hilo.
172                     cout << "Error en camaras" << endl;
173                     salida=true;
174             }
175
176             flip(frameDcha,frameDcha,-1); ///Se realiza un giro en una imagen debido a que una
                cámara se encuentra girada para reducir la distancia entre objetivos.
177
178              /// Rectificado
179             remap(frameIzda, frameIzdaRect, map_l1, map_l2, INTER_LINEAR);
180             remap(frameDcha, frameDchaRect, map_r1, map_r2, INTER_LINEAR);
181
182             /// Color
183             Mat izqmascara, dchamascara, izqsalida, dchasalida;
184             if(modeHSV){
185                     /// Conversion de RGB a HSV
186                     Mat modelColor_izda, modelColor_dcha;
187                     cvtColor(frameIzdaRect,modelColor_izda,COLOR_BGR2HSV); ///Pasa izda a HSV
188                     cvtColor(frameDchaRect,modelColor_dcha,COLOR_BGR2HSV); ///Pasa dcha a HSV
189
190                     /// Distincion colores
191                     inRange(modelColor_izda,Scalar(lH,lS,lV),Scalar(hH,hS,hV),izqmascara);
192                     inRange(modelColor_dcha,Scalar(lH,lS,lV),Scalar(hH,hS,hV),dchamascara);
193             }else{
194                     /// Distincion colores
195                     inRange(frameIzdaRect,Scalar(lB,lG,lR),Scalar(hB,hG,hR),izqmascara);
196                     inRange(frameDchaRect,Scalar(lB,lG,lR),Scalar(hB,hG,hR),dchamascara);
197             }
198
199             /// StereoSGBM
200             Mat disp, disp8, disp_compute; /// Declaracion imagenes StereoSGBM
201
202             /// Declaracion y computacion de stereoSGBM
203             Ptr<StereoSGBM> sgbm =
                StereoSGBM::create(minDisparity,numDisparity*16,SADWindowSize,P1,P2,
                disp12MaxDiff,preFilterCap,uniquenessRatio,speckleWindow,speckleRange,true);
204             sgbm->compute(frameIzdaRect, frameDchaRect, disp);
205             normalize(disp, disp8, 0, 255, CV_MINMAX, CV_8U);
206
207             Mat mapaDispFil;
208             disp.copyTo(mapaDispFil,izqmascara);
209             mapaDispFil.convertTo(disp_compute,CV_32F, 1.f/16.f);
210
211             mutex_ima3D.lock(); ///Se espera a que el mutex esté libre para poder acceder a la
                variable global.
212
213             /// Imagen 3D
214             reprojectImageTo3D(disp_compute,ima3D,Q,true);
215
216             mutex_ima3D.unlock();
217             mutex_coordinacion.unlock();
218     }
219
220     capDcha.release();
221     capIzda.release();
222 }
223     ///Función llamada por el hilo 2, espera a que el hilo 1 haya realizado una iteración y
        comienza a buscar cilindros y sus extremos a partir de la matriz "Ima3D"
224 void busqueda()
225 {
226     ///Creación de los objetos que se utilizarán durante la ejecución de la función.
227     pcl::PointCloud<PointT>::Ptr cloud (new pcl::PointCloud<PointT>);
228     pcl::PointCloud<PointT>::Ptr cloud_original (new pcl::PointCloud<PointT>);
229     pcl::PointCloud<pcl::Normal>::Ptr cloud_normals (new pcl::PointCloud<pcl::Normal>);
230     pcl::ModelCoefficients coefficients_cylinder;
231     pcl::PointIndices::Ptr inliers_cylinder (new pcl::PointIndices);
232     pcl::PointCloud<PointT>::Ptr cloud_cylinder (new pcl::PointCloud<PointT> ());
233     std::vector<pcl::ModelCoefficients> coeficientes;
234     std::vector<PointT> extremo_dedo;
235     pcl::NormalEstimation<PointT, pcl::Normal> ne;
236     pcl::SACSegmentationFromNormals<PointT, pcl::Normal> seg;
```

```
237     pcl::ExtractIndices<PointT> extract;
238     pcl::search::KdTree<PointT>::Ptr tree (new pcl::search::KdTree<PointT> ());
239
240     ///Número máximo de dedos que pueden ser encontrados (depende del guante utilizado)
241     unsigned num_max_dedos=3;
242
243     ///Espera hasta que el hilo del estéreo genere la primera imagen.
244     mutex_coordinacion.lock();
245     mutex_coordinacion.unlock();
246
247
248     while(1) ///Inicio del bucle
249     {
250             mutex_ima3D.lock();
251             cloud_original->points.clear();
252             unsigned tam=0;
253             ///Bucle para determinar el tamaño que deberá tener la nube de puntos para que pueda
                almacenar todos los puntos calculados por el estéreo
254             for(int i=0;i<ima3D.rows;i++)
255             {
256                     for(int j=0;j<ima3D.cols;j++)
257                     {
258                             if(abs(ima3D.at<Vec3f>(i,j)[0])<1000 &&
                                abs(ima3D.at<Vec3f>(i,j)[1])<1000)
259                                     tam++;
260                     }
261             }
262             ///Definición del tamaño de la nube de puntos
263             cloud_original->width = tam;
264             cloud_original->height = 1;
265             ///Introducción de los puntos en la nube
266             for(int i=0;i<ima3D.rows;i++)
267             {
268                     for(int j=0;j<ima3D.cols;j++)
269                     {
270                             PointT basic_point; ///Se declara un punto con unas determiandas
                                coordenadas y después se incluye en la nube.
271
272                             if(abs(ima3D.at<Vec3f>(i,j)[0])<1000 &&
                                abs(ima3D.at<Vec3f>(i,j)[1])<1000)
273                             {
274                                     basic_point.x = ima3D.at<Vec3f>(i,j)[0];
275                                     basic_point.y = ima3D.at<Vec3f>(i,j)[1];
276                                     basic_point.z = ima3D.at<Vec3f>(i,j)[2];
277                                     cloud_original->points.push_back(basic_point);
278                             }
279                     }
280             }
281             mutex_ima3D.unlock();
282
283             ///Reducción de los puntos almacenados en la nube, el resultado se vuelca en la nube
                "cloud"
284             pcl::VoxelGrid<pcl::PointXYZ> sor;
285             sor.setInputCloud (cloud_original);
286             sor.setLeafSize (1.0f, 1.0f, 1.0f);
287             sor.filter (*cloud);
288
289             ///Definición de los parámetros de búsqueda de cilindros
290             seg.setOptimizeCoefficients (false);
291             seg.setModelType (pcl::SACMODEL_CYLINDER);
292             seg.setMethodType (pcl::SAC_RANSAC);
293             seg.setNormalDistanceWeight (0.1);
294             seg.setMaxIterations (500);
295             seg.setDistanceThreshold (3);
296             seg.setRadiusLimits (5, 10);
297
298             unsigned contador=0;
299             unsigned tam_original=cloud->points.size();
300
301             for(contador;contador<num_max_dedos;contador++)
302             {
303                     ///Estimación de direcciones normales de cada punto
304                     ne.setSearchMethod (tree);
305                     ne.setInputCloud (cloud);
306                     ne.setKSearch (50);
307                     ne.compute (*cloud_normals);
308
309                     ///Búsqueda de cilindros
310                     seg.setInputCloud (cloud);
311                     seg.setInputNormals (cloud_normals);
312                     seg.segment (*inliers_cylinder, coefficients_cylinder);
313
314                     ///Extracción de puntos de la nube cloud y volcado de los mismos en
                        "cloud_cylinder"
315                     extract.setInputCloud (cloud);
316                     extract.setIndices (inliers_cylinder);
317                     extract.setNegative (false);
318                     extract.filter (*cloud_cylinder);
```

```
319                     extract.setNegative (true);
320                     extract.filter (*cloud);
321
322                     if(cloud_cylinder->points.size()>tam_original/(num_max_dedos*2))
                        ///Comprobación del tamaño del cilindro detectado
323                     {
324                             coeficientes.push_back(coefficients_cylinder); ///Almacenado de los
                                parámetros del cilindro en un vector, información que se pasará al
                                robot
325
326                             ///Búsqueda del extremo del dedo.
327                             extremo_dedo.push_back(cloud_cylinder->points[0]);
328                             for(unsigned j=0;j<cloud_cylinder->points.size();j++)
329                             {
330                                     if(cloud_cylinder->points[j].y<extremo_dedo[contador].y)
331                                             extremo_dedo[contador]=cloud_cylinder->points[j];
332                             }
333                     }else break; ///Si el cilindro encontrado no cumple el tamaño mínimo se
                        deja de buscar
334             }
335             dedos_encontrados=contador;
336
337             system("clear"); ///Llamada al sistema para que limpie el terminal, en Windows
                cambiar "clear" por "cls".
338             ///Representación de la información obtenida.
339             cout << "Se han encontrado " << dedos_encontrados << " dedos." << endl;
340             for(unsigned i=0;i<dedos_encontrados;i++)
341                     cout << "Extremo del dedo " << i+1 << " -----> " << extremo_dedo.at(i) <<
                        endl;
342     }
343}
```