

SISTEMA PARA DETECCIÓN Y RECONOCIMIENTO DE IRREGULARIDADES SOBRE CIRCUITOS VIALES UTILIZANDO EL SENSOR KINECT



Autores: Huincalef Rodrigo A. , Urrutia Guillermo G.

Director de tesina: Ingravallo Gabriel

Tesina presentada a la Facultad de Ingeniería de la Universidad Nacional
de la Patagonia San Juan Bosco como parte de los requisitos para la
obtención del título de Licenciado en Sistemas

Facultad de ingeniería - Sede Trelew

UNPSJB - Universidad Nacional de la Patagonia San Juan Bosco

25 De Febrero Del 2018

Organización de los contenidos

Capítulo 1. Introducción

En este capítulo se introducen los objetivos de la tesina, el marco teórico y los desarrollos propuestos.

Capítulo 2. Antecedentes de software para la gestión de fallas viales

Este capítulo comienza definiendo el concepto de falla, los tipos de material, tipos de reparación que se pueden efectuar sobre cada uno, y los tipos de falla que se corresponden sobre los distintos tipos de material. Durante este capítulo también se debaten antecedentes históricos, software y proyectos de investigación, relativos al sensado de fallas en distintos contextos.

Capítulo 3. Dispositivos hardware y herramientas software para el sensado de fallas

El objetivo de este capítulo es enumerar los distintos tipos de dispositivos y sensores que pueden utilizarse para realizar la captura de irregularidades viales, sus principales características y funcionamiento. Por otro lado, se expondrán librerías en distintos lenguajes de programación, y métodos de almacenamiento de las fallas empleados por los mismos.

Capítulo 4. Técnicas de reconocimiento y procesamiento de fallas

Durante el desarrollo de este capítulo se definirá el concepto de Machine Learning, sus usos, contextos de aplicación y algoritmos que brinda para el reconocimiento de patrones en conjuntos datos. Posteriormente, se expondrán aquellas técnicas empleadas para realizar la identificación, el saneamiento y la clasificación de los datos de una nube de puntos con datos de una falla.

Capítulo 5. Herramientas GPS y Geocoding

El objetivo de este capítulo consiste en explicar el concepto de GPS, latitud, longitud, funcionamiento, dispositivos GPS disponibles y utilidades software para compartir conexiones GPS entre sistemas operativos Android y Linux. Además, se explicará el concepto de Geocoding y Reverse Geocoding, librerías disponibles en PHP y Python y, servicios web ofrecidos por terceros compatibles con éstas.

Capítulo 6. Caso de aplicación

El objetivo de este capítulo es explicar tanto la arquitectura general de la aplicación cliente como de la aplicación web, los objetivos de las mismas, el ciclo de trabajo y las formas de interacción con ambas aplicaciones. Por un lado, se expondrá el funcionamiento de la aplicación cliente, librerías empleadas para su desarrollo, formas de interacción de ésta con el sensor Kinect y diseño. Por otro lado, se presentará la funcionalidad base que la aplicación web ofrecía anteriormente, diseño, la funcionalidad incluida como parte del desarrollo de la presente investigación y librerías relacionadas al funcionamiento agregado a la misma.

Capítulo 7. Conclusiones y líneas futuras

En este capítulo, se explican las conclusiones de la investigación realizada y posibles líneas de investigación futuras con el presente desarrollo.

Índice:

Abstract	1
Reconocimientos	2
Resumen	3
1 Resumen	3
Introducción	4
1 Objetivo general	4
2 Objetivos específicos	4
3 Motivación	4
4 Marco teórico	5
5 Desarrollos propuestos	6
6 Resultados esperados	6
7 Cronograma de actividades	6
Capítulo 3.Sensor Kinect y herramientas software implicadas el sensado de fallas	8
1 Sensores 3D	8
1.1 Sensor Kinect V1	14
2 Librerías para la utilización del sensor Kinect	18
2.1 Kinect for Windows SDK 1.8 (Xbox Development Kit)	19
2.2 OpenNI	22
2.3 Freenect y Librería Point Cloud Library(PCL)	23
2.3.1 Algoritmos de pre-procesamiento de nubes	24
2.3.2 Algoritmos de segmentación de objetos	37
2.3.3 Algoritmos para generación de descriptores	39
Capítulo 4.Técnicas de reconocimiento y procesamiento de fallas	41
1 ¿Qué es Machine Learning(ML)?	41
1.1 Flujo de trabajo en ML	42
1.2 Aplicaciones de ML	56
1.3 Beneficios del uso de ML	57
2 Mecanismos para Machine Learning(ML)	57
2.1 Árboles de decisión(Tree)	57
2.2 Redes Neuronales(ANN)	59
2.3 Maquinas de soporte de Vectores(SVM)	60
3 Selección de features para ML en PCL	62
3.1 PFH-FPFH	62
3.2 VFH	65
3.3 GRSD	66
3.4 ESF	68
4 Metodología de pre-procesado de muestras (Pipeline de Cropeado)	69
5 Metodología para el procesamiento de muestras con ML	72
6 Bitácora de pruebas para clasificación	73
Bibliografía	79

Abstract

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquid ex ea commodo consequat. Quis aute iure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint obcaecat cupiditat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquid ex ea commodo consequat. Quis aute iure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint obcaecat cupiditat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquid ex ea commodo consequat. Quis aute iure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint obcaecat cupiditat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquid ex ea commodo consequat. Quis aute iure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint obcaecat cupiditat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Reconocimientos

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vivamus et augue vel nibh pharetra cursus et vel mauris. Integer a ante in leo fermentum tincidunt quis vitae mauris. In vitae bibendum lectus. Curabitur porttitor, sapien et tincidunt suscipit, nisl ligula euismod nisi, ut suscipit dui metus vel eros. Nullam consectetur erat quam, in sollicitudin lacus lobortis eget. Proin felis diam, pellentesque eu finibus dapibus, congue a ligula. Donec gravida quam ante, ac rutrum mauris pharetra ac. Interdum et malesuada fames ac ante ipsum primis in faucibus. Curabitur malesuada metus nec gravida sollicitudin. Morbi a semper ex. Sed sollicitudin lobortis dolor quis mattis. Sed vel feugiat erat.

Quisque dignissim ligula ligula, id fermentum dui imperdiet ac. Quisque pellentesque lectus dolor, id ultrices nisi porttitor sed. Maecenas quam neque, vestibulum in libero nec, vestibulum venenatis mauris. Aliquam nec orci dictum, efficitur tortor id, lacinia arcu. Phasellus tincidunt tristique dui, pretium convallis est luctus nec. Integer molestie euismod dui, vel pharetra purus egestas vitae. Nullam eu leo est. Integer euismod sodales feugiat. Ut ac scelerisque est.

Fusce auctor ex ut tellus sollicitudin, vel placerat purus auctor. Cras ut dolor iaculis, lobortis felis dignissim, vehicula elit. Ut malesuada lectus ligula, quis sagittis erat consectetur vel. Etiam tempor tellus sit amet varius tempor. Mauris lacinia mi et eros bibendum lacinia. Ut blandit porttitor dictum. Sed pellentesque diam ut tortor faucibus, vel vestibulum metus congue. Sed sed purus sem. Sed sagittis maximus enim, id consequat mi convallis ut.

Resumen

1. Resumen

El objetivo de la presente tesina consiste en incorporar a un sistema de detección de fallas, funcionalidad para el sensado tridimensional de fallas viales, exponiendo software disponible para registro de distintos tipos de fallas sobre senderos viales, dispositivos hardware y herramientas de software asociadas al sensado de las mismas, y técnicas para el reconocimiento y clasificación de las muestras de fallas capturadas.

Con respecto al software para el registro de fallas sobre senderos viales, se expondrá de manera breve la terminología relacionada a los distintos tipos de fallas, y desarrollos software relacionados con el sensado y registro de fallas en sistemas de información.

En cuanto a dispositivos hardware, se estudiarán distintos tipos de sensores para el relevamiento de superficies, características principales y funcionamiento. Asimismo, se investigarán los lenguajes de programación disponibles, librerías, herramientas y disponibilidad en distintos sistemas operativos. Adicionalmente, se incluirán las utilidades de software para obtención de latitud y longitud con un dispositivo compatible con GPS, y las librerías web que permiten obtener datos de la dirección estimada asociada a dichas coordenadas.

Además, se expondrán las técnicas de filtrado y clasificación empleadas para el análisis de distintos tipos de falla, antecedentes históricos, y proyectos de similares características.

Finalmente, se realizarán modificaciones a una aplicación web previamente desarrollada, con el fin de incluir la visualización de distintos tipos de falla en un navegador, el filtrado de distintos tipos de fallas sobre una calle determinada, y la estimación de la dirección aproximada de una falla, basándose en la latitud y longitud. Además, se realizará el desarrollo de una aplicación cliente que permita la captura de distintos tipos de fallas por medio del sensor Kinect. Ésta será capaz de interactuar con la aplicación web, para obtener información respecto de la ubicación de las fallas registradas anteriormente por sus usuarios, lo que le permitirá asociar archivos de capturas a las mismas. También, la aplicación cliente permitirá registrar en la aplicación web aquellas fallas descubiertas durante un recorrido (sin registro previo), obteniendo la latitud y longitud de la falla por medio de un GPS, para luego realizar una estimación de la dirección en la aplicación web. Posteriormente, se podrá realizar una selección de las todas las fallas capturadas y realizar la subida de las mismas con sus archivos de capturas al servidor, desde la aplicación cliente. Por último, se desarrollará un script separado, que utilizará los filtros y mecanismos de machine learning ofrecidos por la librería Point Cloud Library (PCL), para el aislamiento y la clasificación de los tipos de fallas “Bache” y “Grieta”.

Introducción

1. Objetivo general

Extender un sistema de notificación y administración de información pertinente a irregularidades en superficies viales, por medio del censado visual tridimensional semiautomático de información vial, con el fin de contribuir a la toma de decisiones respecto del estado de los circuitos viales.

2. Objetivos específicos

- Desarrollar un servicio web para el almacenamiento de capturas realizadas por el sensor desde el aplicativo de captura.
- Procesar la información tridimensional contenida en un archivo con el fin de identificar las diferentes propiedades y clasificar el tipo de falla capturada.
- Implementar una aplicación que permita la captura tridimensional, en un formato especificado por el sensor Kinect, y su posterior carga por medio del servicio web.
- Integrar el servicio web de almacenamiento de capturas a la aplicación web de gestión de fallas para su visualización en línea.

3. Motivación

El sensor Kinect desarrollado por Microsoft, es un sensor de detección de movimiento que permite la interacción usuario-dispositivo sin la necesidad de controles adicionales. Este sensor, ofrece una amplia gama de características entre las que se destacan una cámara RGB de 640x480 px a 30 FPS, un emisor y un sensor infrarrojos, que le permiten obtener información acerca de la profundidad, capacidades de reconocimiento de voz y reconocimiento facial. Esta amplia variedad de características, sumadas al bajo costo de adquisición, y la disponibilidad de paquetes de código abierto disponibles, hacen que este dispositivo sea apropiado para la explotación en innovaciones.

Las nuevas tecnologías web y la amplia variedad de dispositivos y conectividad actual, hacen de Internet una plataforma computacional con una gran incidencia en la operación y acceso a la información

por parte de las personas. Estas características, sumadas al crecimiento en el uso de estas tecnologías en dispositivos móviles actuales, permiten aumentar la facilidad con la que los usuarios pueden acceder e informar de manera intuitiva, acerca de problemas relacionados a irregularidades en las calles de la ciudad.

La librería Point Cloud Library (PCL), es una herramienta libre, potente y con un gran conjunto de algoritmos para el procesamiento de imágenes y nubes de puntos en 2D/3D. Esta librería reúne investigadores, universidades, compañías e individuos de todo el mundo y se está convirtiendo rápidamente en una referencia para cualquier interesado en el procesamiento 3D y robótica.

El núcleo de PCL está estructurado en pequeñas librerías y algoritmos para áreas específicas del procesamiento 3D que pueden ser combinadas para resolver problemas como el reconocimiento de objetos, registro de nubes de puntos, segmentación y reconstrucción de superficies facilitando la reutilización de código.

Las irregularidades son una característica común en las rutas y calles pavimentadas de una ciudad lo que genera problemas en el flujo del tráfico y puede ocasionar situaciones de conducción potencialmente peligrosas. Debido a la constante degradación de los diferentes tipos de fallas, estas deben ser detectadas y reparadas tan pronto como sea posible. Existen diferentes formas para la detección de la ubicación y las propiedades de las irregularidades como forma, tamaño, profundidad y volumen, lo que juega un rol importante con respecto a la reparación de las mismas.

La reciente introducción de programación de aplicaciones web en la carrera, requirió introducir personal especializado de la Universidad Nacional del Sur (UNS), y por medio de las prácticas realizadas en esta asignatura, se inició con el desarrollo de un trabajo final que inicialmente georeferenciaba fallas. Ésto dio pie a un proyecto de investigación dirigido por Gabriel Ingravallo (UNPSJB Trelew), y asesorado por Diego C. Martínez (UNS Bahía Blanca) para la recolección y análisis de fallas viales y que hacía uso del sensor Kinect, del que participaron desde su inicio los presentes tesis. Sin embargo, el proyecto no cuenta con un mecanismo de análisis y clasificación de diferentes tipos de fallas.

La falta de innovación con respecto al registro y análisis del estado de las vías de circulación de la ciudad y la necesidad de un mecanismo de análisis y clasificación de distintos tipos de fallas en el proyecto de investigación, conforman la principal motivación de esta tesis, que busca extender la funcionalidad de éste, agregando la visualización de las dimensiones de la falla, incluyendo el procesamiento de nubes de puntos para las irregularidades, la clasificación y extracción de sus propiedades relevantes.

4. Marco teórico

A través del tiempo, la tarea de registrar los defectos y depresiones en el pavimento siempre ha sido crucial para adoptar una estrategia precisa para el mantenimiento y las tareas de reparación de los senderos viales y, debido a que la medición manual es una tarea costosa en tiempo y recursos, se ha experimentado con diversas tecnologías con el fin de automatizar y reducir este costo. En el rango de estas tecnologías, se han considerado tanto sensores de medición de diferentes características inherentes a las irregularidades viales, como así también tecnologías para el geoposicionamiento y el registro y administración fallas por medio de sistemas web.

Entre las aproximaciones de detección y análisis de fallas, se abarcan la investigación y experimentación con imágenes y videos 3D, empleando técnicas de procesamiento de imágenes, con el fin de aislar las fallas basadas únicamente en la diferencia de niveles color en pixeles y/o textura que las fallas suelen poseer, respecto de los pixeles del sendero vial donde se encuentran. Alternativamente, se ha experimentado con sistemas que emplean sensores de vibración, tales como acelerómetros, que ubicados precisamente en uno o varios lugares de un vehículo, intentan detectar las diferencias de vibración abruptas, que se encuentran en un rango propio de las fallas. Otra aproximación, consiste en emplear sensores para medir las características espaciales de las fallas viales, y luego realizar un procesamiento tridimensional de las coordenadas capturadas aplicando diversos algoritmos, de manera que se puedan detectar diferencias de nivel entre puntos del plano y de la falla, y así lograr aislar la falla del plano. Cada una de las aproximaciones descriptas anteriormente cuentan con ventajas como así también con desventajas, siendo cada una sensible a diferentes factores externos en el entorno en donde se realiza la medición.

5. Desarrollos propuestos

Se propone el desarrollo de una aplicación que haciendo uso de las tecnologías web actuales, permitirá a los actores involucrados en el proceso de mantenimiento de vías de circulación recolectar, procesar y visualizar los diferentes tipos de fallas sobre las distintas calles de la ciudad. Así como, permitir a los habitantes de la ciudad informar y mantenerse informados respecto del estado de las calles.

Con respecto al proceso de recolección se propone implementar un pequeño aplicativo desarrollado en Python, que junto al sensor Kinect, posibilitará almacenar la nube de puntos censada (que representa una porción de un sendero vial) por el dispositivo. En cuanto al procesamiento, se investigará y experimentará con las técnicas ofrecidas por la librería PCL, que contribuyan a la identificación y delimitación de un tipo de irregularidad dentro de la captura realizada. Una vez identificado el método más conveniente, se desarrollarán varios módulos en C++, interconectados que permitirán la identificación y clasificación del tipo de falla y la extracción de sus propiedades.

Por último, con respecto a los mecanismos de notificación y visualización, se hará uso de distintas tecnologías web del lado del cliente (Javascript, CSS3, HTML5) que darán el formato a las herramientas ofrecidas para visualizar las fallas, y operará en conjunto con framework PHP (CodeIgniter 2) del lado del servidor.

6. Resultados esperados

Con el desarrollo de la presente tesisina, se espera obtener una plataforma que permita mantener informados a los habitantes locales con respecto al estado de las calles de la ciudad a través del acceso a una aplicación web que indique la ubicación de las fallas y el estado actual en el que se encuentran las mismas. Esta a su vez, permitirá a los habitantes informar la localización falla sobre una calle determinada.

Por otro lado, se producirá un módulo de software, que mediante el procesamiento de nubes de puntos tridimensionales posibilite la obtención de una medida objetiva que caracterice distintos tipos de falla. Para ello, se investigará acerca de las estructuras ofrecidas con la librería PCL, y se experimentará con el sensor Kinect, para lograr comprender la estructura interna de una captura realizada por el dispositivo y el funcionamiento de los algoritmos que procesan dicha captura (algoritmos que eliminan el ruido y filtran dicha captura).

7. Cronograma de actividades

Las actividades que se efectuarán para cumplir con el objetivo de la tesisina se detallan enumeradas junto a su número de tarea:

1. Documentar la información técnica durante el desempeño de las tareas.
2. Investigar acerca del funcionamiento del sensor Kinect y los distintos módulos ofrecidos por su SDK bajo Linux.
3. Investigar la librería de procesamiento PCL.
4. Investigar acerca métodos de detección de fallas sobre vías de circulación.
5. Investigar sobre la utilización del sensor Kinect para detección de objetos.
6. Experimentar con la librería PCL y el sensor Kinect acerca de distintos métodos de procesamiento y análisis de nubes de puntos tridimensionales.
7. Desarrollar un módulo en Python para la captura de nubes de puntos.
8. Documentar los pasos requeridos para sanear y detectar una falla en el archivo de nube de puntos.
9. Documentar las técnicas requeridas para delimitar y clasificar un tipo de falla a partir de una nube de puntos.

10. Desarrollar módulos en C++ para el análisis de fallas (el saneamiento, la detección, delimitación y clasificación de los tipos de fallas).
11. Investigar el renderizado de archivos de nubes de puntos con WebGL.
12. Integrar la funcionalidad de la aplicación web con los módulos de procesamiento de nubes de puntos desarrollados.
13. Defensa de la tesina ante mesa examinadora.

Capítulo 3. Sensor Kinect y herramientas software implicadas el sensado de fallas

1. Sensores 3D

Los scanners o sensores 3D, son elementos de medición que se emplean con el fin de analizar objetos o entornos del mundo real y obtener información respecto a sus características físicas, tales como colores y/o forma, que posteriormente pueden emplearse para generar modelos 3D digitales de éstos. Estos modelos se representan por medio de una nube de puntos (Point Cloud), que es una estructura de puntos definidos en algún sistema de coordenadas, siendo la definición más empleada las coordenadas tridimensionales (X,Y,Z). Estas representaciones pueden ser directamente renderizadas inspeccionadas, sin embargo, para su uso en aplicaciones comerciales de edición o modelado deben ser convertidas en mallas poligonales o modelos triangulados, modelos NURBS (modelo matemático empleado para la representación de superficies y curvaturas) o modelos compatibles con el diseño asistido por computadora (CAD). Estos sensores son similares a las cámaras digitales, ya que cuentan con un campo de visión cónico y solamente pueden recolectar información respecto del entorno en lugares que cuentan con suficiente iluminación. Sin embargo, a diferencia de éstas que únicamente capturan información respecto de los colores de la superficie, los scanners 3D son capaces de recolectar información acerca de la distancia de cada punto en la imagen dentro de su rango de visión, por lo que la coordenada de cada punto puede ser precisada. Aunque la imagen proporcionada por un scanner 3D incluya información de posicionamiento de los puntos que componen una imagen, una única captura del objeto no bastará para brindar un modelo tridimensional completo de las características de éste, por lo que si se desea obtener éste resultado, se deben realizar varias capturas desde distintos puntos de vista del mismo objeto y luego realizar la unión final de todas estas capturas en una captura final.

Existen diferentes tipos de sensores 3D que emplean distintas técnicas para la obtención del objeto 3D, aunque de manera general se clasifican en: Técnicas de contacto, donde las características del objeto se obtienen por medio del contacto físico, sosteniéndolo por medio de un brazo robótico que puede ser manipulado para escanear la superficie completa de un objeto o, manteniéndolo apoyado sobre una plataforma fija; Y técnicas sin contacto, donde se detecta radiación no visible (rayos infrarrojos X) o radiación visible (luz solar) sobre el objeto con el propósito de adquirir información de éste. Dentro de los tipos de sensores sin contacto, éstos se pueden subdividir en activos y pasivos.

Los sensores activos sin contacto, son aquellos que emiten algún tipo de radiación o luz sobre un objeto y absorben su reflexión con el fin de inspeccionar el objeto y, aproximaciones existentes para medir esta radiación se encuentran las siguientes:

-
- Time of Flight(ToF)
 - Diferencia de fase (Phase Shift)
 - Triangulación por láser
 - Luz estructurada
 - Luz Modulada

Los tipos de láser ToF emiten varios impulsos láser hacia un objeto y miden el tiempo que requiere alcanzar el objeto y ser reflejado de vuelta al sensor emisor y por medio de la siguiente fórmula computan la distancia:

$$D = c * t/2 \quad (5.1)$$

Dado que la velocidad de la luz en la atmósfera c es una constante que se conoce, solamente se debe calcular el tiempo de viaje y de retorno t , por lo que la precisión con la que un láser ToF detecta un objeto, depende en gran medida de la precisión respecto de la medición del tiempo. Por lo tanto, la detección de este tipo de dispositivos consiste en realizar un escaneo por medio de la emisión de un láser sobre un objeto y en base a este resultado, computar las distancias y las coordenadas de los puntos asociados a éstas, que componen el objeto.

Existen diferentes métodos para el escaneo del objeto por medio de lasers, entre ellas las aproximaciones más comunes son las que emplean los sensores Lidar y las Cámaras ToF. El sensor Lidar (Light Detection And Ranging) emplea un pulso láser para medir la distancia a un objeto variando rápidamente su dirección por medio de un sistema de espejos rotativos, modificando los ángulos horizontales y verticales, de manera que se puedan captar las distancias de todos los puntos del objeto que se encuentra en el campo de visión. Una vez obtenidas las distancias y conociendo los ángulos horizontales y verticales, se pueden computar la posiciones (X,Y,Z) para cada uno de los puntos. Así, partiendo de la diferencia entre los tiempos de retorno de los rayos y, adicionalmente, empleando diferencias entre longitudes de onda (emitidas y recibidas), se genera un modelo tridimensional del objetivo. Estos dispositivos se emplean principalmente en satélites para la generación de mapas terrestres de alta resolución, topografía, documentación histórica de objetos antiguos y en la detección de objetos en vehículos autónomos.



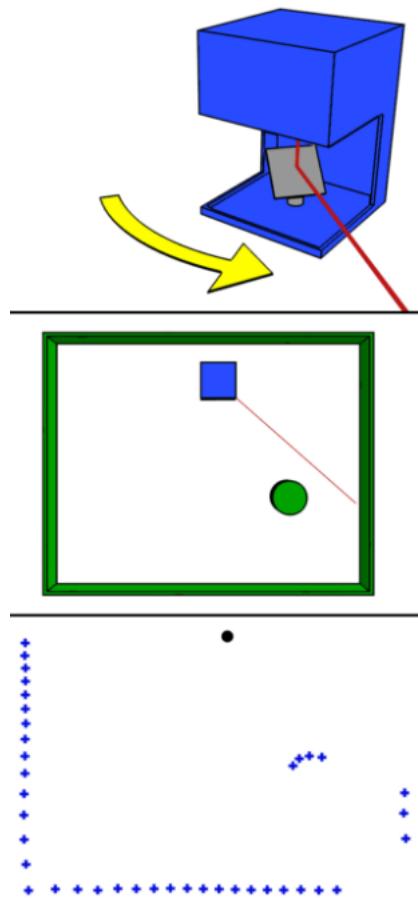


Figura 5.1: Ejemplo de láser tipo Lidar. En la figura superior, se puede observar el rayo proyectado (línea roja) que es desplazado para escanear una escena entera por medio de un espejo rotativo. En la figura central se denota con azul el láser, con una círculo verde un objeto esférico y un rectángulo verde que constituye las paredes del entorno en el que se ubica el dispositivo. Finalmente, en la figura inferior se muestra el sensor representado como un punto negro, y los puntos azules simbolizan cada uno de los puntos capturados por el dispositivo durante el escaneo.



Figura 5.2: Robot móvil con un láser Lidar SICK Laser Rangefield.

Por otro lado, la cámara ToF no escanea individualmente cada uno de los puntos en el campo de visión, sino que emite varios pulsos de radiación simultáneamente, capturando la escena completa por

cada emisión. Luego una cámara sensible a la radiación captura la imagen producida por la reflexión del conjunto de pulsos (cuyas características variarán dependiendo de la distancia entre el sensor y el objeto), se filtra la radiación de interés, cuya longitud de onda coincide con la del tipo que fue emitida, se calcula las coordenadas de los puntos y se genera la imagen de salida. Estos dispositivos se emplean en:

- Aplicaciones automovilísticas para detección de peatones y prevención de colisiones, interfaces humanas donde se captura la interacción con el usuario, tales como televisores y videojuegos, donde se emplea el dispositivo para capturar movimientos y gestos que son utilizados como gamepads (como la segunda generación del sensor Kinect Version 2), en robótica, robots que se desplazan en un entorno donde deben esquivar obstáculos o seguir una persona, o en visión por computadora en entornos industriales para tareas automatizadas de medición, o detección de objetos que serán empleados por un robot para realizar una tarea.



Figura 5.3: Kinect V2 con cámara ToF desarrollada por Microsoft.

Este tipo de sensores, tienen la ventaja de ser rápidos para el muestreo, de alta precisión, aptos para trabajos de medición en monumentos o elementos de construcción, con una alta densidad de puntos por captura, una velocidad de captura entre 10.000 y 100.000 puntos por segundo y un rango de medición alto entre 200 y 300 m. Sin embargo, estos dispositivos no cuentan con una resolución de profundidad alta, la precisión del modelo generado es aproximadamente 1 cm y no se cuenta con información de color.

- Los láseres de diferencia de fase (Phase Shift) funcionan emitiendo un haz de luz constante que sigue un patrón de onda sinusoidal, con una longitud de onda específica, el cual es almacenado en el sensor y emitido hacia el objeto. Luego, el reflejo de este rayo es capturado por el sensor y es comparado contra el patrón original almacenado en el sensor, con el fin de obtener la diferencia de fase entre ambas señales y así computar las distancias.

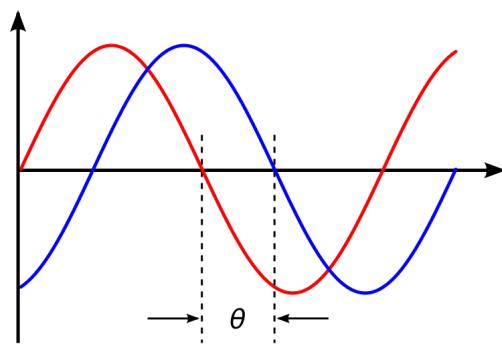


Figura 5.4: Representación gráfica de la diferencia de fase entre dos ondas sinusoidales.

Este proceso es similar a la técnica de tiempo de vuelo, excepto que la fase del láser reflejado refina la precisión respecto de la detección de la distancia. El alcance de este tipo de sensores se encuentra limitado por las características de la señal emitida, ya que la precisión de la medición es inversamente proporcional a la frecuencia empleada, por lo que medir con una alta frecuencia brinda mayor precisión en la distancia, pero menor rango de medición. Este tipo de dispositivo cuenta con una velocidad de adquisición alta (ubicándose ésta entre 100.000 y 1.000.000 de puntos por segundo) y un alcance de captura intermedio entre 70-100 m. Sin embargo, aunque la velocidad de captura es considerablemente superior a los scanners ToF, las nubes de puntos generadas por el método de diferencia de fase suelen contener más ruido.

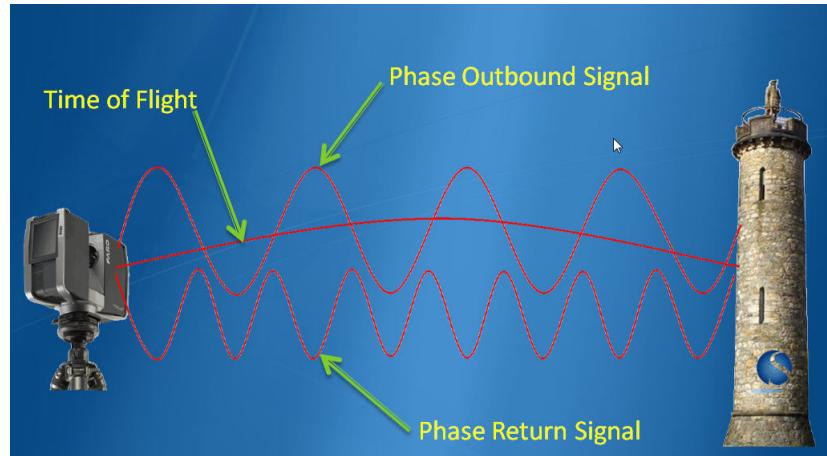


Figura 5.5: Comparación gráfica de laser ToF y Phase Shift

Los escáneres 3D de triangulación, se componen por un láser y una cámara que captura los rayos emitidos por éste a través de una lente, la cual permite enfocar la posición de cada punto abarcado por el rayo en el campo de visión del sensor. De esta forma, la captura de un objeto consiste en emitir un rayo sobre el objeto y capturar el reflejo de éste sobre la misma en la lente de la cámara y, dependiendo de la posición que adopte el reflejo del rayo en el lente se puede computar el ángulo α para cada punto. Finalmente, con este valor, la distancia entre la cámara y el láser y el punto del objeto donde rebota el rayo se forma un triángulo, a partir del cual es posible calcular por triangulación la distancia entre la cámara y un punto. Así, dependiendo de que tan lejano el objeto se encuentre el punto, parecerá en distintas posiciones del campo de visión de la cámara.

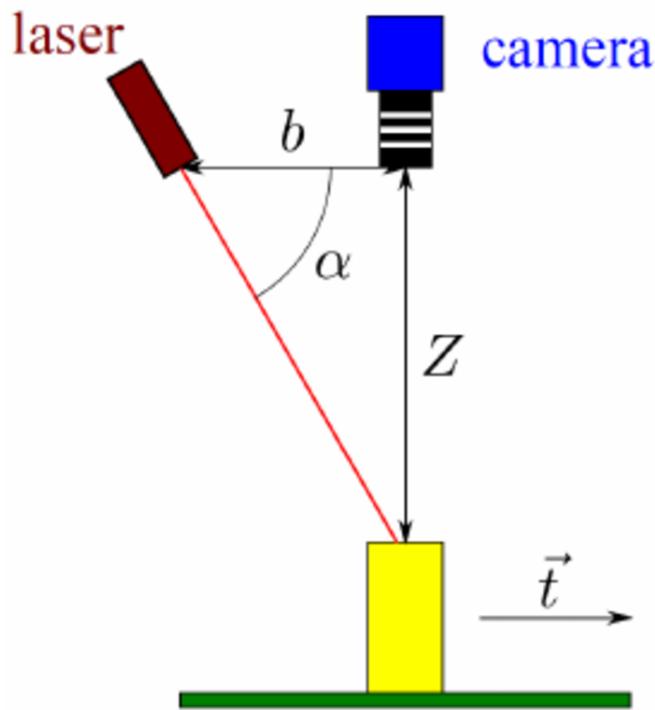


Figura 5.6: Esquema gráfico de scanners de triangulación.

Los sensores de luz estructurada 3D (Structured Light 3D), proyectan un patrón de luz sobre un objeto o superficie, pudiendo ser éste unidimensional (una línea de luz) o bidimensional (una grilla o patrón de líneas). Un sensor que se encuentra desplazado del emisor de luz, se emplea para captar la deformación del rayo original y a partir de esta calcular la distancia. Este tipo de dispositivos

se caracterizan por ser veloces, ya que en lugar de escanear un punto a la vez, los sensores escanean múltiples puntos en el campo de visión con una sola emisión, lo que elimina la posibilidad de distorsión por movimiento. La resolución y velocidad de estos sensores es similar a la de las cámaras VGA, y su precisión es similar a las ToF (aproximadamente 1 cm), contando con un alcance máximo entre 3 m y 6 m, sin embargo estos dispositivos tienen dificultades para captar objetos pequeños menores a 1 cm.



Figura 5.7: Ejemplo de sensor ASUS Xion Pro con luz estructurada.

En los escáner de luz modulada (Modulated Light), la radiación emitida por el láser es una luz que varía continuamente en base a un patrón establecido, siendo generalmente una onda sinusoidal de la que se repiten varios ciclos, hacia el objeto que se desea escanear. Así, una cámara detecta la diferencia entre la onda de la radiación emitida y la radiación reflejada por el objeto, y dependiendo de la diferencia entre éstas se computan las distancias.

Por otro lado, la técnica de escáner sin tacto pasivos, no emite ningún tipo de radiación sino que su funcionamiento se basa en capturar la radiación reflejada del ambiente, como la luz solar o la luz infrarroja. Estos tipos de escáner se caracterizan por ser económicos, ya que no requieren de hardware especializado para la emisión de un tipo de radiación. Dentro de esta categoría de dispositivos, se encuentran los siguientes:

- Escáner estereoscópicos (Stereoscopic Scanners): Estos dispositivos emplean dos cámaras de video en posiciones diferentes (desplazadas algunos centímetros) enfocadas hacia el mismo objeto, que captarán imágenes distintas del mismo y, por medio del análisis de estas diferencias, se puede calcular la distancia de cada punto en las imágenes. Un ejemplo de este tipo de dispositivos son las cámaras estéreo, que cuentan con la ventaja de ser económicas, sin embargo requieren una perfecta calibración de ambas cámaras de video y son sensibles a las malas condiciones de iluminación.



Figura 5.8: Cámara estéreo

- Silhouette scanners: Estos escáner capturan una secuencia de imágenes para generar un contorno alrededor de un objeto que contrasta con el fondo, que posteriormente son superpuestos para formar un hull visual y generar una aproximación del objeto.

Con respecto a los escáner de contacto, un ejemplo de su funcionamiento son las cámaras de medidas de coordenadas(Coordinate Measuring Machine) empleadas para la medición de las características geométricas de partes o productos industriales ensamblados. Este dispositivo se compone de tres ejes X,Y,Z ortogonales entre si, donde cada uno mantiene una escala para registrar las coordenadas del elemento que se analiza. Así, este tipo de escáner desplaza uno de los ejes (ya sea automáticamente o manualmente) mientras que el resto se mantiene fijo, y graba cada una de las coordenadas del objeto.



Figura 5.9: Ejemplo de Coordinate Measure Machine.

Existen varias áreas donde se aplican los escáneres 3D, entre las más comunes se encuentran:

- Control de calidad industrial. Una de las principales aplicaciones de los escáneres 3D consisten en la digitalización de partes producidas, tanto en el diseño como en la producción de la parte final. Estos dispositivos deben ser precisos y versátiles, con el fin de obtener la mayor cantidad de información acerca del proceso de construcción de partes.
- Registros históricos. En esta rama se emplean dispositivos 3D sin contacto que permitan el análisis de restos animales o artefactos antiguos sin dañarlos, con el fin de generar un modelo a mayor o menor escala, para ser exhibido en museos.
- Ciencias médicas. Dentro de las ciencias médicas, los escáneres 3D se emplean por ejemplo, en la rama de construcción de piezas dentales, ya que por su precisión y adquisición sin contacto, permiten generar de manera satisfactoria piezas cuyas dimensiones serían complejas de adquirir.
- Gráficos por computadora. Debido a que con las tecnologías actuales en la industria de videojuegos permiten la creación de entornos con mayor nivel gráfico de detalle, actualmente se recurre a los escáneres 3D para la construcción de éstos en el entorno del videojuego, ya que escanear estos objetos consume menos tiempo que la creación a mano con herramientas digitales.

1.1. Sensor Kinect V1

El sensor Kinect es un dispositivo de juego compatible con las plataformas Xbox y PC, desarrollado por las compañías Microsoft y Primesense, pensado para la interacción del usuario sin la necesidad de controles físicos de juego, definiendo para la interacción humano-máquina una interfaz de gestos y de comandos hablados. Este dispositivo tiene la capacidad de generar imágenes de rango (Range Image), que son aquellas que, por cada pixel de la imagen tienen asociada la información de distancia de cada uno, hacia el punto de captura. Este dispositivo fue lanzado en dos versiones, la versión Kinect V1 para la consola Xbox 360 y la versión Kinect V2 para la consola Xbox One.

La versión Kinect V1 (empleada para la captura de muestras de la presente tesis) se basa en la técnica de proyección de luz estructurada 3D con luz infrarroja(IR) constituyéndose por: Un emisor IR, una cámara IR o sensor de profundidad IR, una cámara de video color RGB, un conjunto de micrófonos en la parte inferior para la captura de comandos de voz(array de micrófonos), un acelerómetro y un motor de inclinación. Para realizar el sensado de objetos en el campo de visión, el sensor recolecta constantemente varias imágenes o frames por segundo(fps) paralelamente, correspondientes a las cámaras IR y de video. La cámara IR funciona a 30 fps y admite resoluciones de 320x240 (con 16 bits de profundidad) y 640x480 pixeles (32 bits con color), mientras que la cámara de video funciona a 30 fps en una resolución de 640x480 pixeles y a 12 fps con una resolución de 1280x960 pixeles. Así, en cada frame el emisor IR emite un patrón de puntos con distintas intensidades en 830nm, que son capturados por la cámara IR la cual se encarga de filtrar únicamente las señales IR, evitando que otros tipos de señales del entorno (tales como las señales de control remoto o luces propias de la iluminación interior), interfieran con el funcionamiento del sensor. De esta forma, la cámara IR captura la señal IR, que se representa como una imagen en escala de grises, donde cada pixel contiene la distancia Cartesiana en milímetros hacia la coordenada de ese pixel desde el dispositivo de captura. El sensado de objetos se encuentra delimitado por un rango de distancia entre 0.8 m y 0.4 m por defecto para la versión de Xbox 360, mientras que para la versión de Windows se incluye además un rango cercano de 0.4m y 3 m.

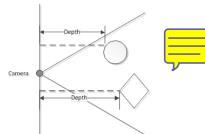


Figura 5.10: Funcionamiento del stream de profundidad

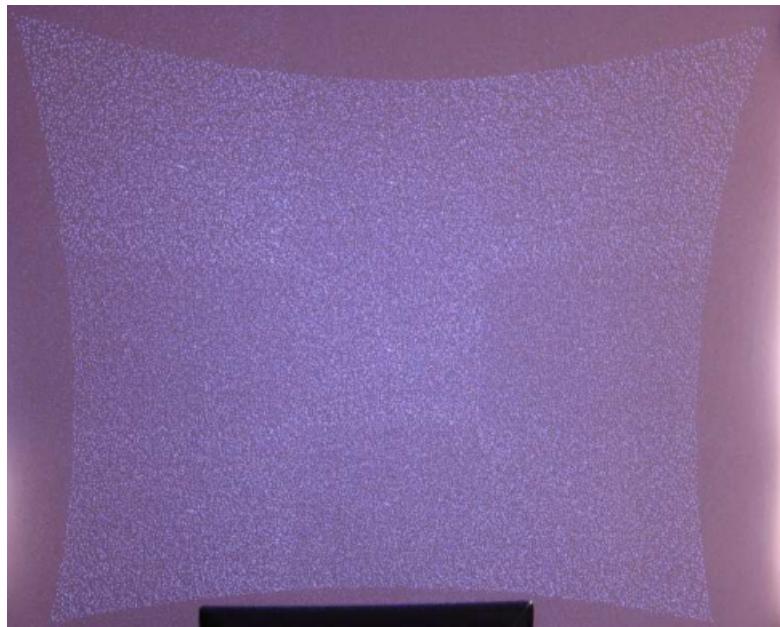


Figura 5.11: Patrón de puntos proyectados sobre una superficie

Luego, el chip de procesamiento interno del sensor analiza las diferencias entre el patrón original emitido y la información de profundidad sensada por la cámara IR, se realiza una reducción de los datos capturados y se combina esta información con los datos de la cámara RGB de video para generar la nube de puntos final.

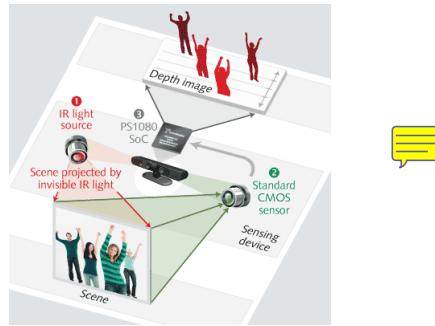


Figura 5.12: Esquema general de funcionamiento del Kinect V1

Aunque la cámara de video RGB admite una resolución mayor a la cámara IR, ésta se ajusta para combinarse con la cámara IR y producir la nube de puntos final. Adicionalmente, la cámara RGB ofrece algunas características para optimizar la calidad del video tales como balanceo de blancos automático, saturación de color, corrección de defectos y eliminación de parpadeo.



Figura 5.13: Diagrama externo del sensor Kinect V1

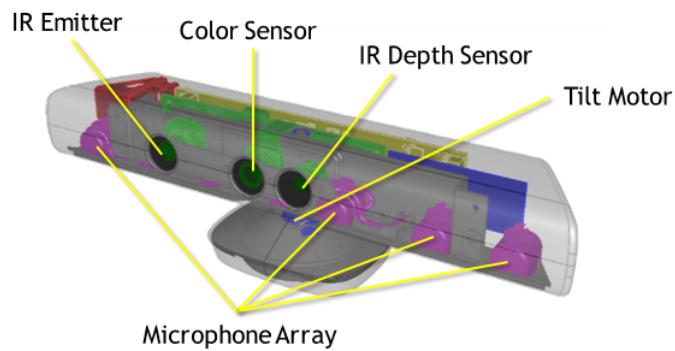


Figura 5.14: Representación externa de los componentes de hardware del sensor Kinect V1

Empleando la información de profundidad, el dispositivo ofrece la posibilidad de realizar el rastreo de esqueletos de jugadores, permitiendo rastrear las articulaciones de 2 personas completas (con esqueletos compuestos de 20 uniones principales para personas de pie y las 10 uniones pertenecientes a la cintura para personas de sentadas) y la ubicación de 4 personas adicionales.

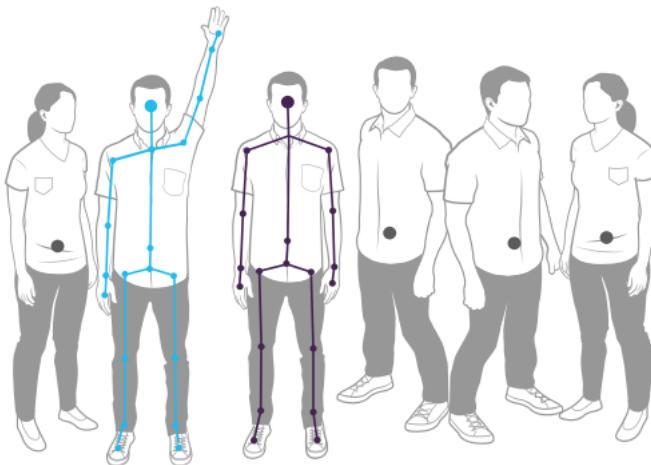


Figura 5.15: Rastreo de dos esqueletos completos (compuestos por segmentos azules y uniones, representadas como puntos azules) y 4 esqueletos parciales.

El acelerómetro del dispositivo se emplea para conocer la orientación del sensor con respecto a la gravedad, y se encuentra ubicado en el centro del dispositivo, de manera que el eje Z apunta a la dirección en la que el sensor apunta.

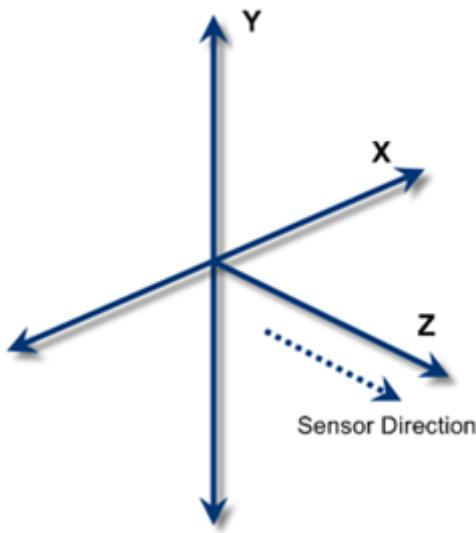


Figura 5.16: Ejes del dispositivo

Con respecto al audio captado por el dispositivo, éste detecta comandos en un rango que abarca $\pm 50^\circ$ en frente del dispositivo, pudiendo modificarse programáticamente la dirección en la que apunta el array de micrófonos en incrementos de 10° en una escala total de 100° . Además, el array de micrófonos puede cancelar 20 decibeles(dB) de ruido del ambiente frontal, mientras que el sonido que proviene desde detrás del dispositivo obtiene 6 dB más de supresión. Por defecto, el dispositivo captura los comandos hablados del jugador con el mayor nivel de decibeles.

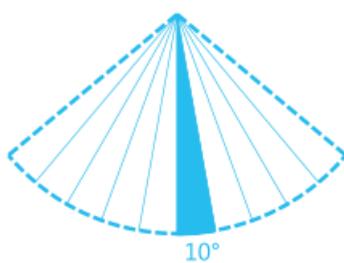


Figura 5.17: Modificación del array de micrófonos

Este sensor contiene un campo de visión de 43° horizontalmente y 57° verticalmente, que puede ser variado verticalmente a través del motor de inclinación en $\pm 27^\circ$, siendo éste el área de interacción con el dispositivo, donde se capturarán todos aquellos elementos que se encuentren en frente del sensor y no se encuentren bloqueados por algún otro objeto.

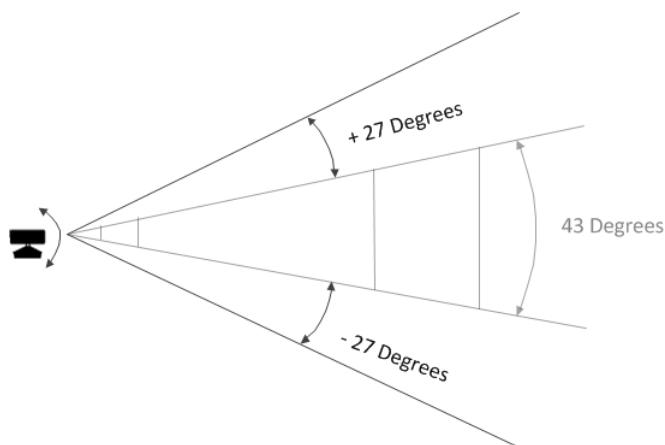


Figura 5.18: Extensión de inclinación

La versión Kinect V2, fue lanzada para Xbox One y en lugar del sensor de luz estructurada 3D desarrollada por Primesense, esta versión emplea una versión de cámara Time-of-Flight desarrollado por Microsoft, que cuenta con mayor precisión para capturar los movimientos, una resolución de video de 1920x1080 pixeles a 30 fps para la cámara de video, 512x424 pixeles a 30 fps en la cámara IR, capacidad de detección de mayor cantidad de articulaciones (ya que en Kinect V1 se podían detectar 6 cuerpos pero solo 2 con sus articulaciones completas, mientras que en esta versión se pueden capturar 6 cuerpos con sus articulaciones completas) y mayor rango de detección del jugador (con una distancia de detección entre 0.5 y 4.5 mts con software oficial).

2. Librerías para la utilización del sensor Kinect

Existen diferentes drivers y librerías que permiten interactuar con el sensor Kinect y desarrollar aplicaciones orientadas a diferentes propósitos y con distintas funcionalidad, aunque principalmente se destacan las siguientes:

- Microsoft Kinect SDK(Librería oficial)
- OpenNI
- Freenect(OpenKinect) y PCL

2.1. Kinect for Windows SDK 1.8 (Xbox Development Kit)

El Kinect SDK de Microsoft es un conjunto de librerías y herramientas que permiten programar aplicaciones en plataformas de Microsoft empleando la funcionalidad que ofrece el sensor Kinect. Esta SDK permite programar aplicaciones Windows Presentation Foundation (WPF), que es una tecnología que permite emplear los lenguajes de la plataforma .NET y el lenguaje declarativo XAML para crear aplicaciones visualmente atractivas, aplicaciones de escritorio WinForms y aplicaciones web con HTML5 (por medio de una API en Javascript que a través de un servidor configurado localmente permite la interacción o visualización desde un navegador). Esta librería funciona únicamente en plataformas Windows, requiere el framework .NET versión 4 y el IDE Microsoft Visual Studio, e incluye todos los drivers requeridos para interactuar con el sensor Kinect a través de los diferentes sensores de éste. Dentro del rango de funcionalidad que se pueden incorporar en las aplicaciones con el presente SDK se encuentran:

- Reconocimiento y seguimiento de personas por medio de esqueletos (Skeletal Tracking).
- Cálculo de la distancia entre un objeto y el sensor empleando información de profundidad.
- Captura de audio sin ruido y localización del origen de éste, e incorporación de comandos hablados a una aplicación a través de la definición de una gramática que permita el uso de voz (speech recognition).
- Reconocimiento de gestos para el ingreso de comandos con Kinect.
- Rastreo de rostros en tiempo real, obteniendo la posición y las expresiones faciales, para el uso en un avatar o comunicación con el dispositivo a través del desarrollo de una interfaz de Usuario en Lenguaje Natural (Natural User Interface, NUI).
- Utilidades para: la grabación y almacenamiento de un conjunto de frames de profundidad y color desde el Kinect con el fin de testear un escenario repetidas veces empleando Kinect Studio y la interacción en tiempo real con modelos renderizados desde el sensor Kinect con Kinect Fusion.

De esta manera, la arquitectura de esta librería se compone de los siguientes elementos:

1. Hardware del Kinect.
2. Drivers del Kinect. Los drivers para Windows del dispositivo que se instalan durante la instalación del SDK que permite acceder a la funcionalidad del array de micrófonos a través de la API de audio estándar de Windows, controles de streaming para audio, video y profundidad y funciones de enumeración para varios dispositivos para la utilización de más de un dispositivo.
3. Componentes de audio y video. Éstos son agrupados en la interfaz NUI de Kinect y permiten el acceso al stream de audio, video y profundidad.
4. Componentes DMO (DirectX Media Object) para el filtrado de sonido y el ruido (beamforming) y localización de audio.
5. APIs Estándar de Windows. APIs para el manejo de audio, speech y media.

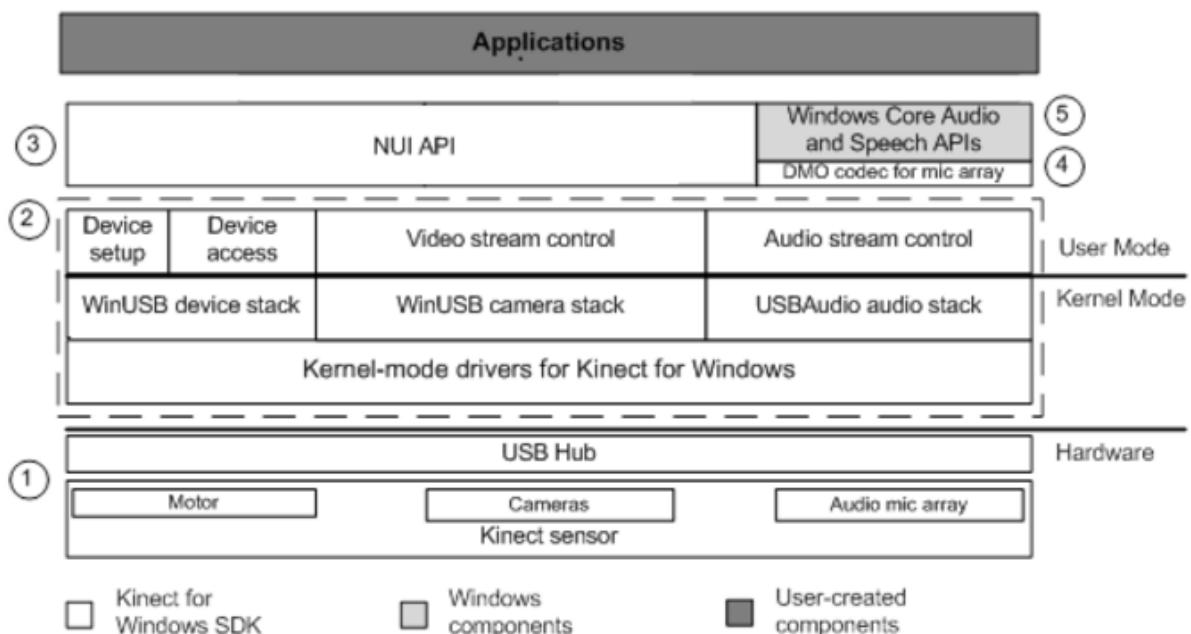


Figura 5.19: Arquitectura Kinect For Windows SDK

Entre los módulos principales de la librería se encuentran los siguientes:

- NUI.
- Kinect Interaction.
- Face Tracking.

El módulo NUI es el módulo principal del SDK y permite acceder a información de sonido, imágenes a color y profundidad capturada directamente desde el dispositivo, como así también ofrece funcionalidades que procesan esta información, tales como son: Un pipeline que permite reconocer y rastrear el cuerpo humano, el cual convierte la información de profundidad en uniones, que en conjunto representan el esqueleto del cuerpo humano, integración con la API Microsoft Speech para proporcionar un motor de procesamiento de comandos hablados que permita agregar comandos de voz a la aplicación, y la integración con la SDK Face Tracking para reconocimiento de expresiones faciales. De esta forma, para que las aplicaciones interactúen con el sensor Kinect, el módulo define una clase principal `KinectSensor` que representa el sensor y que agrupa cada conjunto de frames de video, profundidad y skeletons en streams, que obtienen de manera continua información del dispositivo, y que deben ser habilitados y configurados por el desarrollador de manera explícita para comenzar con el sensado. Así, el flujo de trabajo para la obtención de información con la librería consiste en:

1. Seleccionar un dispositivo Kinect. Esto se realiza por medio de iteración de la colección `Kinect.KinectSensors` que agrupa todos los dispositivos conectados y permite obtener el nombre y el estado del dispositivo (si se encuentra conectado y funcionando correctamente).
2. Luego de seleccionar el dispositivo, se deben habilitar los streams de los que se desee obtener información, invocando para ésto al método `enable()` de cada stream, que recibe la configuración que especifica el formato de los datos de imagen, la tasa de frames y la resolución de los pixeles de datos, definida como un tipo enumerado en las clases de formato para cada stream. Los streams para frames de imágenes a color, profundidad skeleton se encuentran definidos en las clases `ColorStream`, `DepthStream` y `SkeletonStream`, respectivamente.
3. Posteriormente, se debe iniciar la recolección de datos desde el sensor con el método `start()`.
4. Para la obtención de frames, la aplicación obtiene el último frame (color o profundidad) invocando a un método del stream habilitado y lo copia a un buffer si ésta es disponible, o si no lo es, puede retornar inmediatamente o esperar el siguiente frame. Para la obtención de frames el SDK proporciona dos modelos diferentes: modelo por consulta (polling) o modelo de eventos; El modelo

por consulta consiste en que al momento de solicitar el siguiente frame se especifique una cantidad fija de milisegundos, de manera que se retorne el control a la aplicación cuando el siguiente frame este disponible o cuando el tiempo de espera expire. Mientras que en el modelo por eventos, se definen eventos separados para cada tipo de stream y handlers que reciben el frame del tipo de dato asociado al stream.

- Finalmente, se finaliza la captura de información desde el sensor con el método stop() de KinectSensor.

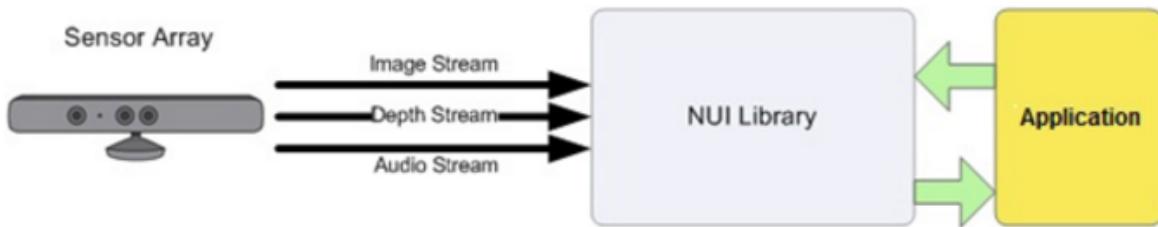


Figura 5.20: Interacción de sensor Kinect y aplicación desarrollada por usuario

Por otro lado, el módulo KinectInteraction es un módulo que emplea una combinación del stream de profundidad, stream de esqueleto y algoritmos complejos con el fin de proporcionar a las aplicaciones la capacidad de incorporar la interacción con el usuario por medio de gestos a través de una mano principal (se realiza el seguimiento de ambas pero solo una se emplea para controlar la interacción), efectuando detección y rastreo de la posición y estado de la misma, y brindando la posibilidad de registrar los siguientes gestos del usuario:

- Agarrar y liberar un elemento (Grip and Release), donde el agarre consiste en mantener la mano abierta enfrente del sensor y luego hacer un puño con la mano, mientras que liberar es la apertura del puño cerrado.
- Detección de gesto presionar(Press), en el que el usuario mantiene su mano abierta enfrente su palma con el sensor y mantiene sus brazos parcialmente extendidos, para luego extenderlos hacia el dispositivo.
- Información respecto del control virtual que manipula el usuario con su mano principal. Esta información se obtiene por medio de un stream de interacción, similar al resto de los streams, que brinda frames que pueden ser procesados para proporcionar información en la interacción del usuario con la aplicación, tales como la posición de la mano y el estado actual (presionando, agarrando, o liberando) y el control que el usuario está empleando.

Este módulo define una API nativa en C++ y una API en C# que brinda las características de identificación de usuario, estado y rastreo de la mano, así también incluye un data stream de interacción (interaction stream), similar al resto de los streams, que permite obtener frames respecto de la interacción del usuario con la aplicación (posición y estado de la mano). Adicionalmente, este módulo define controles en C# para aplicaciones WPF que pueden ser empleados para construir aplicaciones interactivas, tales como son listas scrolleables, botones que responden a los gestos y regiones interactivas.

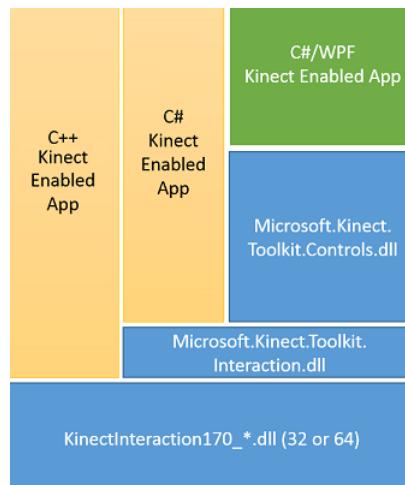


Figura 5.21: API en modulo Kinect Interaction

El módulo de Face Tracking SDK utiliza información de los streams de color y de profundidad para deducir la posición de la cabeza y las expresiones faciales, para proporcionar a la aplicación esta información. La calidad de rastreo de rostros depende de la calidad de los frames de entrada de estos streams, por lo que frames más difusos u oscuros serán rastreados con un rendimiento menor que los frames más brillantes o nítidos. El flujo de trabajo con esta API, consiste en crear un objeto principal IFFaceTracker para la obtención de frames, invocar al método de obtención de nuevos frames de este objeto y procesar los mismos dentro de un bucle, hasta que por alguna condición de corte no se desee continuar con el procesamiento. Esta interfaz proporciona de las siguientes clases para realizar el seguimiento de rostros:

- IFFaceTracker. Esta es la interfaz principal a través de la cual se leen los frames, por medio de los métodos startTracking() para la inicialización del objeto y la determinación de orientación del sensor y, continueTracking() que emplea información anterior de startTracking() o continueTracking() para sucesivas llamadas y se almacenan en un buffer de tipo FT_SENSOR_DATA.
- IFTResult. Esta clase contiene información respecto del resultado de la operación de obtención de frames.
- IFTImage. Esta clase define los diferentes formatos admitidos para la imagen capturada por el sensor, mantiene buffers para almacenar los datos de la imagen y permite acceder a la información propia de ésta (pixels, width, height, etc.).
- IFTModel. Esta clase permite invocar a métodos para convertir la información capturada a mallas 3D de vértices.

2.2. OpenNI

OpenNI framework es un SDK open-source empleado para el desarrollo de librerías y aplicaciones de sensado 3D, que ofrece un rango variado de herramientas para la colaboración y la promoción del software desarrollado, brindando una plataforma de marketing para descargar y compartir software en la comunidad OpenNI. La librería OpenNI proporciona acceso a los dispositivos desarrollados por PrimeSense y Asus Xtion y permite acceder a las imágenes de profundidad y RGB, y streams IR. El flujo de trabajo con OpenNI consiste en emplear la clase principal openni::OpenNI, que engloba todos los dispositivos conectados del sistema y agrupa los eventos de conexión y desconexión de dispositivos, para inicializar todos los dispositivos por medio de OpenNI::initialize(), enumerar todos los dispositivos con OpenNI::enumerateDevices(), leer y procesar información del dispositivo conectado y finalmente, terminar la ejecución con OpenNI::shutdown(). Luego de la inicialización, se puede acceder al dispositivo por medio de las siguientes clases principales:

- openni::OpenNI. Es la clase principal para acceder a los dispositivos conectados, eventos relacionados a conexión y desconexión de dispositivos, información de versión de la API y errores.

- `openni::Device`. Esta clase representa un dispositivo conectado al sistema y requiere que OpenNI se haya inicializado antes de que pueda ser generada una instancia. Esta clase contiene métodos para habilitar el flujo de streams del dispositivo, obtener información del dispositivo, revisar el estado de la conexión y realizar sincronización de frames, para aquellos dispositivos que cuentan con cámaras de video y de profundidad, donde puede que la tasa de frames entre ambos esté desfasada.
- `openni::VideoStream`. Representa todos los streams de datos obtenidos desde un dispositivo y se emplea para obtener objetos de tipo `VideoFrameRef`. Esta clase permite habilitar, deshabilitar y configurar el stream de datos (framerate, resolución y tipo de pixel) y la lectura de frames puede realizarse ya sea empleando un modelo de polling o un modelo dirigido por eventos.
- `openni::VideoFrameRef`. Abstacta los datos y los metadatos de un frame leído desde un stream. Permite acceder a tamaño de los datos, resolución del frame, timestamp, tipo de sensor y datos del frame (Array Stride).

2.3. Freenect y Librería Point Cloud Library(PCL)

Frennect (Libfreenect) es un driver multiplataforma, de código abierto para el sensor Kinect disponible para Windows, Linux y OS X, que incluye todo el código necesario para inicializar, activar y comunicarse con el sensor Kinect y es desarrollado por la comunidad abierta OpenKinect (que cuenta con más de 2000 miembros) y cuyo interés radica en explotar las capacidades del sensor sobre diversas plataformas. Libfreenect se encuentra disponible tanto para la versión 1 de Kinect (en Xbox 360) y 2.0 (Xbox One). Este driver permite acceder a imágenes con RGB y profundidad, motor, acelerómetro, led y audio del y proporciona el siguiente conjunto de wrappers para los lenguajes de programación:

- Python
- C, C++ y C#
- Java JNI, Java JNA
- Common Lisp
- Actionscript

Adicionalmente, libfreenect brinda las utilidades de prueba Record y Fakenect: La primera permite grabar una secuencia de frames del dispositivo en disco volcando las lecturas de los streams de video, profundidad y acelerómetro, mientras que la segunda se enfoca en permitir leer las grabaciones hechas por Record, de manera que  sea necesario contar con un sensor conectado para realizar pruebas.

Por otro lado, PCL es un proyecto que comenzó en 2010 por Willow Garage (compañía desarrolladora de la librería de imágenes OpenCV) y de la compañía desarrolladora de Robotic Operating System(ROS), cuya primera versión fue oficialmente liberada en 2011. Point Cloud Library(PCL) es una librería independiente, de código abierto, multiplataforma (disponible para Linux, Windows, MacOS, y Android/iOS), escrita en C++, para la captura, el procesamiento geométrico y almacenamiento de nubes de puntos 2D/3D. Aunque esta librería fue pensada para desarrollar en C++, también existe un binding para Python que incluye el siguiente conjunto de funciones: Entrada/Salida de archivos PCD, segmentación, suavizado, filtrado y registración. Esta librería ofrece algoritmos vinculados a tareas relacionadas a la visión artificial (o visión por computadora), que es un área de la inteligencia artificial, donde se busca que una computadora obtenga información y logre un entendimiento de alto nivel de las propiedades de ésta (tales como formas, iluminación, distribución de colores) a partir de un video o imagen del mundo real. Esta disciplina incluye aquellos métodos que permiten adquirir, analizar, procesar y extraer datos que puedan ser convertidos a información numérica y simbólica que pueda ser de utilidad durante la automatización de una tarea. Dentro del rango de aplicaciones en las que se emplea la visión artificial las más comunes son las siguientes:

- Reconocimiento óptico de caracteres(OCR) interpretando códigos escritos a mano.
- Inspección de máquinas, asesorando la calidad de partes empleando estéreo visión con iluminación especializada para medir tolerancias en partes de dispositivos aéreos o de automóviles.

- Seguridad automotriz, detectando obstáculos como peatones en los senderos viales, bajo condiciones donde las técnicas de visión activas como Lidar no funcionan correctamente.
- CGI (computer-generated imagery) en Cine-TV, donde la filmación real con actores se une con imágenes generadas por computadora rastreando puntos clave en el video origen, con el fin de estimar el movimiento de la cámara y la forma del entorno.
- Captura de movimiento, utilizando marcadores retro-reflectivos capturados desde distintas cámaras con el objetivo de capturar digitalmente el patrón de movimiento de actores para realizar una animación por computadora.
- Reconocimiento de huellas digitales para el acceso de personal autorizado automatizado.

De esta forma, PCL es una librería que ofrece diferentes módulos independientes que pueden ser combinados de distintas formas en un pipeline de instrucciones, con el fin de lograr el reconocimiento de distintos tipos de objetos en una nube de puntos. Los algoritmos de estos módulos están pensados para abarcar un diverso rango de tareas que son necesarias para una correcta detección de objetos, tales como filtrado de puntos con valores atípicos distantes del resto en una nube (outliers en la nube), almacenamiento, lectura y conversión de nubes de puntos en distintos formatos, descomposición de la nube para realizar búsquedas, concatenar y fusionar dos nubes de puntos con los mismos o distintos campos, segmentar partes de una escena, extraer puntos clave y computar descriptores geométricos con el propósito de distinguir elementos del mundo real. De manera general, el pipeline de PCL para el reconocimiento de objetos se compone de las siguientes etapas:

- Pre-procesamiento de nube: Durante esta etapa se elimina el ruido de la nube previamente capturada, se aplican algoritmos para estructurarla y se estiman features que proporcionan información acerca de las características de la superficie que serán empleadas durante las siguientes etapas.
- Segmentación de objetos: En esta etapa se realiza la segmentación por medio de distintas técnicas con el fin de obtener clusters de interés, que serán utilizados para generar descriptores.
- Generación de descriptores: Durante esta fase, se computan los descriptores para el/los cluster/s aislados. Un descriptor es una estructura compleja que codifica información respecto de la geometría que rodea un punto, de manera que permiten identificar un conjunto de puntos a lo largo de varias nubes de puntos, sin importar el ruido, la resolución o las posibles transformaciones. Adicionalmente, algunos descriptores capturan información global respecto del objeto al que pertenecen, como el punto de visión que puede ser utilizado para computar la posición.

A continuación, se enumeran y describen los algoritmos principales empleados durante cada fase.

2.3.1. Algoritmos de pre-procesamiento de nubes

Representación y almacenamiento de una nube de puntos

Con respecto al almacenamiento persistente de nubes de puntos, aunque éste se puede realizar en diversos formatos, PCL cuenta con su formato PCD (Point Cloud Data file) personalizado definido para complementar el resto de los formatos, donde no se soportan algunas características de procesamiento geométrico, estando disponible en dos versiones: Binaria y ASCII. Cuando este formato se encuentra en versión ASCII, es posible inspeccionarlo con un editor de textos para analizar los datos relacionados a la nube de puntos. Internamente, éste se compone por un encabezado donde se almacena información respecto de la información en la nube de puntos y un cuerpo que contiene las coordenadas 3D, y opcionalmente información RGB o RGBA, para cada punto de la captura. En el encabezado de la nube se encuentran los siguientes datos:

- VERSION, que especifica la versión de la librería PCL.
- FIELDS, que indica los atributos a través de los que se indicarán las coordenadas de cada punto, e información necesaria para el procesamiento de éstos (tales como información de color, normales, etc.). Este valor es una tupla de cadenas separadas por espacios, entre las que se encuentran: x y z | x y z rgb | x y z normal_x normal_y normal_z.

- SIZE, especifica el tamaño en bytes según el tipo de dato que se utilice para representar los valores de cada dimensión descrita por FIELDS, siendo este de 8 para valores tipo double, 4 si cada dimensión se representa por valores int o float, 2 para tipos short no signados y de 1 byte para tipos char sin signo.
- TYPE, que indica el tipo de cada dimensión descrita por FIELDS, siendo I para tipos enteros, U para tipos sin signo y F para valores flotantes.
- COUNT, especifica cuantos elementos tiene cada dimensión, siendo este valor de 1 para nubes que solamente contendrán datos y del tamaño del descriptor para nubes que se representan como descriptores.
- WIDTH, indica la longitud del conjunto de puntos que componen la nube de puntos, adoptando distintos valores dependiendo de la forma de organización de la nube, siendo las posibles formas: Organizada o Desorganizada. Una nube de puntos organizada, es aquella donde internamente los puntos en la nube se organizan en filas y columnas como en una matriz, mientras que en una nube desorganizada todos los puntos se organizan en una única fila. Por ejemplo, si este campo se definiera como *WIDTH 640*, significaría que los puntos que componen la nube se encuentran estructurados en filas con 640 puntos por fila.
- HEIGHT, indica la cantidad de filas que contiene la nube de puntos, siendo 1 para las nubes de puntos desorganizadas y un valor numérico para las nubes de puntos organizadas.
- VIEWPOINT, representa el punto de visión desde el que fueron adquiridos los puntos de la nube, que puede posteriormente ser empleado en descriptores que requieren orientación. Este campo se especifica como una traslación (traslacion_x, traslacion_y, traslacion_z) mas las unidades de cuaternion (o versores) que representan valores numéricos respecto de las rotación y orientación del sensor.
- POINTS, especifica el número total de puntos en la nube.
- DATA, indica por medio de una cadena de caracteres el formato en que la nube de puntos es almacenada, siendo los posibles valores ascii o binary.

Por otro lado, el cuerpo de la nube en formato ascii contiene las coordenadas de cada punto, junto con la información adicional, representándose éstas como una secuencia lineas separadas por caracteres de nueva linea, mientras que si se almacenan en formato binario, la cabecera y el cuerpo son una copia del arreglo de puntos en memoria a disco.

De esta forma, un ejemplo de archivo PCD en formato ascii, para una nube no estructurada con coordenadas e información de color se define de la siguiente manera:

```
# .PCD v.7 - Point Cloud Data file format VERSION .7 FIELDS x y z rgb SIZE 4 4 4 4
TYPE F F F F COUNT 1 1 1 1 WIDTH 213 HEIGHT 1 VIEWPOINT 0 0 0 1 0 0 0 #Valor por defecto
POINTS 213 DATA ascii 0.93773 0.33763 0 4.2108e+06 0.90805 0.35641 0 4.2108e+06 ...
```

Opcionalmente PCL ofrece los siguientes formatos para almacenamiento de nubes de puntos diseñados por distintas organizaciones para ser empleados por distintos programas:

- OBJ: Es un formato de archivo geométrico desarrollado por Wavefront Technologies, que representa la geometría específicamente de un objeto, detallando la posición de cada vertex, las coordenadas de las texturas y normales asociadas a éstos, y las caras que forman cada polígono.
- PLY: Polygon File Format es un formato donde se almacenan un conjunto de polígonos que representan un objeto o superficie 3D, que puede contener información respecto de color y transparencia, normales, texturas de las coordenadas y valores de confianza para éstas. Este formato, permite almacenar distintas propiedades para las caras frontales y traseras de los polígonos y puede ser almacenado tanto en formato ascii o binario.
- STL: Es un formato nativo para el software de diseño y prototipado 3D de modelos, que pueden ser aceptados como entrada en impresoras 3D. Este tipo de archivo representa un objeto como un conjunto de triángulos no ordenados, describiéndola a través de las normales y los vértices que lo componen en un sistema Cartesiano. Este archivo se puede almacenar en formato ascii y binario.

La representación en PCL de las nubes de puntos en memoria, se realiza por medio de la creación de instancias de la clase de pcl::PointCloud por cada nube leída, que almacena las coordenadas

de los puntos que componen un objeto como un vector (`std::vector`) y encapsula el comportamiento propio de nube de puntos, que puede ser necesario al momento de iterar, concatenar o acceder a puntos de ésta, tal como es la solicitud de la cantidad total de puntos en ésta. La clase `PointCloud` es una clase template (definida como `PointCloud<PointT>`) con respecto a los tipos de puntos, lo que significa que se utiliza el comportamiento y la estructura de esta clase base para generar instancias de nubes de puntos con distintos tipos de puntos. Los tipos de puntos en PCL se emplean para representar tanto las coordenadas y/o atributos agregados (Normales, BoundaryPoints, etc.) de un objeto 3D como así también los descriptores; De esta forma, la clase base que representa una coordenada de una nube de puntos es `pcl::PointXYZ` para una coordenada 3D y `pcl::PointXY` para una coordenada en un espacio 2D, y dependiendo de la información adicional que se agregue a una coordenada, se incluye el nombre de esta característica como parte de la nomenclatura de la coordenada base. Así, por ejemplo si se desean emplear coordenadas que contengan información espacial y agregar información de color, se deberá emplear la clase `pcl::PointXYZRGB`, o si por el contrario se desea emplear alguna característica de un punto sin incluir sus coordenadas, se emplea el nombre que PCL emplee para nombrar a esta característica, por ejemplo si se desea emplear únicamente normales se debe emplear el tipo de punto `pcl::Normal`. Para los tipos de puntos que se corresponden con descriptores (explicado en la sección Algoritmos para generación de descriptores), el tipo de punto se define como el nombre del descriptor, la palabra `Signature` y el tamaño de éste, siendo ejemplos de tipos de puntos asociados a descriptores los siguientes: `FPFHSignature33`, `PFHSignature125`, `VFHSignature308`, etc.

Lectura y escritura de nubes de puntos

Con respecto a la lectura y escritura de nubes de puntos, éstas se realizan por medio del módulo `pcd_io` especificando el tipo de punto que se leerá/escribirá de una nube determinada. Para la lectura de nubes de puntos, se deben importar los tipos de puntos y el módulo `io`, luego definir una nube de puntos para el tipo de punto e invocar al método `loadPCDFile()` que aceptará una cadena con el path completo de la nube como primer argumento y la nube definida anteriormente como parámetro de salida:

```
#include <pcl/io/pcd_io.h>
#include <pcl/point_types.h>
...
pcl::PointCloud::Ptr cloud(new pcl::PointCloud());
if (pcl::io::loadPCDFile<pcl::PointXYZ>(argv[1], *cloud) != 0)
{
    return -1;
}
...
```

Con respecto a la escritura de nubes, esta consiste en definir la nube de salida e invocar al método de guardado que toma el nombre del archivo PCD de salida y la nube con el contenido previamente leído, siendo `savePCDFileASCII()` para almacenar ésta en formato ascii o `savePCDFileBinary()` para modo binario:

```
pcl::PointCloud::Ptr cloud(new pcl::PointCloud());
// Se carga o copia la nube a la variable cloud
...
pcl::io::savePCDFileASCII("output.pcd", *cloud);
```

Visualización de nubes de puntos

PCL ofrece la herramienta de línea de comandos `pcl_viewer` para la visualización de nubes de puntos, que cuenta con la capacidad de abrir varias nubes simultáneamente superponiéndolas de manera ordenada y obtener y visualizar características relevantes ésta, tales como mostrar los ejes

Cartesianos (X,Y,Z), obtención manual de coordenadas a partir de una selección, rotación de nube de puntos, modificación de los puntos que representa ésta, visualización de curvaturas principales y de normales. Esta herramienta emplea la clase `pcl::visualization::PCLVisualizer` del módulo `visualization` y puede ser utilizada para implementar un visualizador propio. Adicionalmente, se puede emplear la clase `CloudViewer` para crear un visualizador con menos funciones, pero más sencillo de configurar y que proporciona una ventana y herramientas de zoom y rotación.

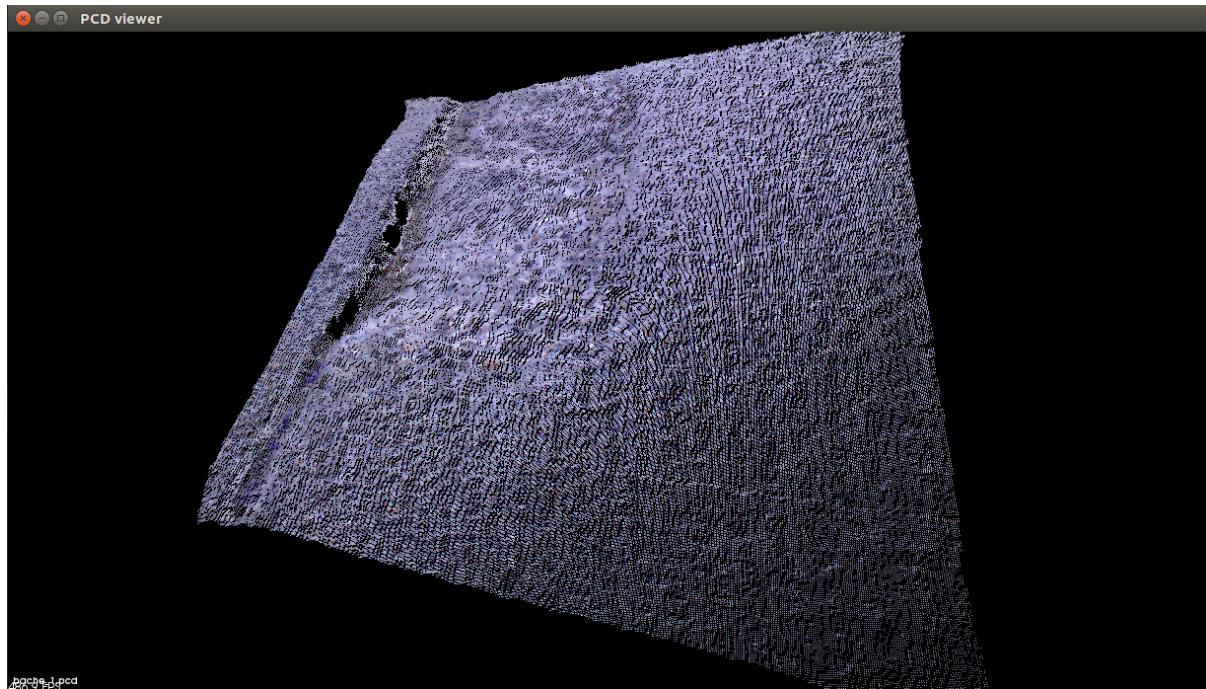


Figura 5.22: Ejemplo de visualizador de PCL

Computacion de Bounding Boxes(AABB-OBB)

La bounding box mínima (MBB) para el conjunto de puntos que componen un objeto, se considera el rectángulo (o box) formado por las coordenadas mínimas necesarias para contener todos los puntos objeto dentro de ésta y cuya medida (área, volumen o hypervolumen en espacios de más de tres dimensiones) es la mínima. La bounding box de un objeto puede ser computada a través del convex hull, ya que si se dispone de éste, la bounding box mínima para los puntos del objeto es la misma que la bounding box del convex hull.

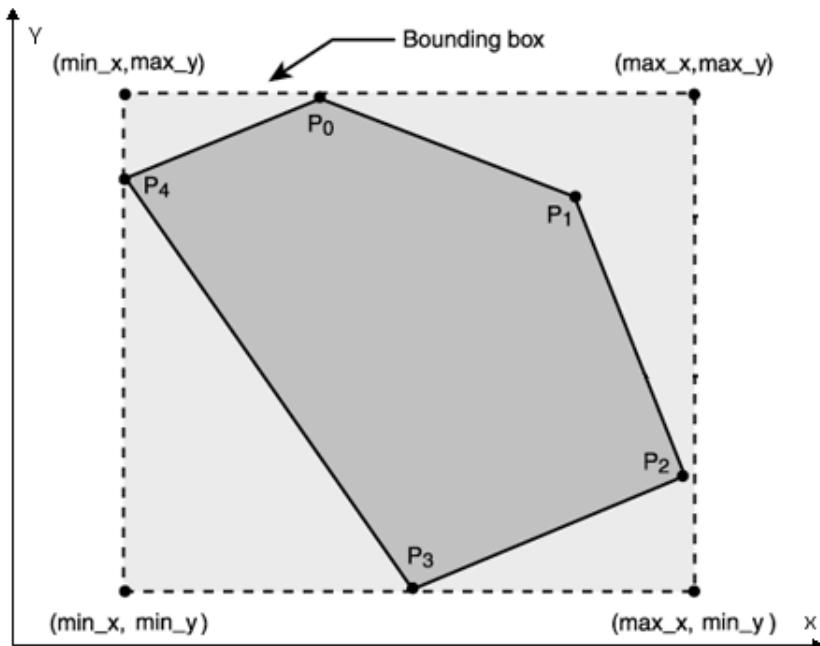


Figura 5.23: Ejemplo de conjunto de puntos de un objeto (P₀-P₄) con su convex hull y el bounding box asociado.

Existen varios tipos de Bounding Box dependiendo del sistema de coordenadas que se considere: Axis Aligned Bounding Box (AABB) y Oriented Bounding Box (OBB). AABB consiste en computar las coordenadas de la caja de manera que los bordes de ésta, sean paralelos a la orientación de los ejes Cartesianos globales en la captura, donde el origen se encuentra en el centro de la escena. Por otro lado, OBB consiste en computar las coordenadas de la caja tomando como referencia un sistema de coordenadas Cartesiano local del objeto, cuya orientación y origen se encuentran definidos en base al centro de los puntos que componen el objeto, de manera que la bounding box se encontrará paralela a los ejes del objeto en sí.

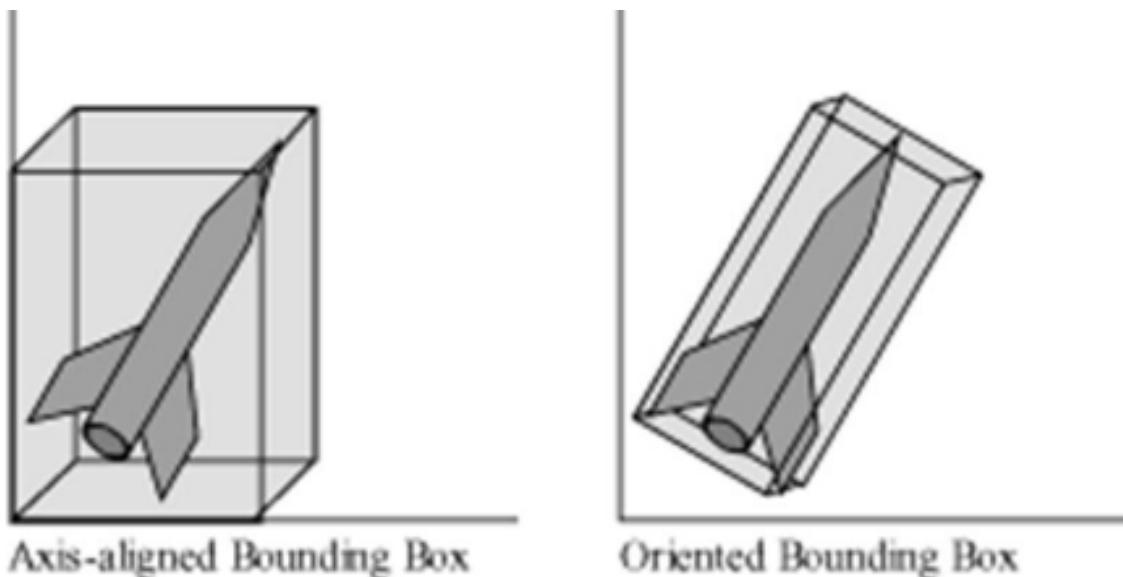


Figura 5.24: AABB vs OBB

En PCL este comportamiento se implementa en la clase `pcl::MomentOfInertiaEstimation`, que a partir de una nube de entrada permite obtener el centro del objeto y los puntos máximos y mínimos en

los ejes X,Y,Z de las boxes AABB y OBB. A continuación se muestra un ejemplo de código donde se obtienen las respectivas boxes:

```
...
// Se instancia el objeto y se envia la nube 'cloud' de entrada inicializada previamente
pcl::MomentOfInertiaEstimation <pcl::PointXYZ> feature_extractor;
feature_extractor.setInputCloud (cloud);
feature_extractor.compute ();

...
// Se definen los puntos máximos y mínimos para AABB y OBB
pcl::PointXYZ min_point_AABB;
pcl::PointXYZ max_point_AABB;
pcl::PointXYZ min_point_OBB;
pcl::PointXYZ max_point_OBB;
pcl::PointXYZ position_OBB;

// Se obtienen los puntos
feature_extractor.getAABB (min_point_AABB, max_point_AABB);
feature_extractor.getOBB (min_point_OBB, max_point_OBB, position_OBB, rotational_matrix_OBB);

// Se instancia un visualizador y se agregan los puntos de ambas cajas
boost::shared_ptr<pcl::visualization::PCLVisualizer> viewer (new
pcl::visualization::PCLVisualizer ("3D Viewer"));

viewer->addCube (min_point_AABB.x, max_point_AABB.x, min_point_AABB.y, max_point_AABB.y, min_
point_AABB.z, max_point_AABB.z, 1.0, 1.0, 0.0, "AABB");
viewer->addCube (position, quat, max_point_OBB.x - min_point_OBB.x, max_point_OBB.y - min_
point_OBB.y, max_point_OBB.z - min_point_OBB.z, "OBB");
...
```

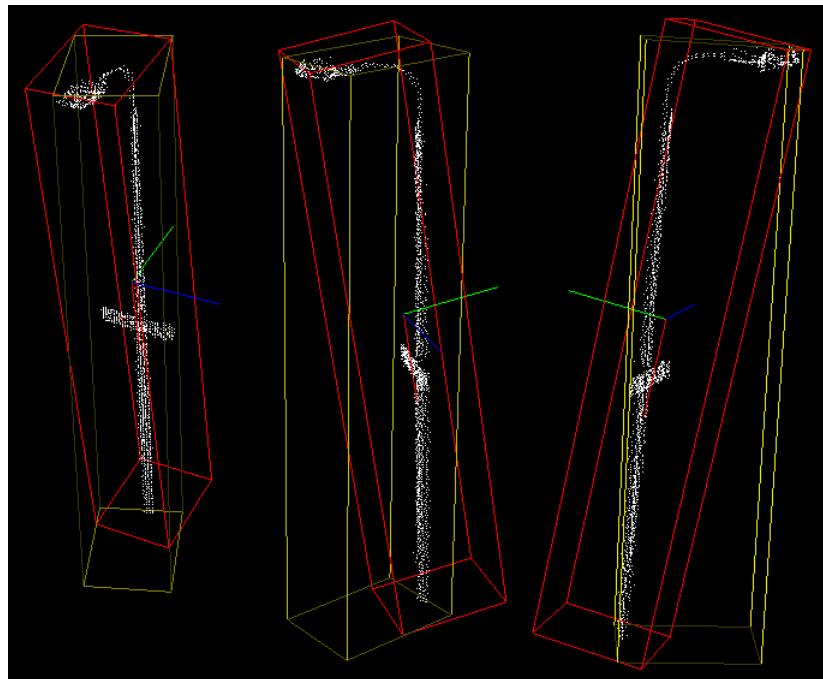


Figura 5.25: Ejemplo de poste de luz con AABB en amarillo y OBB en rojo.

Computación de índices

Algunos de los algoritmos de PCL retornan índices, éstos contienen la posición del punto dentro del vector de puntos que mantiene el objeto PointCloud, sin incluir la información completa de cada punto.

Esto permite computar los índices de puntos de interés (o su complemento) que sean relevantes para una operación determinada y, posteriormente, copiarlos a otra nube, reduciendo la cantidad de puntos a procesar. De esta manera, para extraer los índices se emplea la clase pcl::ExtractIndices, que a partir de algún algoritmo aplicado a una PointCloud que proporciona los índices de los puntos filtrados (en una estructura pcl::PointIndices) y la nube de puntos original, permite el filtrado de la información completa de los puntos. Por ejemplo, la segmentación permite obtener los indices de los puntos pertenecientes a un cluster segmentado. A nivel de código fuente la estructura general es la siguiente:

```
// Objeto para almacenar la nube procesada anteriormente
pcl::PointCloud<pcl::PointXYZ>::Ptr cloudProcesada(new pcl::PointCloud<pcl::PointXYZ>);
...

// Índices obtenidos
pcl::PointIndices::Ptr pointIndices(new pcl::PointIndices);

// Variable para almacenar los puntos extraídos a partir de los índices
pcl::PointCloud<pcl::PointXYZ>::Ptr nubeExtraida(new pcl::PointCloud<pcl::PointXYZ>);

pcl::ExtractIndices<pcl::PointXYZ> extract;
extract.setInputCloud(cloudAll);
extract.setIndices(pointIndices);
extract.filter(*cloudExtracted);
```

Remover valores NaN

Durante la captura de nubes de puntos pueden existir inconsistencias en los valores de las coordenadas para determinados puntos debido a problemas de posicionamiento con el sensor o por características de la superficie, estos valores se representan en PCL como NaN. Los valores NaN (Not a Number) son valores numéricos flotantes que no pueden ser representados o que son indefinidos y que, si son utilizados en otros algoritmos de PCL, provocarán un fallo. Por este motivo, los valores NaN deben ser removidos antes de la aplicación de los algoritmos de PCL a una nube de entrada (si el algoritmo en cuestión no ofrece esta funcionalidad), teniendo en cuenta que al eliminar los valores NaN de una nube, esta cambiará su tamaño, por lo que si es necesario que la nube se mantenga organizada será necesario reorganizarla con la cantidad de puntos filtrados. Para realizar esta tarea PCL ofrece la función pcl::removeNaNFromPointCloud() que acepta la nube de entrada, de salida y un mapping (que es un vector de enteros) que permite identificar que punto de la nube original, se corresponde con que punto de la nube filtrada.

```
# .PCD v0.7 - Point Cloud Data file format VERSION 0.7 FIELDS x y z rgba SIZE 4 4 4 4
TYPE F F F U COUNT 1 1 1 1 WIDTH 640 HEIGHT 480 VIEWPOINT 0 0 0 1 0 0 0 POINTS 307200
DATA ascii nan nan nan 10135463 nan nan nan 10398635
```

A continuación se representan las instrucciones básicas para realizar el filtrado de la nube de puntos de entrada:

```
//Definición de la nube de puntos
pcl::PointCloud<pcl::PointXYZ>::Ptr cloud(new pcl::PointCloud<pcl::PointXYZ>);

//Pasos realizados para leer la nube de puntos de disco
...

//Definición del objeto mapping y aplicación del método para remover NaN
std::vector<int> mapping;
pcl::removeNaNFromPointCloud(*cloud, *cloud, mapping);
```

Descomposición de nubes: KD-Tree y Octree

La descomposición de nubes de puntos consiste en organizar la nube de puntos en una estructura de manera que el filtrado y análisis del entorno de los mismos (búsqueda de vecinos más cercanos, búsqueda de vecinos en un radio determinado o, el punto más cercano, etc.) sea más eficiente. Para conseguir esto, PCL ofrece dos tipos de estructuras: Kd-Tree y Octree. La estructura Kd-Tree es un árbol binario que organiza un conjunto de puntos en un espacio K-dimensional, estando determinada la cantidad de dimensiones por los ejes utilizados para definir las coordenadas de cada punto en la nube. De esta forma, si se emplean nubes de puntos tridimensionales, el árbol kd-tree organizará los puntos por medio de divisiones en los ejes X,Y,Z. En esta estructura, cada nodo representa un punto de la nube y cada nivel del árbol es una separación de puntos en alguna de las dimensiones. Así, en un espacio tridimensional la división comienza por crear el nodo raíz del árbol que divide los puntos respecto al eje X en base a un criterio (típicamente la raíz de cada subárbol es el punto medio del conjunto de coordenadas en ese eje), creando un nodo izquierdo que representa al subárbol de los puntos cuyo valor de X sea menor y un nodo derecho para el subárbol de los valores mayores; posteriormente, se realiza la división de puntos en el espacio Y para los nodos hijos del nodo raíz empleando el mismo procedimiento y para el espacio Z con los hijos de la división en Y. Al llegar al eje Z, se repite nuevamente todo el proceso para continuar subdividiendo el espacio hasta que no existan puntos para continuar la división.

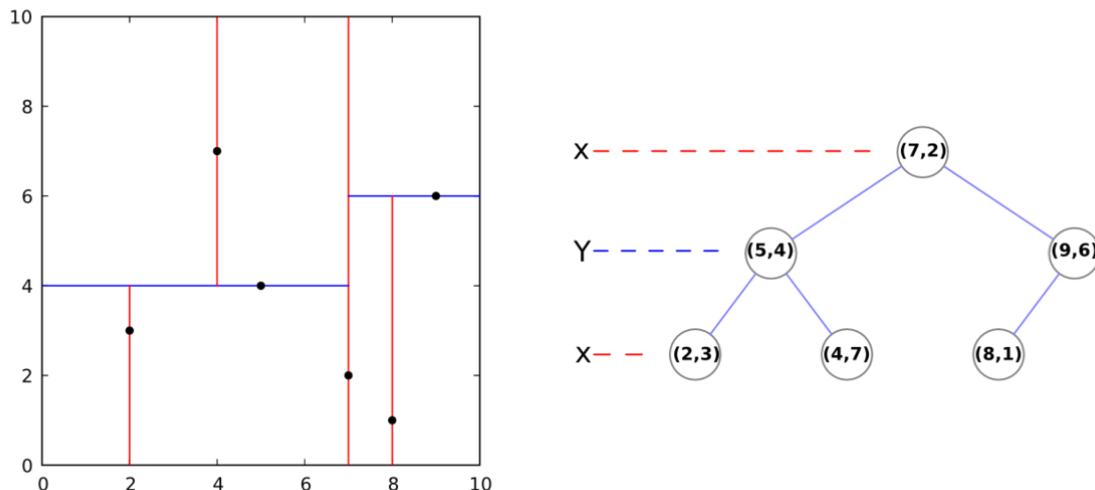


Figura 5.26: Ejemplo de división en un espacio 2D, donde los puntos iniciales se encuentran marcados en negro, las divisiones en X en rojo y las divisiones en Y en azul.

En PCL la clase `pcl::KdTree` implementa este comportamiento para los distintos tipos de puntos y permite realizar la búsqueda por cantidad de vecinos más cercanos (`pcl::KdTree::nearestKSearch()`) o por radio de búsqueda (`pcl::KdTree::radiusSearch()`). A continuación, se muestra una porción de código donde se realiza una búsqueda por cantidad de vecinos cercanos a un punto dado:

```
// Objeto que almacena la pointcloud
pcl::PointCloud<pcl::PointXYZ>::Ptr cloud(new pcl::PointCloud<pcl::PointXYZ>);

// Lectura del archivo .pcd de disco
if (pcl::io::loadPCDFile<pcl::PointXYZ>(argv[1], *cloud) != 0)
{
    return -1;
}

// Objeto kd-tree que representa a la pointcloud instanciada
pcl::search::KdTree<pcl::PointXYZ> kdtree;
kdtree.setInputCloud(cloud);
```

```

// Se define la búsqueda de los 5 vecinos mas cercanos a un punto aleatorio
pcl::PointXYZ point;
point.x = 0.0524343;
point.y = -0.58016;
point.z = 1.776;
// Vector para almacenar los indices de los puntos filtrados
std::vector<int> pointIndices(5);

// Distancias cuadradas de los vecinos obtenidos hasta el punto aleatorio
std::vector<float> squaredDistances(5);

// Se realiza la búsqueda y se imprimen resultados
if (kdtree.nearestKSearch(point, 5, pointIndices, squaredDistances) > 0)
{
    std::cout << "5 nearest neighbors of the point:" << std::endl;
    for (size_t i = 0; i < pointIndices.size(); ++i)
        std::cout << "\t" << cloud->points[pointIndices[i]].x
            << " " << cloud->points[pointIndices[i]].y
            << " " << cloud->points[pointIndices[i]].z
            << " (squared distance: " << squaredDistances[i] << ")" << std::endl;
}
    
```

Por otro lado el Octree es una estructura de datos jerárquica empleada tanto para la búsqueda, como para reducción de la cantidad de puntos (downsampling) o la compresión de nubes de puntos. Un Octree es un árbol en el que cada nodo (pixel 3D o voxel) representa un punto en la nube que se considera el centro de cada voxel y que contiene ocho hijos(o ninguno) que son a su vez los vecinos del punto principal. A diferencia del Kd-Tree donde cada nodo representa una división basándose en las dimensiones, este método realiza una subdivisión por puntos y sus vecinos asociados. Esta estructura se emplea además en motores 3D o en la generación de gráficos tridimensionales.

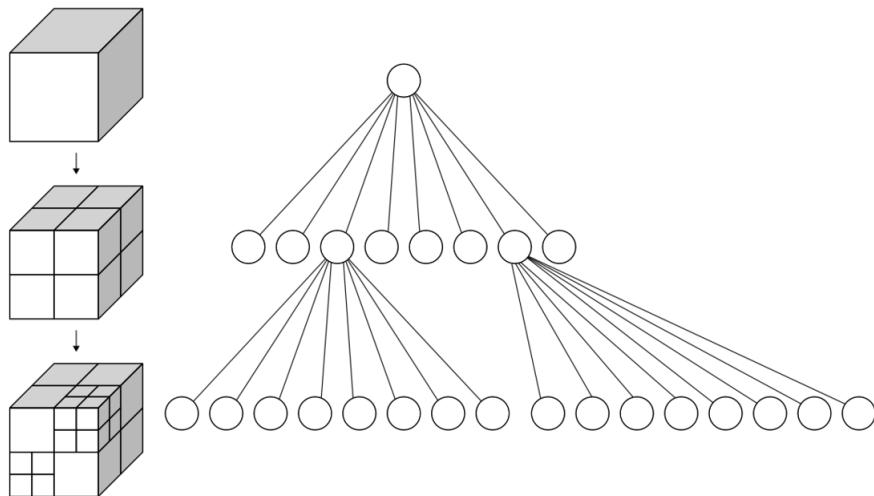


Figura 5.27: Ejemplo de la organización de un Octree

En PCL el Octree se representa por medio diferentes clases según el tipo de función, encontrándose entre las que se destacan las siguientes:

- pcl::octree::OctreePointCloudSearch para la búsqueda por radio, cantidad de vecinos y dentro de un voxel determinado.
- pcl::io::OctreePointCloudCompression para realizar compresión/descompresión de nubes de puntos.
- pcl::octree::OctreePointCloudChangeDetector para comparar dos nubes de puntos en base a sus

Octrees y detectar diferencias, por medio del retorno de índices de puntos que no figuran en una de las nubes.

Estimación de normales

Para diferenciar un punto de otro en una nube de puntos, no basta únicamente con su posición, sino que es necesario computar una característica 3D que sea similar para puntos que se encuentran en superficies similares. Para conseguir ésto, PCL ofrece la computación de normales, donde un vector normal n de un punto, se define como el vector perpendicular al plano tangente que contiene a ese punto. Estos vectores se emplean para diversas tareas entre las que se destacan:

- La generación de gráficos por computadora tridimensionales, en la detección de la orientación de una fuente de luz y mejorar los efectos visuales en una escena.
- Composición digital, donde se renderizan modelos o imágenes 3D por computadora superponiendo varias imágenes. Las capas renderizadas generadas, contienen información de normales pueden ser modificadas para cambiar la textura de un objeto según la fuente de iluminación.

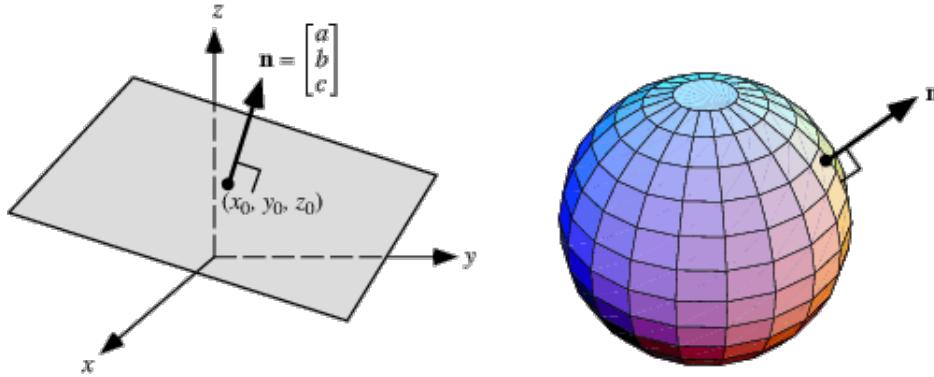


Figura 5.28: Ejemplo de vector normal n , perpendicular a un punto.

Debido a las filas de puntos proporcionan coordenadas de los puntos que componen la superficie de un objeto, la computación de las normales de éstos, se calcula por medio de la generación de una matriz de vectores y valores propios de cada punto P_i (vectores que son invariables a cambios de escala o transformaciones), que es calculada empleando los k vecinos de éste y el centroide de éstos. Los valores de esta matriz se emplean en la técnica de análisis de componentes principales(PCA), que permite obtener las componentes principales con mayor variación, en este caso se obtiene el vector que es más representativo para el punto según sus vecinos más cercanos (vector normal). Una vez realizado este cálculo y teniendo los vectores de cada punto, aún es necesario calcular la orientación de las normales, para éstas utiliza el punto de visión V_p para orientar las normales n_i de todos los puntos, haciendo cumplir siguiente ecuación:

$$\vec{n}_i \cdot (\mathbf{v}_p - \mathbf{p}_i) > 0$$

Figura 5.29: Fórmula de equivalencia normales

La precisión con que se estimen las normales para una superficie en PCL depende en gran medida de la escala que se utilice para el cálculo, que se establece por medio del radio de búsqueda (pcl::Feature::setRadiusSearch) o de la cantidad de vecinos empleados para la computación de la normal (pcl::Feature::setKSearch). Si se emplea un rango razonablemente bajo, se considerarán menos vecinos para cada punto provocando que exista mayor similitud entre normales de la misma superficie y diferencia entre normales de distintas superficies y, en consecuencia, exista un mayor nivel de detalle. Por el contrario, si se emplea una escala muy alta, se considerarán más vecinos para la computación de las normales de puntos, provocando que en las regiones límites entre distintas

superficies se abarque un mayor rango de vecinos de la zona adyacente, provocando que las normales muestren menor diferencia entre superficies diferentes.

En PCL el cálculo de normales se realiza por medio de la clase pcl::NormalEstimation, que acepta un tipo de punto coordenada y un tipo de punto normal, y puede realizarse para toda la nube completa o, para un subconjunto de puntos, por medio de la utilización de índices. Si se desea realizar la estimación para toda la nube, basta con especificar a la clase de estimación de normales la nube de entrada, el método de búsqueda y el radio de búsqueda o la cantidad de vecinos. A continuación, se muestra un ejemplo de código fuente que realiza la computación de normales:

```
#include <pcl/point_types.h>
#include <pcl/features/normal_3d.h>

{
    pcl::PointCloud::Ptr cloud (new pcl::PointCloud());

    // Se lee o se crea una nube de puntos
    ...

    // Se instancia la clase de estimación de normales
    pcl::NormalEstimation<pcl::PointXYZ, pcl::Normal> ne;
    ne.setInputCloud (cloud);

    //Se crea una instancia vacía de kd-tree y se pasa al objeto de estimación de normales.

    pcl::search::KdTree<pcl::PointXYZ>::Ptr tree (new pcl::search::KdTree<pcl::PointXYZ> ());
    ne.setSearchMethod (tree);

    // Variable para normales de salida
    pcl::PointCloud<pcl::Normal>::Ptr cloud_normals (new pcl::PointCloud<pcl::Normal>());

    // Se establece el radio de salida en metros
    ne.setRadiusSearch (0.03);

    // El tamaño de las normales tiene que ser el mismo que el de los puntos en la nube de Entrada
    ne.compute (*cloud_normals);
}

}
```

Si se desea realizar la computación de las normales de algunos puntos, se debe especificar además la estructura de los índices y asignárselo a pcl::NormalEstimation:

```
#include <pcl/point_types.h>
#include <pcl/features/normal_3d.h>

{
    pcl::PointCloud::Ptr cloud (new pcl::PointCloud());

    // Se crea el conjunto de índices para ser empleado (10% del total de puntos)
    std::vector<int> indices (floor (cloud->points.size () / 10));
    for (size_t i = 0; indices.size (); ++i) indices[i] = i;

    pcl::NormalEstimation<pcl::PointXYZ, pcl::Normal> ne;
    ne.setInputCloud (cloud);

    // Se pasan los índices
    boost::shared_ptr<std::vector<int> > indicesptr (new std::vector<int> (indices));
    ne.setIndices (indicesptr);

    pcl::search::KdTree<pcl::PointXYZ>::Ptr tree (new pcl::search::KdTree<pcl::PointXYZ> ());
    ne.setSearchMethod (tree);
}
```

```
// Normales de salida
pcl::PointCloud::Ptr cloud_normals (new pcl::PointCloud());
ne.setRadiusSearch (0.03);
ne.compute (*cloud_normals);
}
```

Filtrado de ruido de la nube

Debido a que una captura puede contener valores espurios, debido a la precisión del sensor, medidas erróneas u falta de puntos en determinadas partes de una nube de puntos, o simplemente es necesario reducir la cantidad de puntos para disminuir el tiempo de computación. Para solucionar ésto, PCL ofrece varios algoritmos de filtrado de nubes de puntos entre los que se encuentran:

- Passthrough Filter
- Conditional Removal
- Outlier Removal

El algoritmo de Passthrough Filter consiste en remover de la nube aquellos elementos que se encuentran fuera de un rango especificado por el usuario, por lo que este método únicamente requiere especificar el eje de filtrado y el rango sobre ese eje (mínimo y máximo). Este método se realiza por la clase `pcl::PassThrough`, que requiere el tipo de punto para el filtrado. A continuación, se muestra el proceso de filtrado para una nube existente:

```
// Se define la nube cloud para el tipo de punto pcl::PointXYZ
...
pcl::PassThrough filter;
filter.setInputCloud(cloud);

// Se filtran los valores en el eje Z que no se encuentren entre 0-2 mts.
filter.setFilterFieldName("z");
filter.setFilterLimits(0.0, 2.0);

filter.filter(*filteredCloud);
```

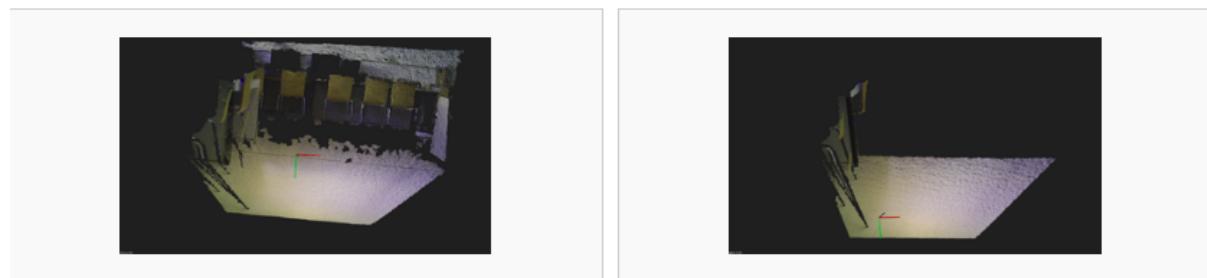


Figura 5.30: Ejemplo de nube original a la izquierda y nube filtrada con passthrough en eje Z.

El algoritmo Conditional Removal consiste en crear una o mas condiciones que verifican los valores de los atributos de un punto (tales como las coordenadas sobre un eje) y mantener solo aquellos puntos que cumplen ésta. Para ello, PCL representa las condiciones por clases siendo las condiciones disponibles AND (`pcl::ConditionAnd`) y OR (`pcl::ConditionOr`), que por medio del método `addComparison()` permiten especificar el tipo de atributo, el operador de comparación ($<$, \leq , $=$, \geq , $>$) y el valor de la condición. Finalmente para realizar el filtrado, se crea una instancia de `pcl::ConditionalRemoval` que recibe las condiciones especificadas y genera la nube de salida. En el siguiente ejemplo se realiza el mismo filtrado que en Passthrough Filter empleando el Conditional Removal:

```

pcl::ConditionAnd<pcl::PointXYZ>::Ptr condition(new pcl::ConditionAnd<pcl::PointXYZ>);

// GT (Greater Than), LT(Less Than)
condition->addComparison(pcl::FieldComparison<pcl::PointXYZ>::ConstPtr(new pcl::FieldComparison
    <pcl::PointXYZ>("z", pcl::ComparisonOps::GT, 0.0)));
condition->addComparison(pcl::FieldComparison<pcl::PointXYZ>::ConstPtr(new pcl::FieldComparison
    <pcl::PointXYZ>("z", pcl::ComparisonOps::LT, 2.0)));

// Se filtran los puntos de la nube cloud previamente inicializada,
// y se guarda el resultado en filteredCloud
pcl::ConditionalRemoval<pcl::PointXYZ> filter;
filter.setCondition(condition);
filter.setInputCloud(cloud);
filter.filter(*filteredCloud);

```

Con respecto al algoritmo Outlier Removal, existen dos variantes: Basado en radio y Estadístico. En el método basado en radio se especifica un radio de búsqueda y la cantidad mínima de vecinos que un punto debe poseer para no ser considerado como outlier. De esta manera el algoritmo iterará todos los puntos en la nube y por cada punto verificará que dentro del radio especificado existan al menos la cantidad mínima requerida de vecinos. Este comportamiento se realiza por medio de la clase pcl::RadiusOutlierRemoval.

Por otro lado, el Statistical Outlier Removal itera cada punto en la nube y calcula la distancia media entre el punto y sus vecinos, la cual es comparada con la distancia de una distribución normal Gaussiana con media μ y desvío estándar σ , eliminando aquellos puntos que caen fuera del rango de la distribución. Este método se implementa por medio de la clase pcl::StatisticalOutlierRemoval que acepta la nube, la media y el desvío estándar de la distribución de probabilidad.

Resampling de la nube: DownSampling y UpSampling

Resampling consiste en modificar la cantidad de puntos en una nube, ya sea aumentando la cantidad de puntos, reconstruyendo la superficie original para que sean suficientes para el análisis (upsampling) o, disminuyéndola sin comprometer significativamente la precisión para que el análisis de la misma sea más eficiente (downsampling). El downsampling en PCL se puede realizar el método de Voxelización o de Uniform Sampling. El método de Voxelización consiste en emplear un conjunto de voxels organizados en una estructura Octree para computar el punto medio del voxel, es decir, aquel punto que es un promedio de las coordenadas de todos los puntos que pertenecen al Voxel Grid. De esta manera, prevalecen solamente aquellos puntos principales que son representativos para cada voxel. PCL implementa este comportamiento por medio de la clase pcl::VoxelGrid, que permite especificar el tamaño de cada voxel (en cm) para cada una de las dimensiones X,Y,Z. En la siguiente porción de código se muestra un ejemplo de voxelización:

```

...
pcl::VoxelGrid<pcl::PointXYZ> filter;
filter.setInputCloud(cloud);

// Se especifica el tamaño del voxel en cada eje
filter.setLeafSize(0.01f, 0.01f, 0.01f);
filter.filter(*filteredCloud);
...

```

El método de Uniform Sampling realiza la misma tarea, sin embargo, retorna los índices de los puntos filtrados en lugar del punto y se emplea principalmente como parte del proceso de generación de descriptores:

```

pcl::UniformSampling<pcl::PointXYZ> filter;
filter.setInputCloud(cloud);
filter.setRadiusSearch(0.01f);
pcl::PointCloud<int> keypointIndices;
filter.compute(keypointIndices);

```

El upsampling en PCL se realiza por medio del método Moving Least Squares(MLS), que es un método empleado para la reconstrucción de una superficie en base a un conjunto de datos de muestra (en este caso puntos). Este método consiste en generar una función continua que representa al conjunto de datos de muestra, empleando los valores de las variables independientes y dependientes para el computo. Para ello, dado un conjunto de muestras $S = \{ (x_i, f_i) / f(x_i) = f_i \}$, con x_i, f_i siendo números reales, se computa por cada punto arbitrario x el valor mínimo cuadrado ponderado (Mean Least Square) con respecto a cada una de las muestras, produciendo un conjunto de polinomios de grado m $p(x_i)$. De todos éstos, se emplea el polinomio que minimice el error mínimo cuadrado para calcular el valor de este punto en la función.

$$\sum_{i \in I} (p(x_i) - f_i)^2 \theta(\|x - x_i\|)$$

Figura 5.31: Fórmula para el cálculo de MLS

De esta forma, MLS obtiene una función final a partir de un conjunto de funciones locales calculadas en base a los datos de muestra, cuyo valor de precisión es controlado por medio de los pesos θ . El método de MLS se implementa en la clase `pcl::MovingLeastSquares`, que requiere la nube de entrada, un Kd-Tree para estructurar la nube, y un radio de upsampling para generar los nuevos puntos, determinando este valor la cantidad de puntos producidos (si es demasiado grande se generarán menos puntos).

2.3.2. Algoritmos de segmentación de objetos

Segmentación

La segmentación consiste en dividir una nube de puntos en uno o varios clusters para que puedan ser procesados independientemente (donde cada cluster representa un objeto de interés para ser procesado), lo que en combinación con otras herramientas, permite obtener modelos pertenecientes a objetos individuales en la captura y aislar superficies con distintas formas. PCL ofrece varios métodos alternativos para realizar la segmentación entre los que se encuentran:

- Euclidean Segmentation
- Region Growing Segmentation
- Min-Cut Segmentation

Euclidean Segmentation, o segmentación Euclidiana, itera cada uno de los puntos de la nube, y por cada uno computa la distancia Euclidiana entre el punto iterado y uno de sus vecinos y, si ésta es menor a un límite (o threshold), significa que ambos pertenecen al mismo cluster, por lo que ambos puntos son marcados como iterados y agregados al mismo cluster. Este proceso continúa por cada uno de los vecinos del punto y luego por los vecinos de éstos, hasta que no existan más puntos que agregar al cluster. Cuando sucede ésto, se crea un nuevo cluster y el proceso se repite con aquellos puntos que no se han agrupado aún en un cluster, hasta iterar todos los puntos en la nube. Este algoritmo se implementa en PCL por medio de la clase `pcl::EuclideanClusterExtraction`, que acepta como parámetros una nube de puntos de entrada, un tamaño máximo y mínimo para los clusters, un árbol de búsqueda (como Kd-Tree) y un valor de tolerancia para controlar la tolerancia (distancia en cm) para considerar a un punto como perteneciente a un cluster o no; Así, si este valor es pequeño provocará que los objetos en la nube se dividan en varios clusters y, si es demasiado grande para el/los objeto/s que se desean segmentar agrupará todos los puntos de éstos en el mismo cluster.

Este algoritmo posee una variación que se denomina Conditional Euclidean Segmentation, o segmentación Euclidiana condicional, que además de realizar la computación y verificación de distancias, permite que el usuario especifique una condición para cada par de puntos a ser comparados (denominándose *seed* o semilla al punto procesado y *candidate* o candidato al vecino de la semilla que está siendo iterada). En esta función el usuario recibe una copia de ambos puntos y la distancia cuadrada de éstos y retorna un valor booleano, que en caso de ser verdadero permite que el candidato pueda ser agregado al cluster y falso en caso contrario. Este algoritmo se encuentra implementado en la clase `pcl::ConditionalEuclideanClustering`,

que recibe los mismos parámetros que la función estándar y permite especificar la función de condición por medio del método setConditionFunction().

Region Growing Segmentation, realiza el agrupamiento en clusters en base a una verificación de la suavidad de la superficie, que se determina procesando el ángulo entre las normales y la diferencia de curvaturas entre puntos. Este algoritmo se implementa por medio de la clase pcl::RegionGrowing, que recibe los mismos parámetros de Euclidean Segmentation y adicionalmente emplea la estimación de normales y un valor límite para la curvatura. Este algoritmo tiene una variación conocida como Region Growing RGB, que en lugar de emplear las normales y la curvatura, utiliza los mismos parámetros que Euclidean Segmentation en combinación con el color de la nube. De esta forma, el proceso de segmentación se realiza con una nube con información de color (con puntos pcl::PointXYZRGB o pcl::PointXYZRGBA) y se puede controlar en base a límites de color para generar clusters, tanto entre puntos como entre clusters.

El algoritmo Min-Cut o corte mínimo, se emplea para segmentar una nube de puntos en dos clusters, donde uno pertenece a un objeto cuyas coordenadas se conocen (foreground) y el otro perteneciente a puntos que no forman parte del objeto y se consideran parte del fondo de la escena donde se encuentra posicionado éste (background). Para realizar ésto, el algoritmo genera un grafo en base a la nube de puntos donde cada punto se representa como un nodo del grafo, y adicionalmente agrega al grafo dos vértices globales más denominados sink y source. Los nodos source y sink se encuentran interconectados por medio de aristas a todos los demás puntos de la nube y además cada nodo que representa un punto, se conecta por medio de aristas a sus puntos vecinos más cercanos. Así, el grafo producido por Min-Cut se genera uniendo los nodos que representan puntos de la nube con sus k-vecinos más cercanos (definiéndose k por el usuario) y, con los nodos globales sink y source con aristas que contienen un peso, que es calculado de manera diferente según los tipos de nodos que éstas unen. La computación de los pesos de las aristas que conectan distintos tipos de nodos se realiza de la siguiente manera:

- Primero, se asignan pesos a las aristas que interconectan los nodos que representan los puntos de la nube de puntos (denominados costo de suavidad), cuyo valor depende de la distancia entre éstos y se computa por medio de la fórmula $smoothCost = e^{((-distanciaEntrePtos/\sigma)*\gamma)}$, donde σ es el espaciado entre los puntos de la nube que depende de la resolución y es establecido por el usuario. De esta forma, mayor será la probabilidad de corte de un borde, cuanto mayor sea la distancia entre puntos en la nube.
- Luego se establecen las penalidades de foreground y background, donde la penalidad de foreground es el peso que se asigna para las aristas que unen cada punto de la nube con el vértice source (valor definido por el usuario), y la penalidad de background son los pesos de las aristas que unen los puntos de la nube con el vértice sink. Ésta última es un valor calculado en base a la distancia horizontal hacia la ubicación del objeto, por medio de la siguiente fórmula: $penalidadBackground = distanciaAlCentro/radio$, donde la distancia al centro del objeto en el plano horizontal es calculada por medio de la ecuación: $distanciaAlCentro = \sqrt{(X - CentroX)^2 + (Y - CentroY)^2}$, donde X , Y son las coordenadas del punto, mientras que el radio es un parámetro especificado por el usuario y define el rango fuera del cual, no existen puntos que pertenezcan al objeto que se está segmentado (o foreground).

Finalmente, luego de configurar el grafo se realiza la búsqueda del mínimo corte recorriendo los nodos del grafo, considerando tanto las penalidades de background/foreground como el valor de suavidad, al momento de realizar un corte mínimo. Así, cuando el corte mínimo se computa, se logra que los puntos vecinos sean asignados al mismo segmento (empleando el valor de suavidad) y que aquellos puntos que se encuentran débilmente conectados al objeto o, que se encuentran dentro del rango definido por el radio de background, sean asignados al background y no al objeto a segmentar.

Finalmente, RANSAC (Random Sample Consensus) es un algoritmo de muestreo aleatorio que para un conjunto de datos de entrada con ruido, estima los parámetros que permiten ajustar éstos a un modelo preestablecido. Este algoritmo considera que en la nube de puntos de entrada existen puntos que pueden ser ajustados a un modelo preestablecido con un margen de error especificado (inliers), y puntos que no se ajustan al modelo de RANSAC(outliers). El funcionamiento de este algoritmo consiste en especificar un tipo de modelo y realizar N iteraciones, donde en cada una:

1. Se toma un subconjunto de puntos mínimos aleatorios de la nube de entrada (considerado suficiente para estimar los parámetros del modelo) y, empleando el tipo de modelo especificado, se entrena un



modelo para este subconjunto de puntos y se computan los parámetros asociados a este.

2. A continuación, el algoritmo verifica cuales puntos de la nube de entrada completa son consistentes con el modelo y sus parámetros estimados previamente, empleando una función de costo o función de pérdida específica del modelo (loss function). Los puntos que no se ajusten al modelo instanciado con un margen de error, se consideran outliers, mientras que el resto de puntos que se ajustan al modelo se consideran inliers hipotéticos y forman parte del conjunto de consenso (consensus set).
3. Se repite de nuevo el paso 1.

De esta forma, el algoritmo RANSAC se repite una serie de veces hasta que se tengan suficientes inliers como para ser considerada confiable a la estimación. Una ventaja de RANSAC es que es sumamente robusto para estimar los parámetros asociados a un modelo, aún cuando se cuenta con ruido en la muestra. Por otro lado, su desventaja radica en que no existe un límite de tiempo para computar estos parámetros, por lo que si se requiere generar un modelo con pocas iteraciones es posible que la solución obtenida no sea satisfactoria.

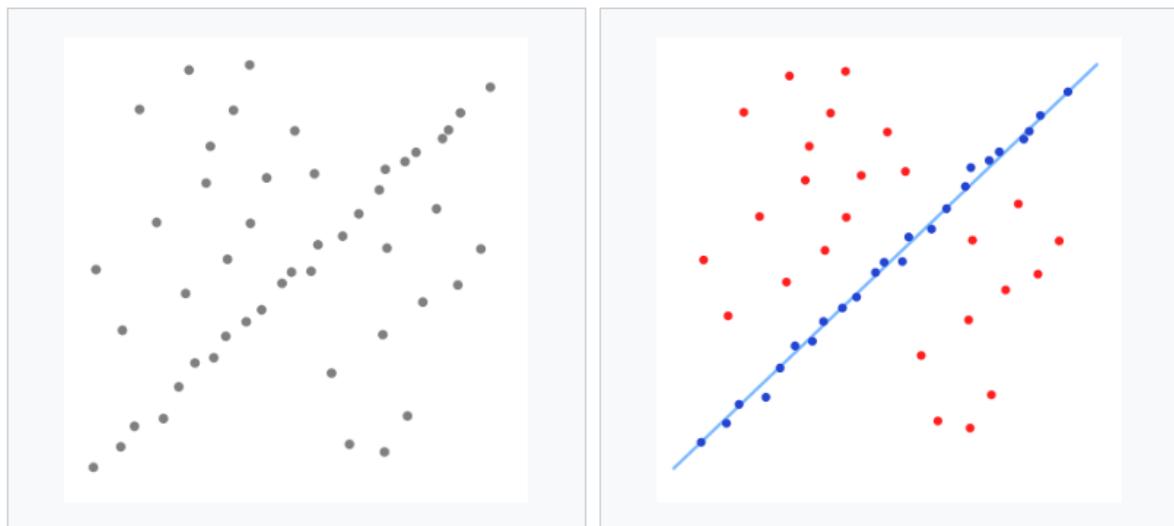


Figura 5.32: Ejemplo de algoritmo RANSAC. En la figura izquierda se puede observar un conjunto de puntos con outliers para ser ajustado con RANSAC empleando el modelo de línea. Mientras que en la derecha, se puede visualizar los puntos azules considerados por el modelo de línea de RANSAC y los outliers que no se ajustan a este modelo representados en rojo.

PCL ofrece varios modelos geométricos predefinidos para emplear con RANSAC, entre los que se encuentran: Circulo 2D, Circulo 3D, Cono, Cilindro, Linea, Esfera, Vara(Stick) y Plano.

2.3.3. Algoritmos para generación de descriptores

Con respecto a la generación de descriptores, PCL ofrece dos tipos de descriptores: Descriptores locales y descriptores globales. Los descriptores locales, se emplean para describir la geometría alrededor de cada punto, sin considerar la geometría total del objeto del que forma parte ese punto, por lo que cuando se computan éstos, se debe hacer un filtrado previo de los puntos clave del objeto o keypoints que se desean procesar. Estos descriptores se emplean para el reconocimiento de objetos y para la registración (registration), que es una técnica donde se alinean dos nubes de puntos y, por medio de transformaciones lineales, se detecta si existen áreas comunes en ambas nubes de puntos.

Por otro lado, PCL ofrece descriptores globales que describen la geometría de un cluster de puntos que representa un objeto, por lo que para emplear estos descriptores se requiere pre-procesar una nube de puntos, con el fin de aislar el objeto. Estos descriptores se aplican para el reconocimiento de objetos y clasificación, estimación de posición y análisis de geometría (tipo de objeto, forma, etc.). Los descriptores locales que emplean un radio de búsqueda, mayormente pueden ser usados como globales, si

se computa un solo punto en el cluster y se modifica el radio de puntos que se consideran vecinos, para abarcar todos los puntos que componen el objeto.

Existen varios tipos de descriptores en PCL, cada uno empleando su propia técnica, ya sea empleando los ángulos de las normales o las distancias Euclidianas entre puntos. Sin embargo, con el fin de reducir el tamaño de cada descriptor, todos se organizan en histogramas cuyos rangos de escala se corresponden con la característica que es parte del descriptor (por ejemplo, distancia entre puntos), asociándose cada una de las características del descriptor a un histograma, donde éstos se encuentran divididos en k subdivisiones y, en cada rango del histograma, se representan las ocurrencias de puntos dentro de ese rango. De esta forma, cada algoritmo para la generación de descriptores realiza su propia subdivisión del histograma, dependiendo del rango de valores que sea más representativo en la variable, por lo que éstas se generan dinámicamente y se producen en mayor medida para los valores donde existen mayor cantidad de puntos con esa característica.

A continuación, se muestran los descriptores tanto locales como globales, que se ofrecen en PCL junto con el tamaño (en bytes) de los histogramas que componen cada uno:

Descriptor	Tipo	Tamaño
PFH (Point Feature Histogram)	Local	125
FPFH (Fast Point Feature Histogram)	Local	33
RSD (Radius-Based Surface Descriptor)	Local	289
3DSC(3D Shape Context)	Local	1980
USC(Unique Shape Context)	Local	1960
SHOT(Signatures of Histograms of Orientations)	Local	352
Spin Image	Local	153
RIFT (Rotation-Invariant Feature Histogram)	Local	32
NARF(Normal Aligned Radial Feature)	Local	36
RoPs(Rotation Projection Statistics)	Local	135
VFH(Viewpoint Feature Histogram)	Global	308
CVFH(Clustered Viewpoint Feature Histogram)	Global	308
OUR-CVFH(Oriented,Unique and Repeatable CVFH)	Global	308
ESF(Ensamble Shape Of Functions)	Global	640
GFPFH(Global Fast Point Feature Histogram)	Global	16
GRSD(Global Radius-Based Surface Descriptor)	Global	21

En el siguiente capítulo, se expondrá en detalle el funcionamiento de los descriptores que fueron seleccionados para ser empleados en el clasificador de tipos de fallas en la presente tesina.

Capítulo 4. Técnicas de reconocimiento y procesamiento de fallas

1. ¿Qué es Machine Learning(ML)?

La Inteligencia Artificial(IA) es una disciplina que abarca todos aquellos mecanismos(árboles de decisión, redes neuronales, etc.) que posibilitan a las computadoras imitar la inteligencia humana con el fin de realizar tareas tales como: la toma de decisiones, resolver problemas y el aprendizaje; Considerándose un comportamiento inteligente, a aquél que involucra percibir o deducir información de un contexto y almacenarla en forma de conocimiento, de manera que pueda ser aplicado en futuras situaciones o contextos. Así, Machine Learning (ML) es una rama dentro de IA en la que se emplean mecanismos que se basan en identificar patrones en un conjunto de datos, para generar conocimiento de estas relaciones que puede ser aplicado en futuras predicciones, mientras que Deep Learning es un subconjunto de algoritmos de ML, donde el aprendizaje se realiza por medio de modelos con sucesivas capas que representan el problema, generados automáticamente por medio de la exposición a datos de entrada, que permiten que una máquina aprenda conceptos complicados a través de su descomposición en conceptos más simples. Estas representaciones en Deep Learning se generan por medio de modelos denominados Redes Neuronales expuestas a enormes cantidades de datos, y cuyo funcionamiento se inspira en el funcionamiento del cerebro humano.

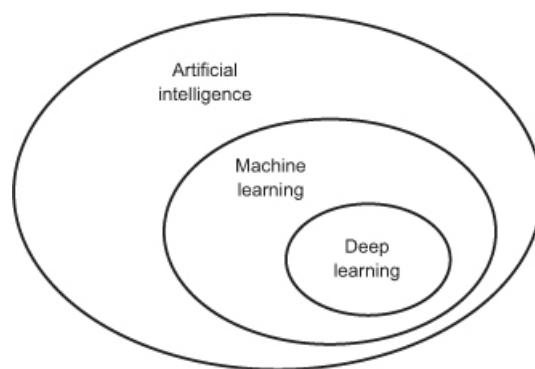


Figura 6.1: Comparación de AI, ML y Deep Learning

De esta forma, ML se diferencia del paradigma de programación clásico donde un equipo de desarrollo programa instrucciones que procesan datos y se generan salidas en base a estructuras predefinidas, en que para esta metodología solamente se ingresan datos (y opcionalmente sus respuestas), y se obtienen reglas de salida que pueden ser aplicadas a nuevos datos para realizar predicciones. Por lo tanto en ML, se considera que dado un programa cuyo rendimiento en la predicción sobre un conjunto de datos (o dataset) se encuentra medido a través de alguna métrica (que indica que tan precisas son sus predicciones sobre ese conjunto de datos), éste aprende de la experiencia si dicho rendimiento mejora al adquirir más experiencia.

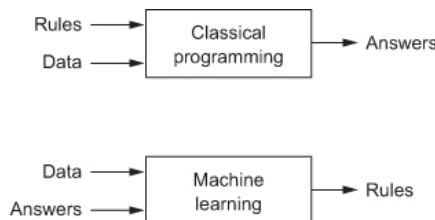


Figura 6.2: Paradigma de ML vs paradigma de programación tradicional.

1.1. Flujo de trabajo en ML

Así, el flujo de trabajo en ML para la generación y prueba de un modelo de predicción se puede subdividir en las siguientes fases:

1. Pre-procesamiento de datos(feature extraction).
2. Etapa de entrenamiento del modelo (training).
3. Evaluación y optimización del modelo.
4. Etapa de validación (testing).

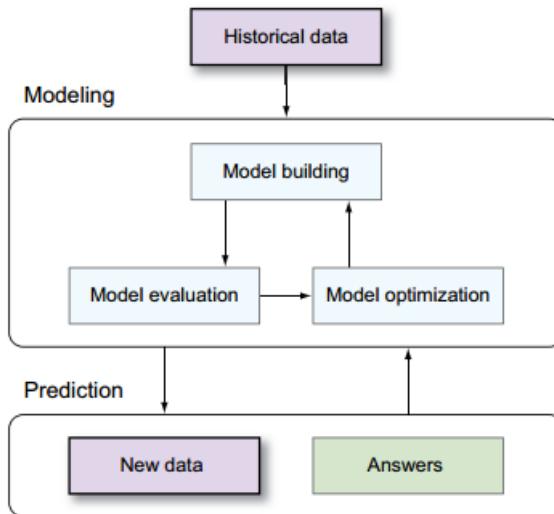


Figura 6.3: Flujo de trabajo general en ML

Debido a que los datos en el mundo real frecuentemente no son aceptables para ser procesados por un algoritmo de ML, dado que contienen valores incorrectos, erróneos o nombres escritos de manera distinta aunque se refieren a la misma entidad, y dado que los algoritmos de ML tienen como finalidad descubrir asociaciones y relaciones en un conjunto de datos de entrenamiento histórico (training dataset) para generar un modelo de predicción, el primer paso para lograr ésto, consiste en realizar el pre-procesado de los datos de manera que se puedan producir datos de alta calidad, ruido leve y correlaciones

fácilmente deducibles que permitan generar un modelo predictivo de alta fidelidad. De esta manera, el pre-procesamiento involucra aplicar técnicas y algoritmos para el saneamiento, visualización y transformación de datos a otro rango de valores, de forma que se reduzca la redundancia de features, la variabilidad de valores y el tiempo de procesamiento, conservando únicamente aquellas features con información relevante para el modelo. Durante esta fase se descarta información, por lo que ésta se debe realizar con cautela ya que si atributos relevantes al modelo se descartan, puede verse afectada la capacidad de predicción del mismo. En general, el pre-procesamiento de datos  en considerar la presencia de las siguientes características en el dataset y aplicar los pasos mencionados:

- Features categóricas: Las features categóricas son aquellos valores no numéricos a los que se les puede asignar un valor numérico, con el fin de que sean de utilidad para los algoritmos, tales como los días de la semana o el género. En general, los algoritmos de machine learning necesitan datos numéricos (salvo algunos casos concretos derivados de los árboles de decisión), por lo que es necesario codificar las features categóricas a través de la creación de clases con valores binarios que representan cada categoría, y luego asignar a cada muestra del dataset un valor (0 o 1) indicando si ésta pertenece o no a una determinada categoría. A continuación, se muestra un ejemplo donde para las categorías de hombre o mujer, se crea una clase binaria y se  valor 1 a la categoría donde se ubica la muestra:

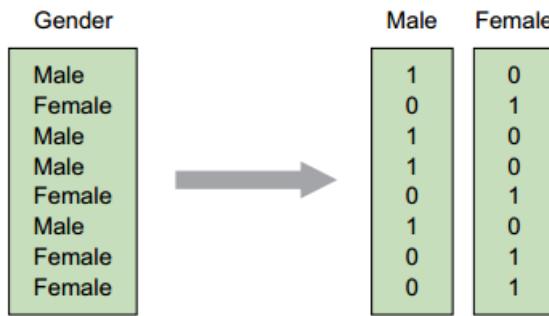


Figura 6.4: Ejemplo de conversión de feature categórica

- Datos faltantes: En general la información faltante en datasets, ya sea porque no pudo ser recolectada o porque no pudo ser medida, puede ocurrir que estos datos sean significativos para el algoritmo de ML, en cuyo caso se debe asignar un valor no válido entre -1 y -999 y proceder a probar el modelo. Mientras que en caso contrario, se puede proceder a eliminar aquellas muestras de datos en las que el valor del atributo no se encuentre. Si la cantidad de muestras descartadas son suficientes como para disminuir la capacidad de predicción del modelo, se puede simplemente reemplazar aquellos valores faltantes por la media o la mediana del resto de valores de ese feature.
- Datos en distintas escalas de valores (Normalización de datos): Algunos algoritmos de ML requieren que las features se normalicen, de manera que residan en el mismo rango de numérico, debido a que el rango de una feature puede influenciar la importancia de la feature con respecto a otras. La normalización consiste en ajustar los valores para que se distribuyan entre un valor mínimo y máximo, generalmente ubicado entre [-1,1] o [0,1]. Existen varias maneras de realizar ésto, una de ellas es rescaling  que la más sencilla consiste en restar al valor mínimo a cada valor del rango de valores y dividir ésto sobre el rango total de valores, lo que brinda valores en el intervalo [0,1] o [-1,1] aplicando la siguiente fórmula:

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

Figura 6.5: Fórmula Rescaling

Alternativamente, se puede aplicar la fórmula anterior reemplazando la resta del valor mínimo por la media de los valores:

$$x' = \frac{x - \text{mean}(x)}{\max(x) - \min(x)}$$

Figura 6.6: Fórmula normalización por media

Otro método para la normalización de features, es la estandarización que consiste en calcular la media y su desvío estándar para los valores de una feature determinada, y luego por cada valor de esa feature substrair la media y dividir por el desvío estándar

$$x' = \frac{x - \bar{x}}{\sigma}$$

Figura 6.7: Fórmula de estandarización

- Verificación de representatividad de los datos (Visualización de datos): Antes de realizar el entrenamiento puede ser necesario realizar la verificación de la relación y validez en las features que componen los datos de entrenamiento (por ejemplo revisar como se relacionan las muestras y los resultados, si se dispone de ambos), necesitándose para ésto representaciones gráficas que indiquen que tan significativos son las muestras de que disponen y los tipos de muestras que podrían estar faltando.

Una de las herramientas empleadas para ésto son los gráficos de mosaicos, donde se representan las proporciones de instancias y los porcentajes de cada clase respecto del total, entre dos features del dataset. Este diagrama consiste en seleccionar dos features y realizar una subdivisión vertical entre las dos clases generando una columna para cada clase, donde el ancho de cada columna es equivalente a la proporción de los datos de esa clase respecto del total de datos. Luego se realiza la división de estos rectángulos por una línea horizontal, donde la altura de cada rectángulo depende de la cantidad de muestras que pertenecen a esa clase. Así, si la línea horizontal que separa ambos rectángulos se encuentra separada de manera considerable, ambas features se encontrarán fuertemente relacionadas, mientras que si por el contrario, se encuentran juntas significará que ambas features no se encuentran relacionadas. A continuación, se muestra un ejemplo para un dataset con información de pasajeros del Titanic, donde se demuestra que el género y la supervivencia se encuentran relacionadas:

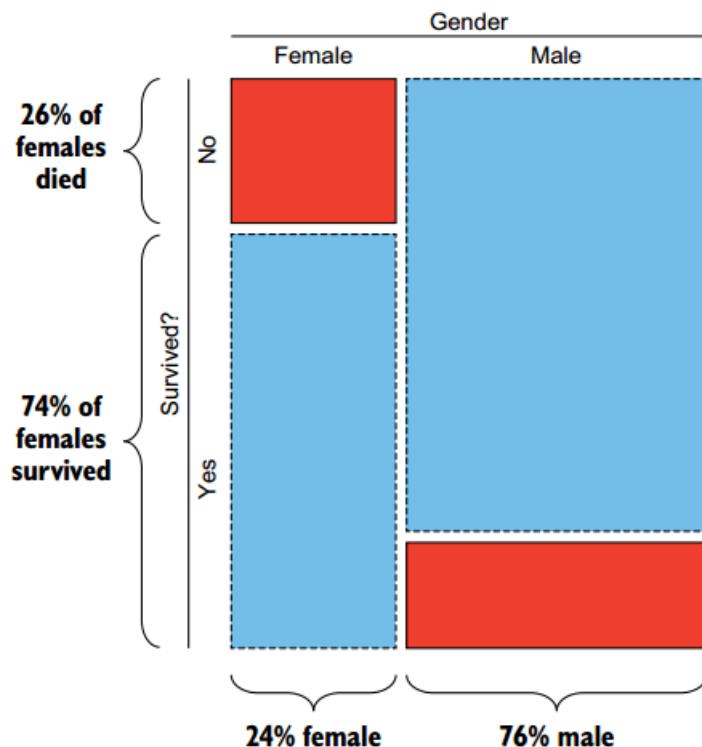


Figura 6.8: Ejemplo de gráfico de mosaicos del dataset del Titanic

Otra herramienta utilizada para este fin son los gráficos de densidad, que permiten mostrar la distribución de alguna de las features, creando para ésto un estimado de la distribución de probabilidad basándose en los valores de esa feature, considerando que los valores proporcionados son una muestra aleatoria que representa la población de valores. Para esto, se utilizan los datos observados en combinación con una técnica estadística conocida como kernel de suavizado (kernel smoothing), la cual dado un conjunto p de valores reales, produce un valor real de salida que es un promedio ponderado de los datos vecinos observados. Así, este diagrama permite visualizar la forma de la distribución en el intervalo completo con mayor precisión a como lo haría un histograma, ya que en éste la forma de la distribución depende de la cantidad de intervalos (barras) que se utilicen. A continuación, la distribución se grafica como una curva que muestra los valores que la variable probablemente puede adoptar, creando un gráfico de densidad por cada categoría que una feature puede adoptar, se pueden visualizar diferencias en el rango de los valores en cada categoría.

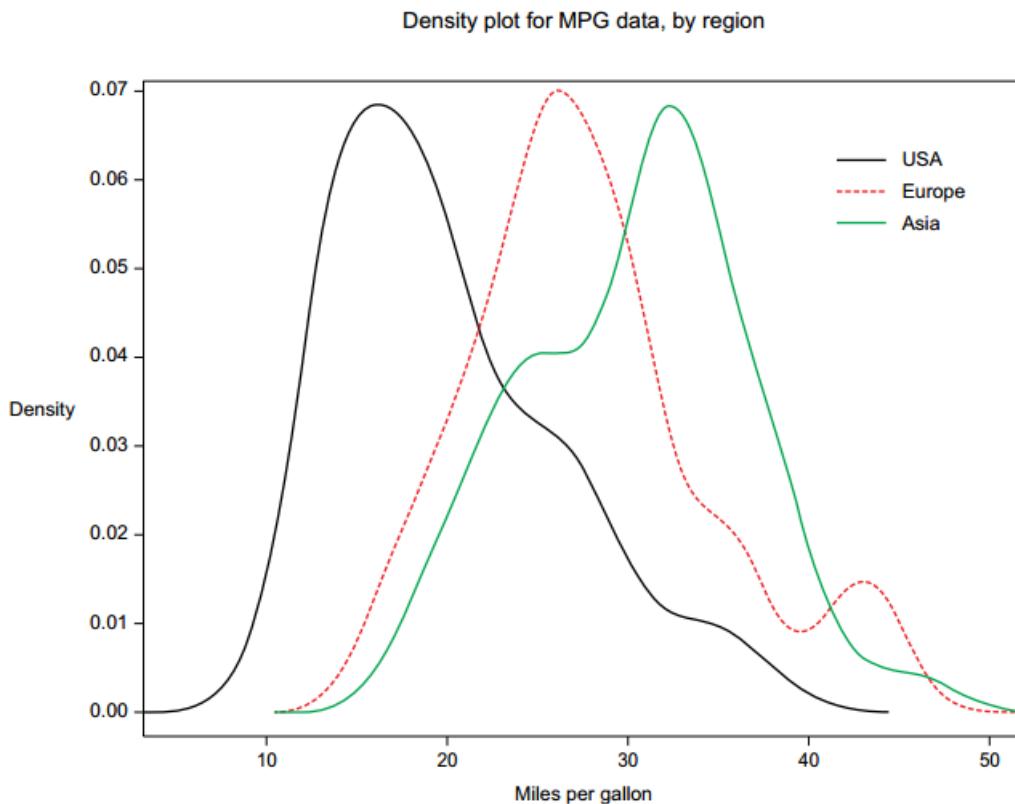


Figura 6.9: Ejemplo de diagrama de densidad para las millas por galón (MPG) que consumen autos fabricados por diferentes países, siendo las clases o categorías las siguientes: USA, Europa o Asia. Este gráfico ilustra la densidad de MPG vs el país del fabricante.

Alternativamente, se pueden emplear diagramas de dispersión (scatter plots), donde se grafican los valores de los dos features, agregando un punto por cada instancia, lo que permite revelar tanto relaciones lineares como no lineares entre features y determinar si existe una relación útil entre ambas features son para el entrenamiento modelo.

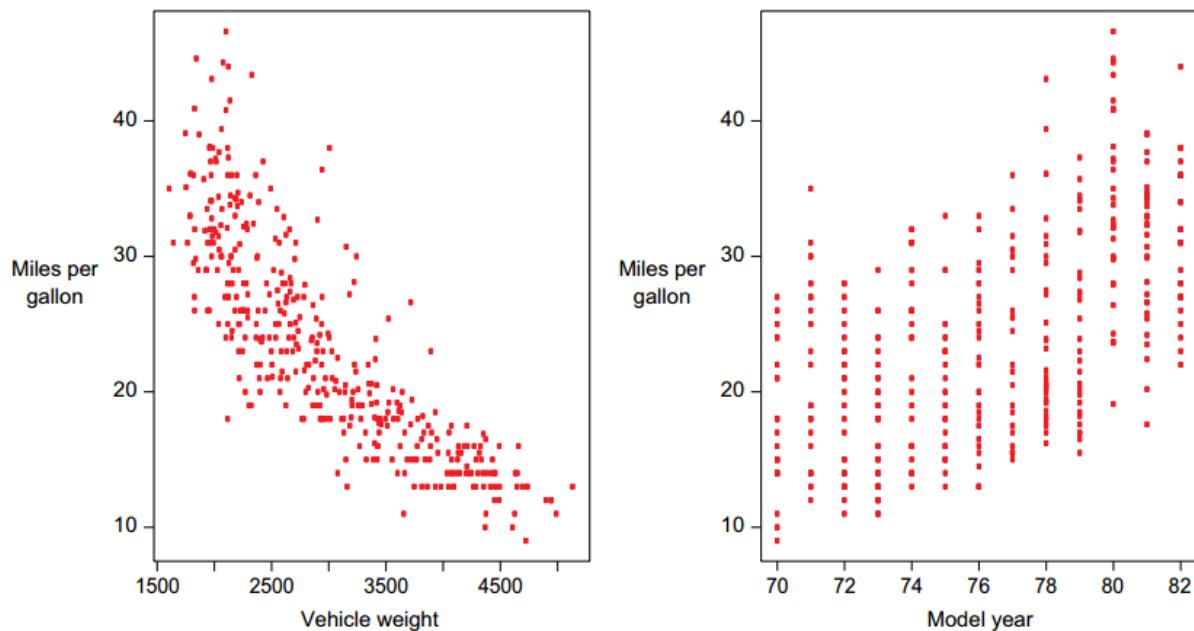


Figura 6.10: Ejemplo de diagrama de dispersión. En la izquierda se muestra que la relación entre las features de MPG y el peso del vehículo no siguen una relación lineal, mientras que en la imagen a la derecha se muestra que MPG y año de fabricación siguen una relación lineal. De estas figuras, se deduce que ambas features se encuentran relacionadas a MPG y sirven para la predicción de MPG.

Otro mecanismo empleado para visualizar la relación entre features de un dataset es la matriz de correlación (o matriz de covarianza) que es una matriz simétrica que consiste en: Dadas n features del dataset, generar una matriz de $n \times n$ que relaciona cada feature con el resto y donde el elemento (i,j) de la matriz representa la correlación entre ambas features, siendo ésta la relación lineal entre ambas variables. Así, si la variabilidad de una feature se encuentra asociada a la variabilidad de la otra, el elemento (i,j) de la matriz contendrá un valor positivo y cuanto más sea esta relación entre ambas features, tenderán a estar negativamente correlacionadas, siendo estos valores inferiores y negativos.

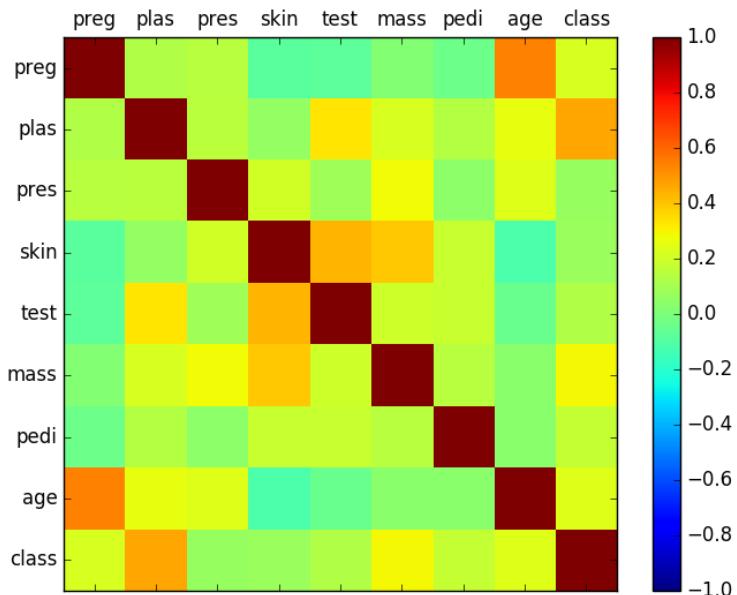


Figura 6.11: Ejemplo de gráfico de matriz de correlación para features relacionadas con personas que padecen diabetes.

Adicionalmente, durante la etapa de pre-procesamiento se puede aplicar un análisis de componentes principales (Principal Component Analysis, PCA) que es un algoritmo que consiste en realizar la transformación de un conjunto de datos, con o sin correlación, a un espacio de menor dimensión sin correlación, denominadas componentes principales. Para ello, aplica a la matriz de correlación la descomposición de valores singulares, que es una factorización donde se computan los eigenvalores (o autovalores) y en base a los se computa su raíz cuadrada, dando como resultado los valores singulares de la matriz. De esta forma, se busca que al reducir la dimensionalidad de los datos, se conserven solamente aquellos elementos que tengan mayor varianza y por lo tanto, sean los que aporten mayor información al modelo a construir.

Durante la fase de pre-procesamiento, si es que los datos del dataset no se estructuraron previamente, también se los estructura, separando los datos de entrada que contienen features en una matriz, donde cada columna se asocia con una feature y cada fila se asocia a una muestra individual de esa feature y, si se dispone de las respuestas a éstos (también denominadas labels o targets), se organizan en un vector que alberga el grupo o clase asociado a cada muestra, siendo éste un vector con 1 columna y con tantas filas como muestras existan en el dataset.

Features										Target
Cust. ID	State	Acct length	Area code	Int'l plan	Voicemail plan	Total messages	Total mins.	Total calls	Churned?	
502	FL	124	561	No	Yes	28	251.4	104	False	
1007	OR	48	503	No	No	0	190.4	92	False	
1789	WI	63	608	No	Yes	34	152.2	119	False	
2568	KY	58	606	No	No	0	247.2	116	True	

Figura 6.12: Ejemplo de separación de features donde se cuenta con información de clientes de una compañía de telefonía, y como target se especifica el feature “churn” que indica que si este cliente se ha dado de baja del servicio pago ofrecido por la compañía o, si ha cambiado de servicio.

Una vez pre-procesados los datos, se lleva a cabo la etapa de entrenamiento donde se exponen los algoritmos de ML a los datos pre-procesados y se ajustan las configuraciones del modelo para controlar el comportamiento de éste ante los datos (hyperparámetros del modelo). Estos hyperparámetros no son ajustados por el modelo empleando los datos de entrada sino que deben ser especificados por el desarrollador, ya que en algunos casos los valores de estos pueden ser difíciles de determinar de manera automática. **Por otro lado, debido a que el interés de generar un modelo radica en observar la capacidad de predicción sobre datos no solamente de entrenamiento sino también aquellos que no ha recibido anteriormente, ya que esto determinará su rendimiento en un entorno real, durante esta fase el dataset completo se suele subdividir en datos de entrenamiento (entre 70 % y 80 % del total de muestras) y datos de testing (30 %-40 % del total de las muestras).**

Existen distintos tipos de métodos de entrenamiento según el objetivo perseguido con la generación del modelo, entre los que se distinguen tres clases principales: Aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por refuerzo. El aprendizaje supervisado, consiste en emplear los datos de entrada y los labels (o clases) asociados a éstos para detectar relaciones entre los datos y sus resultados y predecir nuevos valores en base a éstos. Los modelos producidos por este tipo de método se subdividen en modelos de clasificación y de predicción, siendo los modelos de clasificación aquellos donde se asigna un dato de entrada a una clase predefinida, tales como los detectores de Spam que clasifican cada e-mail en la categoría (Spam o No Spam), o los reconocedores de dígitos manuscritos que asignan a un nuevo valor a una clase entre 0 y 9. Mientras que, los modelos de regresión dado un dato de entrada generan un valor numérico continuo, como por ejemplo el valor del dólar en un modelo financiero.

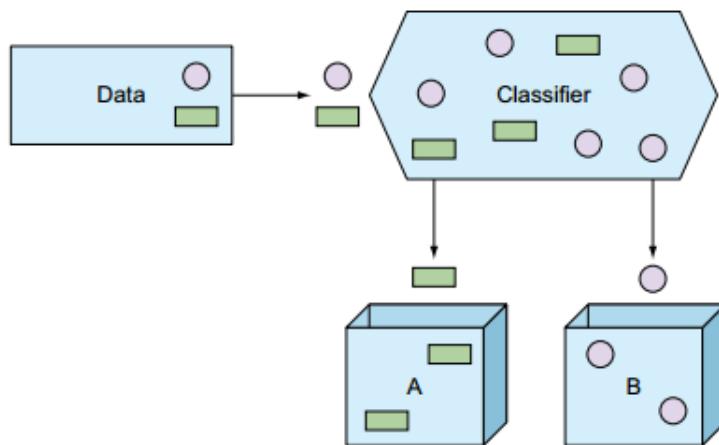


Figura 6.13: Representación gráfica de modelo clasificador

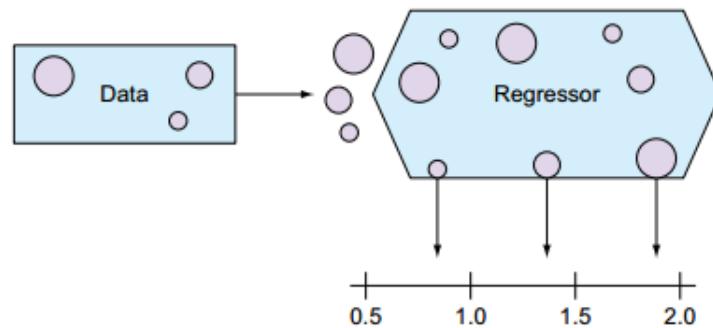


Figura 6.14: Representación gráfica de modelo regresor

Por el contrario, en el aprendizaje no supervisado no se conocen las clases, contando solo con los datos de entrada, por lo que su objetivo es obtener las clases descubriendo grupos de ejemplos similares en los datos (también denominado clustering) o, proyectar los datos desde un espacio de dimensiones

superiores a uno de menores dimensiones, con el objetivo de maximizar la varianza entre las features (reducción de dimensionalidad como PCA). En el aprendizaje por refuerzo, el algoritmo no cuenta con muestras que correspondan con una salida correcta, sino que debe descubrirlas por medio de un proceso de prueba y error, transicionando una secuencia de estados que resultan de la interacción con el entorno.

La fase de evaluación y optimización del modelo se lleva a cabo paralelamente a la fase de entrenamiento y consiste en computar métricas con el dataset de training, para evaluar el desempeño del modelo. Según el tipo de entrenamiento (supervisado o no supervisado), se computan diferentes métricas:

- Clasificación: Accuracy, Precision, Recall, F1-Score, Matriz de confusión.



- Regresión: R2, Variación explicada, Error Medio Cuadrado (Mean Squared Error).
- Clustering: Información mutua(MI), score de homogeneidad, score de completitud.

Con respecto a las métricas de modelos de clasificación, el accuracy es la proporción de las muestras para las que el modelo predice el resultado correcto, mientras que la tasa de error es la proporción para las que el modelo clasifica incorrectamente. Si y_i es el valor del i -ésima muestra, Y_i es el valor verdadero de la muestra, y 1 ($y_i = Y_i$) simboliza la pertenencia de y_i en Y_i , entonces la fracción de predicciones correctas $n_{samples}$ se define como:



$$\text{accuracy}(y, \hat{y}) = \frac{1}{n_{samples}} \sum_{i=0}^{n_{samples}-1} 1(\hat{y}_i = y_i)$$

Figura 6.15: Fórmula para el cálculo del Accuracy

Este valor se indica en el rango de 0-1, por lo que cuanto más próximo es este valor a 1.0 mejor es la capacidad de predicción. Mientras que *precision* indica la capacidad del modelo para clasificar muestras que son **positivas** como tales y no clasificarlas como negativas, considerando el total de valores positivos y negativos; **recall** se calcula por medio de la siguiente fórmula, donde tp son las muestras verdaderas positivas (muestras realmente positivas) y fp son las muestras falsas positivas (aquellas muestras que se clasificaron como positivas pero en realidad son negativas):

$$precision = tp / (tp + fp) \quad (6.1)$$

Recall es la capacidad del modelo de encontrar cuantas muestras positivas reales existen del total de muestras positivas clasificadas (verdaderas positivas y falsas positivas). Este valor se calcula por medio de la siguiente fórmula, donde fn son los falsos negativos (aquellas que son positivas pero son clasificadas como negativas):

$$recall = tp / (tp + fn) \quad (6.2)$$

F1-Score (o F-measure) es un promedio ponderado de *precision* y *recall* que se calcula por medio de la siguiente ecuación:

$$F1 = 2 * (precision * recall) / (precision + recall) \quad (6.3)$$

Las métricas de precision , recall y F1-score se encuentran en el rango 0-1, siendo mejor el desempeño de predicción del clasificador, cuanto más próximo a 1 son estos valores.

Por otro lado, la matriz de confusión es una tabla que permite visualizar y evaluar el accuracy de clasificación, donde cada columna de la matriz representa la cantidad de muestras que fueron predecidas como pertenecientes a una clase y cada fila de la matriz, representa la cantidad de instancias que pertenecen realmente a una clase. Por lo tanto, un elemento (i,j) de la matriz se interpreta como el número de observaciones en el grupo i que fueron clasificados dentro del grupo j , por lo que los elementos que se encuentran en la diagonal de la matriz son la cantidad de muestras para las que el label verdadero fue predecido correctamente, mientras que los elementos que se encuentran fuera de ésta son las muestras clasificadas erróneamente. A continuación, se muestra un ejemplo de la matriz de confusión para un dataset de tipos de planta Iris (Setosa, Versicolor y Virginica):

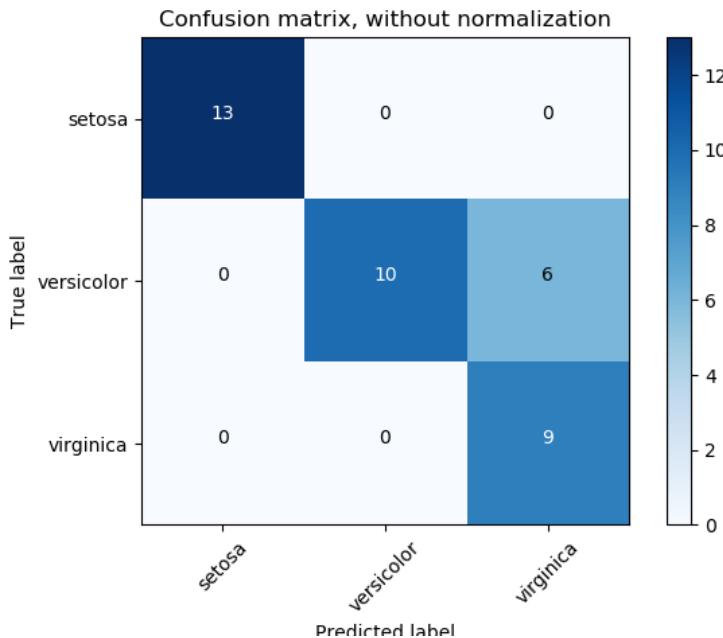


Figura 6.16: Ejemplo de matriz de confusión.

Con respecto a la regresión, la métrica R^2 o r^2 también conocido como coeficiente de determinación, es la proporción de la varianza que existe entre los labels predecidos (variable dependiente) utilizando las muestras de entrada (variable independiente) y el verdadero valor de las mismas, brindando una medida de que tan precisamente las salidas son replicadas por el modelo, basadas en la proporción de la variación total de las salidas, lo que permite establecer que tan eficazmente las muestras futuras serán predecidas. El valor de esta métrica puede ser tanto positivo como negativo, por lo que si este es negativo implica que la capacidad del modelo de predicción es peor que la media de éstos, mientras que si es cero indica que no existe una relación entre los datos de entrada y los labels, por lo que el modelo predice siempre el label independientemente de los datos de entrada; Y finalmente, si este es igual a uno implica que el modelo es capaz de predecir exactamente toda la variabilidad en la variable dependiente (labels). El cálculo de R^2 se realiza por medio de la siguiente fórmula, siendo \hat{y}_i el valor predecido de la muestra, y_i el valor real de la muestra para n muestras:

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=0}^{n_{samples}-1} (y_i - \hat{y}_i)^2}{\sum_{i=0}^{n_{samples}-1} (y_i - \bar{y})^2}$$

 Figura 6.17: Fórmula de R^2

donde:

$$\bar{y} = \frac{1}{n_{samples}} \sum_{i=0}^{n_{samples}-1} y_i$$

 Figura 6.18: Fórmula de cálculo de \bar{y}

La variación explicada mide la proporción en la que un modelo de regresión representa la dispersión (variación) de un conjunto de datos, es decir, entre las predicciones y los valores reales asociados a las muestras. Este score se calcula por medio de la siguiente fórmula, donde y es el label asociado a una muestra, \hat{y} es la salida predecida para ésta y Var es la varianza entre ambas variables:

$$\text{explained_variance}(y, \hat{y}) = 1 - \frac{\text{Var}\{y - \hat{y}\}}{\text{Var}\{y\}}$$

Figura 6.19: Fórmula para el cálculo de la variación explicada

Cuanto más próximo a 1 es este valor, mejor es la capacidad de predicción del modelo.

El error medio cuadrado es una métrica de riesgo que representa el valor esperado del error (o pérdida) cuadrado, que consiste en calcular el promedio de los cuadrados de los errores, es decir, la diferencia entre los labels reales para un conjunto de variables y los valores predecidos para el mismo conjunto. Así, este método aplica la siguiente fórmula matemática donde \hat{Y}_i es el valor predecido, Y_i es el vector de valores observados y n es la cantidad de muestras de entrada para las que se realizaron las predicciones:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (Y_i - \hat{Y}_i)^2.$$

Figura 6.20: Fórmula de Mean Squared Error

Si al resultado obtenido de aplicar esta fórmula, se le aplica la raíz cuadrada se obtiene la métrica de Raíz Cuadrada del Error cuadrado medio (Root-Mean Square Error, RMSE) y representa el desvío estándar de las diferencias de las muestras entre los valores estimados y los valores predecidos. Tanto MSE como RMSE, cuanto más próximos sean a cero los valores de éstas métricas, menos error de predicción existirá.

Por otro lado, con respecto a clustering la métrica de información mutua (Mutual Information, MI) es una medida empleada para comparar la similitud entre dos clases (o labels) para el mismo conjunto de datos. Así, para utilizar esta métrica en un modelo de clustering, se requiere disponer de las clases verdaderas a la que pertenezcan los datos empleados en el entrenamiento del modelo, sin embargo este valor es invariable a los valores absolutos de los labels y a las permutaciones entre ellos. Cuanto más cercano a cero sea este valor, indicará que las asignaciones de clases son independientes y no concuerdan, mientras que cuanto más cercano a uno se observará una mejor concordancia entre asignaciones. Este valor se computa por medio de la siguiente fórmula, donde $|U_i|$ es el número de muestras en el cluster U y $|V_j|$ es el número de muestras en el cluster V :

$$MI(U, V) = \sum_{i=1}^{|U|} \sum_{j=1}^{|V|} \frac{|U_i \cap V_j|}{N} \log \frac{N|U_i \cap V_j|}{|U_i||V_j|}$$

Figura 6.21: Fórmula para el cálculo de información mutua entre clusters U y V.

El score de homogeneidad requiere al igual que la métrica anterior, el conocimiento de las clases reales de las muestras por adelantado y cuanto más próximo a uno sea, significará que ese cluster contiene únicamente puntos de datos que son miembros de la misma clase. Mientras que el score de completitud, permite establecer si todos los miembros de una clase son asignados al mismo cluster. Estas métricas son independiente a las permutaciones en los clusters, y se calculan por medio de las siguientes fórmulas, donde $H(C|K)$ es la entropía condicional de las clases dadas las asignaciones de los clusters, $H(C)$ es la entropía de las clases, n_C y n_K son las muestras que pertenecen a la clase C y al cluster K y $N_{c,k}$ es el número de muestras de una clase c asignada al cluster k :

$$h = 1 - \frac{H(C|K)}{H(C)}$$

Figura 6.22: Ejemplo de fórmula de homogeneidad

$$c = 1 - \frac{H(K|C)}{H(K)}$$

Figura 6.23: Ejemplo de fórmula de completitud

$$H(C|K) = - \sum_{c=1}^{|C|} \sum_{k=1}^{|K|} \frac{n_{c,k}}{n} \cdot \log \left(\frac{n_{c,k}}{n_k} \right)$$

Figura 6.24: Ejemplo de fórmula de entropía condicional dadas las asignaciones de las clases

$$H(C) = - \sum_{c=1}^{|C|} \frac{n_c}{n} \cdot \log \left(\frac{n_c}{n} \right)$$

Figura 6.25: Ejemplo de fórmula de entropía de las clases

Finalmente, durante la fase de validación se procede a analizar y mejorar el nivel de generalización del modelo, es decir, con que precisión éste aplica los conceptos aprendidos de los datos de entrenamiento a nuevos datos dentro del dominio del problema. Dos conceptos relacionados a la pérdida de capacidad de generalización en el entrenamiento supervisado son overfitting y underfitting donde:

- Overfitting ocurre cuando el modelo aprende la distribución de los datos y el ruido del dataset y los considera como conceptos, de manera que se ve afectada negativamente la capacidad de predicción, ya que estos conceptos no aplican a nuevos datos. De esta forma, cuando ocurre overfitting el error de predicción disminuye considerablemente con datos de entrenamiento, sin embargo, al contrastarlo con datos de prueba éste aumenta considerablemente.
- Underfitting sucede cuando el modelo no puede aprender conceptos del dataset de training y, por lo tanto, tampoco puede realizar predicciones sobre datos de testing, mostrando un rendimiento pobre incluso en datos de entrenamiento.

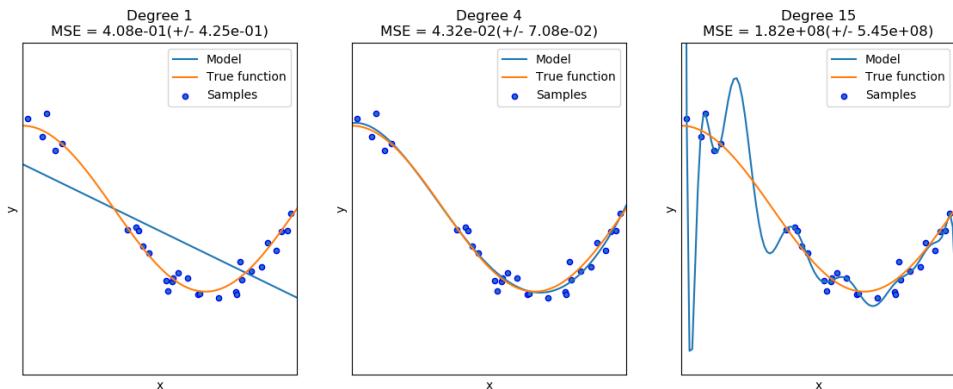


Figura 6.26: En este ejemplo, se muestra la función de tres modelos polinómicos de diferente grado que intentan aproximar parte de la función coseno, representándose éstos por una línea azul, la función real por una línea amarilla y las muestras producidas por puntos azules. En el diagrama de la izquierda, se observa que el modelo (polinomio grado 1) sufre de underfitting, ya que no puede ajustarse a los datos de entrenamiento. En el diagrama de la derecha, se puede observar que el modelo (polinomio de grado 15) sufre de overfitting, ya que aprende cada uno de los datos de prueba incluyendo el ruido y perdiendo la similitud con la función coseno real. En la gráfica del centro, se puede observar que el modelo se ajusta de manera casi perfecta al de la función coseno real, y se ajusta a aquellos datos que la representan, ignorando aquellas muestras que generan ruido.

De esta manera, en esta fase se emplean distintos métodos para evaluar la capacidad de

generalización del modelo entrenado con respecto a los datos de prueba, entre los que se destacan los siguientes:

- Cross-validation: Esta técnica se emplea para evitar problemas como el overfitting y en situaciones donde no se cuenta con suficientes muestras para particionar el dataset en training y testing, perdiendo información relevante para el modelo o para el testing de éste. Este método consiste en realizar particiones en un conjunto de muestras en subconjuntos complementarios de entrenamiento y prueba y, efectuar el entrenamiento sobre la partición de training y realizar la evaluación del rendimiento del modelo sobre la partición de testing. Así, esta técnica busca garantizar que los resultados del predictor son independientes de las particiones de training y testing. Adicionalmente, este método puede ejecutarse repetidas veces, generando diferentes particiones con distintos resultados y luego combinarse éstos (por ejemplo, a través del promediado) con el fin de reducir la variabilidad. Dentro de las aproximaciones para realizar cross validation se distinguen las siguientes:
 - Método Holdout: Este método es el más trivial y consiste en simplemente particionar de manera aleatoria el conjunto de muestras total en subconjuntos complementarios de entrenamiento y prueba, considerando entre el 20 % y 40 % para testing y el resto para training, y posteriormente realizar el entrenamiento y validación con éstos.

Este método tiene la desventaja de que puede existir alta variación al ejecutarse sobre distintos conjuntos de prueba, ya que el resultado depende en gran parte de como se realiza la partición de los datos, sin embargo, tiene la ventaja de ser rápido de computar.

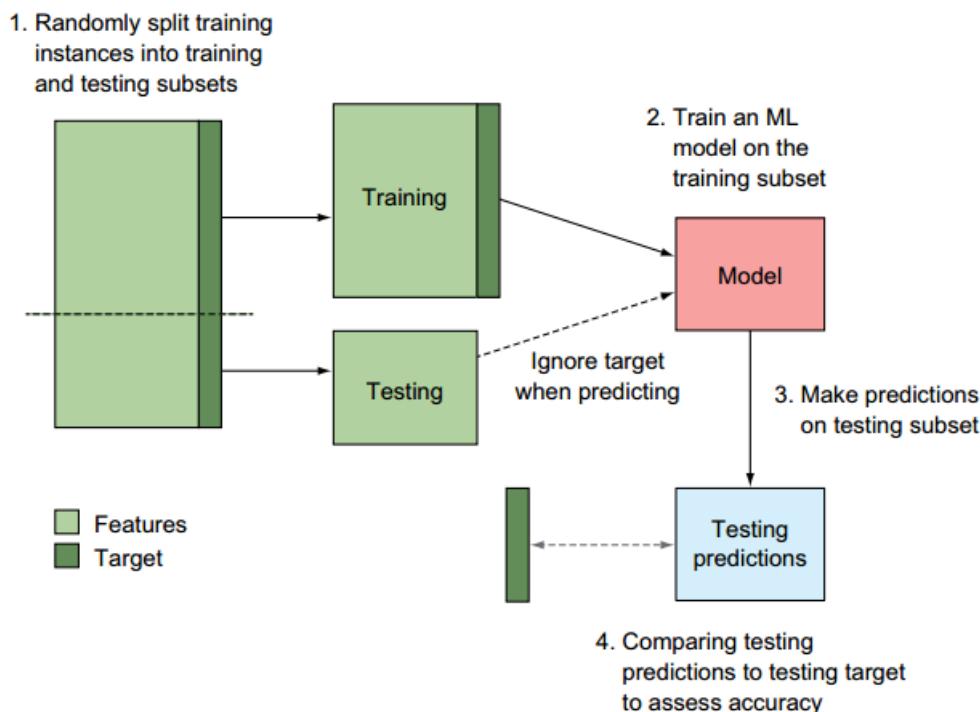


Figura 6.27: Representación gráfica de la retención

- K-Fold cross-validation: Cross validation de K iteraciones consiste en realizar k divisiones complementarias de los datos de prueba (denominados *folds*), empleando uno de los subconjuntos para validación y los restantes $k-1$ como datos de entrenamiento y ejecutar el proceso de training y testing. Este proceso se repite K veces, con distintos datasets de training y testing, computando en cada una de las iteraciones el accuracy, que finalmente es promediado para obtener un resultado final. Este método tiene la ventaja de que cada muestra está en el subconjunto de testing una única vez y en el dataset de training $k-1$ veces, por lo que la variación en las predicciones con distintas muestras se incrementa a medida que k crece. Sin embargo, tiene la desventaja de ser computacionalmente costoso.

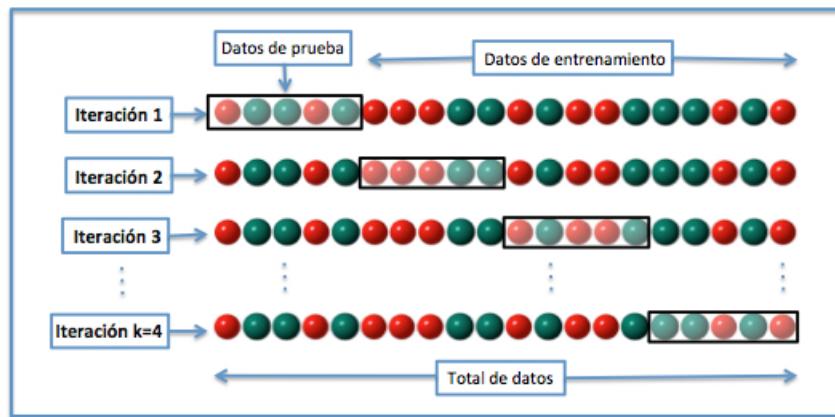


Figura 6.28: Ejemplo gráfico de los subconjuntos generados con K-Fold cross-validation

- Leave-one-out cross-validation (LOO CV): Éste método consiste en realizar una partición de $k-1$ muestras como la partición de entrenamiento y emplear únicamente una muestra para la partición de testing y luego, realizar el entrenamiento y la validación para esa muestra en particular. Este proceso se repite K veces, con distintas muestras asignadas a la partición de testing, y al igual que K-fold, se obtiene la media de los resultados predecidos para computar el resultado final.

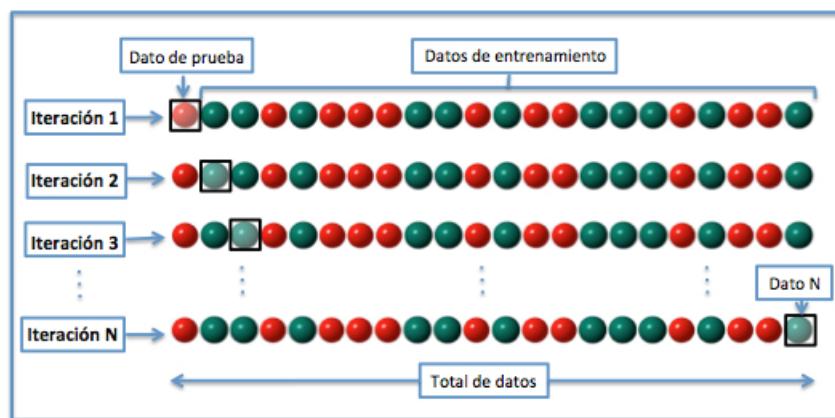


Figura 6.29: Representación gráfica de la generación de subconjuntos para LOO CV

Una variación de éste método es Leave p -out cross-validation, donde se realiza el mismo proceso pero se consideran p elementos para el conjunto de testing en lugar de uno solo.

- Curva ROC: Receiver Operating Characteristic curve o ROC, es una representación gráfica que muestra la eficiencia de uno o varios métodos de clasificación, representando en el eje Y la tasa de verdaderos positivos (TP) y en el eje X la tasa de falsos positivos(FP). De esta manera, empleando los valores proporcionados por distintas matrices de confusión, se puede comparar la eficiencia de clasificación donde cuanto más pronunciada sea la curva hacia la esquina superior izquierda del diagrama, mayor será la cantidad de muestras clasificadas correctamente. Mientras que, si la curva del clasificador se encuentra por debajo de la línea diagonal que divide la esquina inferior izquierda y superior derecha, significará que los resultados proporcionados por éste son peores que un valor aleatorio. Adicionalmente, extrayendo características de la curva se puede comparar el rendimiento del clasificador de manera numérica, uno de los valores empleados es el área de bajo de la curva (Area Under the ROC Curve,AUC) cuyo valor se encuentra en el rango 0-1, mejorando la eficiencia de clasificación cuanto más cercano a 1 es esta métrica.

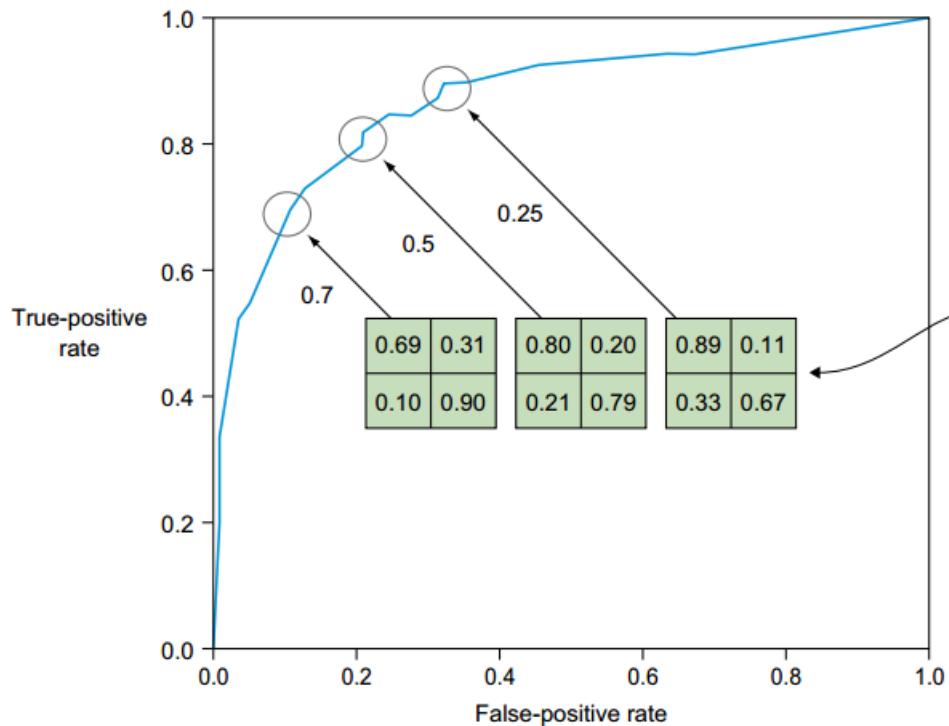


Figura 6.30: Ejemplo de curva ROC

1.2. Aplicaciones de ML

Machine Learning cuenta con un amplio campo de aplicaciones entre las que se encuentran:

- Transcripción de caracteres, donde el modelo observa una representación no estructurada de datos y lo transcribe una forma textual discreta. Un ejemplo de ésto es el reconocimiento óptico de caracteres, donde al modelo le son presentados imágenes conteniendo un texto y se solicita que retorne la representación numérica de éste(por ejemplo en formato Unicode o ASCII). O en el reconocimiento de voz, donde el modelo en base a ondas de sonoras debe emitir una secuencia de caracteres o decodificar las palabras que fueron habladas en el audio de entrada.
- Traducción de idiomas, donde a partir de un conjunto de símbolos en algún idioma, el modelo debe convertirlos en una secuencia de símbolos en otro idioma, aplicado generalmente a procesamiento de lenguaje natural.
- Salidas estructuradas de información, que involucran tareas donde la salida es una estructura de datos que contiene múltiples valores (como un vector) que describe relaciones entre los datos. Un ejemplo de ésto es el parseo, donde se mapea una sentencia en lenguaje natural en un árbol que describe la estructura gramatical etiquetando nodos de árboles como verbos, sustantivos o adverbios; Otro caso de aplicación, es la segmentación de imágenes donde se asigna a cada pixel una categoría específica, como por ejemplo una red neuronal, que es usada para anotar las ubicaciones de las calles en una fotografía aérea.
- Data Mining, donde se cuenta con grandes cantidades de información en bases de datos y éstas contienen información valiosa para el negocio, acerca de las relaciones de los datos, que pueden ser descubiertas automáticamente. Un ejemplo de ésto es aprender reglas generales que ayuden a realizar la evaluación de otorgación de créditos a un posible solicitante, por medio del análisis de información de bases de datos financieras.
- Ranking de páginas web, donde el usuario envía una consulta a un motor de búsqueda, que proporciona páginas web ordenadas según el nivel relevancia. Para lograr ésto el motor de búsqueda necesita conocer cuales páginas son relevantes y concuerdan con la consulta especificada, adquiriendo para ésto información de distintas fuentes: la dirección web de cada página, el contenido de la

misma, la frecuencia con la que varios usuarios entran a estos sitios para la misma consulta, o desde ejemplos de consultas similares en combinación páginas web rankeadas manualmente. Machine Learning es usada durante este proceso para automatizar y mejorar la precisión de las consultas al momento de diseñar un motor de búsqueda.

- Aplicaciones de reserva por Internet que emplean filtrado colaborativo, donde se analiza la información relacionada a productos consumidos anteriormente por un usuario, con el fin de predecir futuros hábitos de consumo y de esta manera ofrecerle productos similares. Un ejemplo de aplicaciones que emplean este tipo de característica son Amazon y Netflix, donde los productos o servicios que se ofrecen a un usuario se predicen empleando los datos de consumo de otros usuarios con preferencias de consumo similares.
- Aplicaciones donde los desarrolladores no cuentan con suficiente conocimiento del dominio como para programar algoritmos efectivos, como son por ejemplo las aplicaciones de control de seguridad, que emplean el reconocimiento facial en base a imágenes o grabaciones de video. Estas aplicaciones deben realizar una clasificación del rostro en una imagen para determinar si es un usuario registrado o un desconocido, lidiando con diferentes condiciones de iluminación, expresiones faciales, diferentes corte de pelo,etc. debiendo para ello, aprender las características más relevantes de cada rostro con el fin sortear estas dificultades.

1.3. Beneficios del uso de ML

Dentro de los beneficios que provienen de la utilización de ML respecto a las alternativas de análisis manual de datos, instrucciones programadas explícitamente y modelos estadísticos simples se encuentran:

- Precisión. ML emplea datos para descubrir el modelo con desempeño óptimo para el problema del dominio y  medida que se disponga de más datos, mejor definido estará el problema y, en consecuencia, se incrementará la precisión de éste automáticamente.
- Automatizado. A medida que las respuestas se validan o se descartan, el modelo de ML aprenderá nuevos patrones automáticamente, por lo que los usuarios del modelo contribuirán a su mejora incrementalmente.
- Rapidez. ML puede generar respuestas en cuestión de milisegundos a medida que ingresan mayor cantidad de muestras de datos, permitiendo que el sistema mejore en tiempo real.
- Personalizable. Varios problemas dirigidos por datos pueden ser resueltos con ML, ya que el modelo es construido basándose en datos propios, y puede ser configurado para optimizar aquellas métricas que sean de valor para el negocio.
- Escalable. A medida que el negocio donde se aplica el modelo de ML crece  el modelo escala para manejar tasas de datos que crecen incrementalmente, llegando a manejar grandes cantidades de datos si se dispone del hardware apropiado.

En la siguiente sección, se exponen el funcionamiento particular de los principales algoritmos empleados en ML para la generación de modelos de predicción.

2. Mecanismos para Machine Learning(ML)

2.1. Árboles de decisión(Tree)

Los árboles de decisión (Classification And Regression Tree, CART) son un mecanismo de ML de aprendizaje supervisado que permite generar un modelo  que realiza predicciones en base a un conjunto de reglas inferidas de los datos de entrenamiento, las cuales pueden ser representadas por un diagrama de árbol. Los árboles de decisión clasifican instancias ordenándolas desde un nodo raíz hasta llegar a nodos hoja, lo que en última instancia proporciona la predicción de una muestra, de esta manera cada nodo en el árbol especifica alguna prueba de algún atributo de la muestra de entrada y, cada rama que desciende

de ese nodo corresponde a uno de los posibles valores para este atributo. De esta forma, el conjunto de reglas desde la raíz a una hoja del árbol representan las reglas de clasificación. Entonces, la clasificación de una instancia comienza en el nodo raíz del árbol testeando el atributo especificado por este nodo, y luego desplazándose por la rama del árbol correspondiente al valor del atributo en el ejemplo dado y, este proceso se repite nuevamente para el nodo actual, hasta llegar a un nodo hoja del árbol.

La construcción del árbol se puede realizar por medio de distintos algoritmos, aunque la mayoría se basa en un algoritmo base top-down de búsqueda ambiciosa, que consiste en probar cada una de las features individualmente, para determinar cual de todas éstas clasifica de manera más eficiente las muestras de entrenamiento, computando para ésto una métrica denominada ganancia de la información (information gain), que mide que tan eficazmente un atributo separa las muestras del dataset de entrenamiento según sus labels. De esta forma, el atributo con la mejor división es seleccionado y establecido como la raíz del árbol. Luego, un nodo hijo del nodo raíz es creado para cada posible valor del nodo raíz y, las muestras de entrenamiento se ordenan de acuerdo al nodo hijo al que pertenecen. Este proceso se repite para cada una de las muestras de entrenamiento asociadas a cada nodo hijo, con el fin de seleccionar el mejor atributo posible para realizar la prueba en ese punto en el árbol. Este proceso continua hasta que la creación de ramas en el árbol no produce una mejora de predicción significativa, o hasta que la cantidad de muestras en cada uno de los nodos hoja del árbol es menor que un valor mínimo previamente establecido. Este proceso es un método de búsqueda ambiciosa en el cual el algoritmo nunca retrocede sobre divisiones realizadas anteriormente en el árbol, para reconsiderar otras alternativas.



Figura 6.31: Ejemplo gráfico de la organización de un árbol de decisión

Los árboles de decisión también pueden sufrir de overfitting, ya que éstos pueden ser creados con estructuras complejas de división que no generalicen de manera eficiente nuevos datos de prueba, por lo que existen varias aproximaciones para evitarlo denominadas poda o pruning, donde se eliminan ramas del árbol con el fin eliminar relaciones redundantes. Dependiendo del momento en que se realiza la poda éstas se clasifican en dos clases principales:

- Poda temprana (early pruning), donde el crecimiento del árbol se detiene, antes de que alcance el punto donde memoriza el ruido y los datos de entrenamiento. Una de las aproximaciones de este tipo denominada reduced-error pruning consiste en dividir el dataset en subconjuntos complementarios de training y testing. Luego, se considera cada uno de los nodos en el árbol como un candidato a ser podado, donde podar un nodo implica remover el subárbol que se encuentra debajo de éste, convertirlo en un nodo hoja y asignarle los resultados de clasificación asociados al subárbol de ese nodo. Así, los nodos se remueven si el árbol luego de la poda de un nodo tiene un desempeño de predicción menor que el árbol original sobre el dataset de testing. Ésto tiene el efecto de que cualquier nodo hoja agregado debido a regularidades coincidentes en el dataset de training es probable que sea podado, ya que es improbable que estas coincidencias también estén presentes en el dataset de testing. De esta forma se podan los nodos de manera iterativa, siempre seleccionando aquellos que incrementen el rendimiento del árbol sobre el dataset de testing, hasta que la poda de nodos produzca un decremento de la precisión del modelo.
- Poda tardía (post pruning) que permiten que el árbol se construya con overfitting y luego se realiza la poda para eliminar esta característica. Estos métodos, luego de que se entrenó con el dataset de training permitiendo el overfitting, se convierte el árbol en un conjunto de reglas construyendo una

regla por cada camino desde la raíz del árbol hasta un nodo hoja. A continuación, se poda cada regla, removiendo cualquier otra precondición (secuencia de tests de atributos de la raíz al nodo hoja) que produzcan una mejora del accuracy, la misma, dejando intactas las reglas empeoran esta métrica. Finalmente, se ordenan las reglas podadas por su accuracy estimada, y se las considera en esta secuencia cuando se clasifican posteriores muestras.

2.2. Redes Neuronales(ANN)

Las redes neuronales artificiales (Artificial Neural Network, ANN) son modelos inspirados en los sistemas neuronales de los cerebros animales, que se encuentran constituidos por neuronas interconectadas, que forman una red, comunicada a través de impulsos eléctricos. Análogamente, las redes neuronales artificiales se componen de neuronas artificiales, que aceptan un conjunto de valores de entrada reales, regulados por un conjunto de pesos w_i que determinan la relevancia de la contribución cada una de las entradas y que se ajustan automáticamente, durante la etapa de aprendizaje de la red. Estas neuronas computan una función en base a los valores de entrada y los pesos, y dependiendo de la relevancia de estos valores con respecto al problema, ésta se activa retornando un valor 1 si el valor computado supera cierto límite (threshold) y -1 en caso contrario.

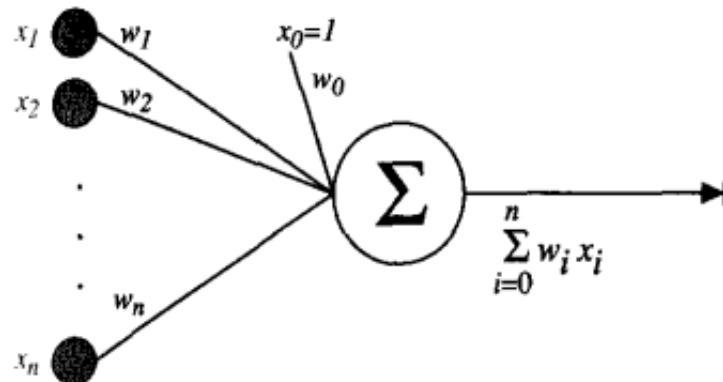


Figura 6.32: Ejemplo de neurona artificial

Por lo general, las redes neuronales se organizan en capas, donde cada una de las capas se compone de un conjunto de neuronas artificiales interconectadas con neuronas de la capa anterior y la siguiente, con el fin de recibir solamente información desde la capa de procesamiento anterior y solo enviar información a la capa de procesamiento siguiente. Entre estas capas se distinguen la capa de entrada (input layer) donde se ubican las neuronas que reciben los datos de las muestras que la red procesará, la capa de salida (output layer) que contiene las neuronas que brindan el resultado final y las capas ocultas o intermedias (hidden layers) que solamente contienen información intermedia empleada durante el procesamiento. Las redes neuronales pueden contar con varias capas ocultas, dependiendo de que tan complejo sean las relaciones del problema que se busca resolver.

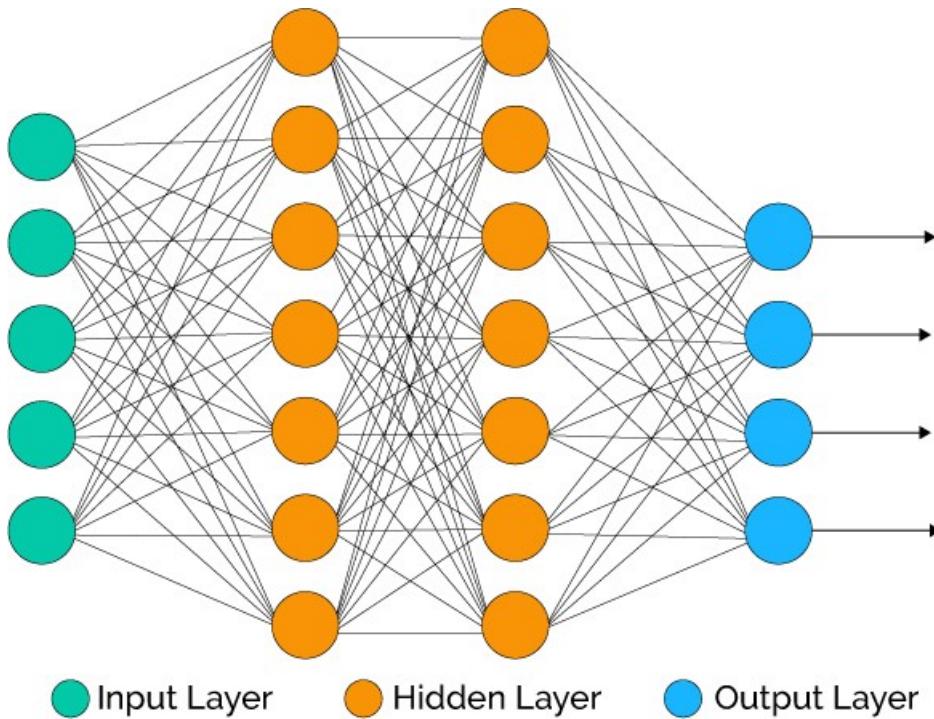


Figura 6.33: Ejemplo de ANN multicapa

2.3. Maquinas de soporte de Vectores(SVM)

Las máquinas de soporte de vectores (Support Vector Machines, SVM) son un mecanismo empleado durante el aprendizaje supervisado para resolver problemas de clasificación y regresión, basadas en la utilización de hyperplanos. En esta técnica las p features asociadas a una cantidad n de muestras de entrenamiento, se organizan en un espacio p -dimensional (siendo cada muestra considerada como un vector de dimensión p) donde cada una de las features representa la coordenada de una muestra en particular. De esta forma, SVM consiste en realizar una separación de los datos en el espacio p -dimensional por medio de hyperplanos, donde un hyperplano para un espacio de dimensión p es un subespacio de dimensión $p-1$ que es capaz de separar los datos de entrenamiento en diferentes clases (para clasificación) o, encontrar la función que define la distribución de las muestras (en regresión), según las features especificadas para las muestras.

Así, la generación de un modelo de SVM consiste en computar y evaluar varios hyperplanos separadores de los datos de entrada y seleccionar de entre éstos, un hyperplano óptimo (o hyperplano de margen máximo) cuya distancia hacia las muestras más cercanas, sea la mayor. Por lo tanto, para obtener el valor de éste se calcula la distancia perpendicular entre el hyperplano divisor y las muestras de todas las clases y se consideran las muestras de ambas clases, cuya distancia al hyperplano sea menor. Estas distancias se conocen como margen (margin). Así el hyperplano óptimo, es el hyperplano separador cuyo margen entre las muestras a menor distancia es el mayor, por lo que mantiene una distancia mayor a las muestras de entrenamiento mas cercanas de ambas clases, y establece la mayor distancia de separación entre ambas clases. De esta forma las coordenadas de las muestras más cercanas al hyperplano conforman los vectores de soporte en el espacio p -dimensional y, soportan el hyperplano óptimo ya que si estos puntos fueran movidos levemente, entonces el hyperplano óptimo también se movería para mantener este margen, por lo que este hyperplano depende de los vectores de soporte. Mientras que, si las coordenadas de cualquier otra muestra se modifican, desplazándose en el p -espacio no afectaría el margen del hyperplano óptimo, siempre y cuando no se modifique la clase a la que ésta fue asignada.

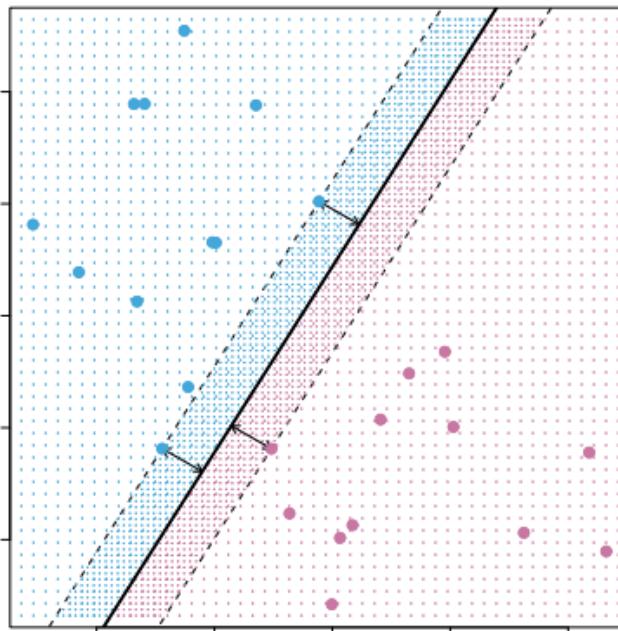


Figura 6.34: Ejemplo de hyperplano separador. La línea negra es el hyperplano que divide las muestras de entrenamiento, y las coordenadas de las tres muestras que son atravesadas por la línea punteada constituyen los vectores de soporte para ese hyperplano.

De esta forma, las SVMs permiten realizar la clasificación con hyperplanos en conjuntos de datos que se encuentran separados linealmente, sin embargo existen situaciones donde éstos no se pueden dividir linealmente, por lo que el desempeño de un clasificador lineal es considerablemente bajo en estas circunstancias. Para solucionar este problema, las SVM emplean funciones de kernel que convierten el espacio de las features de entrada a un espacio de mayores dimensiones, siendo éstos cuadráticos, cúbicos, polinomiales o de orden superior, con el fin de lograr encontrar un hyperplano en este nuevo espacio, que separe las muestras con una mejor precisión. Existen diferentes tipos de kernels que se emplean para lograr ésto, entre los que se encuentran:

- Kernel Linear: Dado para dos vectores x, y correspondientes a las muestras entrada sin modificar la función de kernel, se define de la siguiente forma:

$$K(x, y) = (x, y) \quad (6.4)$$

- Kernel Radial Basis Function(RBF): Este kernel dados dos vectores x, y correspondientes a dos muestras representadas como vectores en el espacio de entrada, se define por medio de la siguiente fórmula, donde γ es un parámetro libre mayor a cero ajustable al problema:

$$K(x, y) = \exp(-\gamma||x - y||^2) \quad (6.5)$$

- Kernel Polynomial: El kernel polinomial para polinomios de grado d se computa por medio de la siguiente fórmula, donde c es un parámetro libre que ajusta la influencia del orden superior de la función contra términos de orden superior en el polinomio, x^*y^* son las representaciones vectoriales de las muestras de entrada:

$$K(x, y) = (xy + c)^d \quad (6.6)$$

Debido a que SVM es solamente una algoritmo de clasificación binario, se han desarrollado estrategias para la clasificación multiclas (o multilabel), con el fin de discriminar entre más de dos clases, entre las que se encuentran las siguientes:

- Uno contra el resto (One-vs-One,OvO): En esta aproximación, dadas N clases se deben entrenar $N(N-1)/2$ clasificadores binarios, generando uno por cada posible combinación de clases y posteriormente, entrenarlos con datos de entrenamiento asociados a las clases que deben predecir. Luego, durante la etapa de predicción se emplea un esquema de votación, donde todos los clasificadores

predicen la misma muestra de testing y, la clase que tenga mayor cantidad de votos positivos es la clase a la que pertenece la muestra.

- Uno contra el Resto (One-vs-Rest,OvR,OvA): Esta técnica consiste en para un problema con N clases, entrenar N clasificadores que emplean todas las muestras del dataset de training, clasificando como positivas aquellas que pertenecen a la clase que el clasificador tiene asignada y negativa si pertenece a cualquier otra clase. Así, al recibir una muestra de entrada, cada uno de los clasificadores genera un valor real, que es un score de confianza que indica la probabilidad de que esa muestra pertenezca a esa clase, y la clase para la que la muestra de testing genera el score más alto, es la clase en la que ésta se asigna.

3. Selección de features para ML en PCL

Con respecto a la elección de features para ML, debido a que únicamente algunas grietas podían ser aisladas aplicando la metodología de cropeado de muestras (Ver *Metodología de pre-procesado de muestras (Pipeline de Cropeado)*), ya que durante la recolección de muestras se observó que en la práctica existían grietas que no poseían profundidad significativa para ser detectadas por el sensor, sino solamente grosor y largo suficiente para ser apreciadas como grietas, se optó por clasificar solo aquellos tipos de fallas que poseen una profundidad necesaria para ser aisladas por descriptores que computan información geométrica relacionada con los ángulos entre las normales de la superficie. Debido a ésto, se seleccionó un subconjunto del rango completo de descriptores locales y globales que PCL ofrece, acorde a las capacidades de computo disponibles y a las propiedades de las normales que éstos computan, siendo los descriptores testeados los siguientes:

- Fast Point Feature Histogram(FPFH)
- ViewPoint Feature Histogram(VFH)
- Global Radious-based Surface Descriptor(GRSD)
- Ensamble Shape of Functions(ESF)

3.1. PFH-FPFH

Los puntos orientados, compuestos por el vector de coordenadas y el vector normal del punto, son computacionalmente eficientes y rápidos de generar, sin embargo, no son capaces de capturar información geométrica significativa alrededor de un punto, por lo que se necesita un descriptor que sea capaz de capturar información geométrica respecto de la curvatura, en base a los vecinos de un punto. Para ello se diseña Point Feature Histogram (PFH), que permite generalizar la curvatura media en base a los k -vecinos de un punto, empleando histograma de múltiples valores, que se caracteriza por ser invariante a la posición que adopta la superficie, robusto ante ruido y diferentes tipos de densidades en las muestras, e invariante a las rotaciones y traslaciones 3D. La implementación de este descriptor en PCL, se basa en el trabajo en [36] donde se define formalmente la metodología para computar las características locales geométricas partiendo desde una malla de triángulos.

El funcionamiento de PFH consiste en representar las relaciones entre puntos en un k-vecindario dados los puntos y sus normales estimadas, de manera que se capture con la mayor precisión posible las variaciones en la superficie, tomando en consideración todas las interacciones entre las direcciones de las normales estimadas. De esta forma, las features de un punto dependen en gran parte de las estimaciones de las normales para los puntos. Formalmente, PFH para cada punto p , perteneciente a una nube de puntos realiza los siguientes pasos:

- Primero, considera aquellos k vecinos que se encuentran a una distancia menor a un radio r para el procesamiento, ubicándose en el centro de la esfera el punto de entrada p , y produciendo un conjunto de puntos $P = \{pj_1, pj_2, \dots, pj_n\}$, y un conjunto de normales asociadas a cada punto $N = \{Nj_1, Nj_2, \dots, Nj_n\}$:

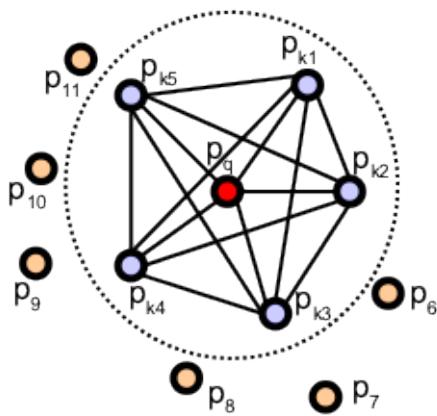


Figura 6.35: Ejemplo de los pk-vecinos considerados como entrada al algoritmo

- Luego, para cada par de puntos en el conjunto P de vecinos e incluyendo el punto central $p,*pj1*$ y $pj2$, y sus normales estimadas, se selecciona un punto ps como origen y un punto pt como objetivo, siendo el punto origen el que tiene el menor ángulo entre la normal de ese punto y un vector imaginario que conecta ps y pt ; Matemáticamente hablando, se debe cumplir la siguiente ecuación: $\|n1 \cdot (p2 - p1)\| \leq \|n2 \cdot (p2 - p1)\|$. Posteriormente, para computar las diferencias entre los puntos y sus normales, se procede a definir tres vectores base u , v y w alrededor del punto origen, siendo u el vector normal ns asociado al punto origen y definiéndose estos vectores por medio de las siguientes fórmulas, donde x es el producto cruz entre dos vectores y $\|\cdot\|$ es la norma euclídea del vector:

$$U = ns \quad (6.7)$$

$$v = u * (pt - ps) / \|pt - ps\| \quad (6.8)$$

$$w = uxv \quad (6.9)$$

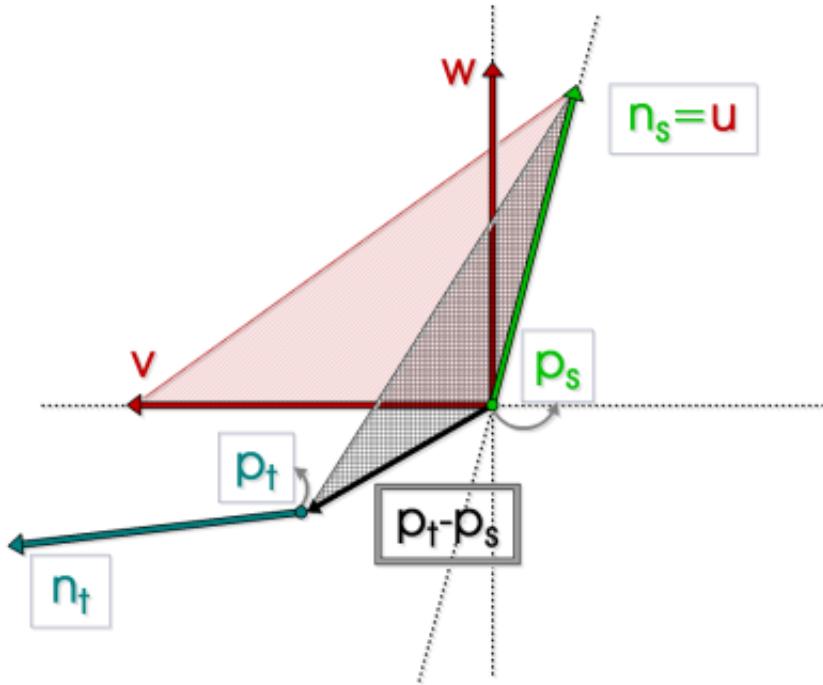


Figura 6.36: Asignación de ejes al punto origen

- A continuación, empleando los vectores uvw y las coordenadas y normales de los puntos, se pueden calcular la diferencia entre las dos normales de la siguiente manera, siendo \cdot el producto escalar entre dos vectores y d la distancia Euclíadiana entre ps y pt , $d = \| ps - pt \|$:

$$\alpha = v \cdot nt \quad (6.10)$$

$$\phi = u \cdot (pt - ps)/d \quad (6.11)$$

$$\theta = \arctan(w \cdot nt, u \cdot nt) \quad (6.12)$$

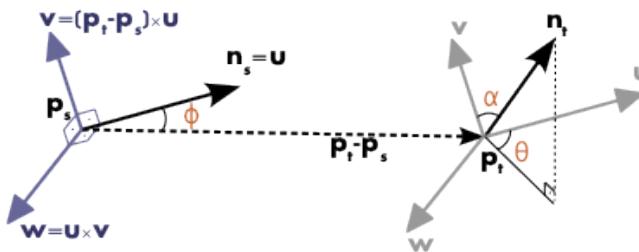


Figura 6.37: Ángulos y sus correspondencias con las normales

- Finalmente, las frecuencias de las tuplas (o features) del descriptor ($\alpha, \phi, \theta, *d*$) por cada punto se organizan en un histograma, y se divide el espectro de valores que puede adoptar cada feature en b subdivisiones y se cuentan las frecuencias de valores en cada subdivisión. Así, el número de subdivisiones por cada feature del histograma, que se pueden formar utilizando n cantidad de features es d^n , en este ejemplo es la cantidad de divisiones div^4 ya que se emplean 4 valores en cada feature. La implementación PFH de PCL, emplea 5 subdivisiones de histograma por feature y no incluye las distancias, lo que resulta en $5^3 = 125$ valores float de features. Este descriptor se define en la clase que define el tipo de punto pcl::PFHSignture125.

Debido a que la complejidad computacional de PFH es del orden $O(n)$, esto puede resultar en cuellos de botella de procesamiento para aplicaciones en tiempo real o con considerable cantidad de muestras, por lo que para solventar este inconveniente se puede emplear FPFH. FPFH consiste en calcular para cada punto p de la nube, los valores de (α, ϕ, θ) análogamente a como se realiza con PFH, solo que este cálculo se realiza solamente entre el punto p y los k -vecinos de éste, denominando este valor como $SPFH(p)$. A continuación, el valor $SPFH(p)$ es ponderado calculando los features para los puntos vecinos p_k , $SPFH(pk)$, y utilizando las distancias wk entre cada punto p_k y el punto p , empleando la siguiente fórmula:

$$FPFH(\mathbf{p}_q) = SPFH(\mathbf{p}_q) + \frac{1}{k} \sum_{i=1}^k \frac{1}{\omega_k} \cdot SPFH(\mathbf{p}_k)$$

Figura 6.38: Fórmula para calculo de descriptor FPFH(p)

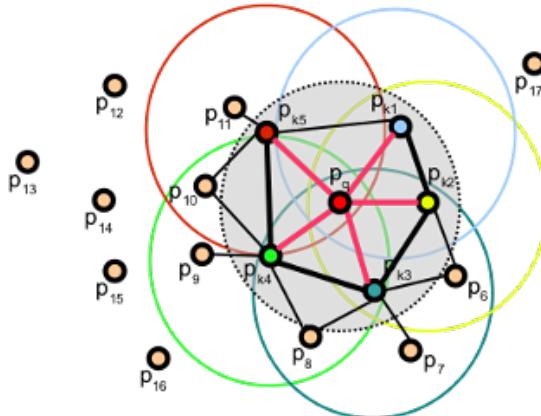


Figura 6.39: Esquema de las relaciones que se consideran para calcular las features de FPFH. El punto central p o pq se encuentra en el centro, las relaciones entre p y sus k -vecinos empleados para computar $SPFH(p)$ se encuentran resaltados en rojo y las relaciones entre los k -vecinos empleadas para ponderación se encuentran remarcadas en negro.

3.2. VFH

VFH es una variación de FPFH que se emplea para la identificación y reconocimiento de posición, donde se aprovecha la velocidad de procesamiento y la potencia de este descriptor y se agrega el componente de punto de visión, que no es afectado por variaciones en la escala de los datos. VFH agrega el punto de visión a FPFH, computando un histograma de ángulos con la diferencia de ángulos entre la normal del punto de visión y cada uno de los puntos de la superficie capturada:

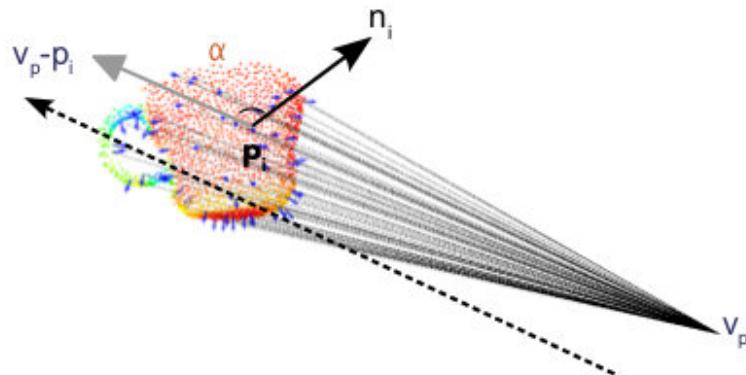


Figura 6.40: Representación gráfica del primer componente entre el punto de visión y cada uno de los puntos de la superficie.

Además se agrega un componente de forma de superficie, generando para ésto un histograma FPFH extendido, donde se incorpora la computación de los ángulos relativos entre las normales en cada punto de la captura y el centroide del objeto (punto central):

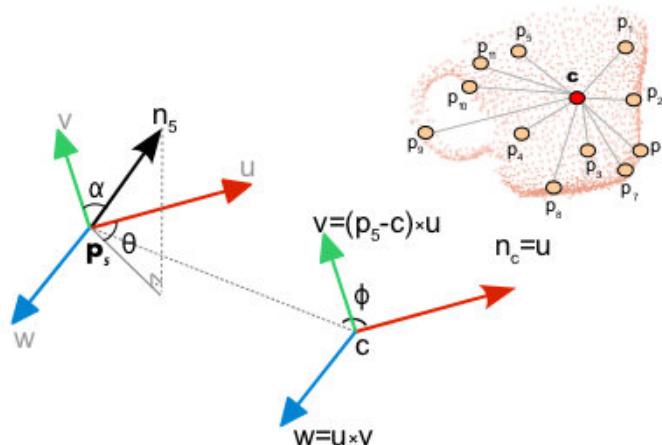


Figura 6.41: Incorporación de la diferencia entre normales de puntos y centroide del objeto

3.3. GRSD

Este descriptor global emplea el descriptor local Radious-based Surface Descriptor (RSR), que se basa en la descripción geométrica de una superficie por medio del cálculo de información radial, computada a través de información inherente a los puntos vecinos. El funcionamiento de este algoritmo se basa en establecer una relación entre los ángulos de las normales λ , la distancia entre éstas d y el radio de una superficie r por medio de la siguiente fórmula:

$$d = r * \alpha \quad (6.13)$$

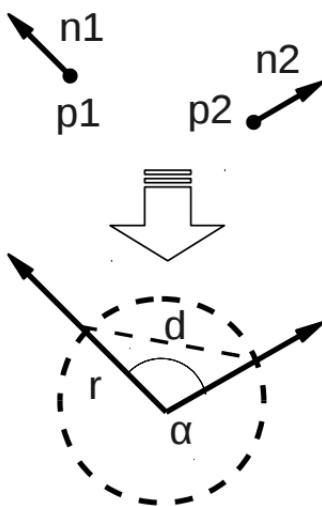


Figura 6.42: Representación gráfica ángulo, el radio y la esfera

Por lo tanto, para un punto p dado y cada uno de sus puntos vecinos, se calcula la diferencia entre normales, por medio del cálculo del ángulo α , la distancia entre las normales d y, con estos valores, se obtiene el radio r de la esfera que engloba tanto a p y su normal, como a uno de sus puntos vecinos y la normal de éste. Este proceso, genera un conjunto de radios que describe cada una de las esferas que contiene a p con uno de sus vecinos, y de todas éstas únicamente se agregan al descriptor de ese punto los radios máximo y mínimo.

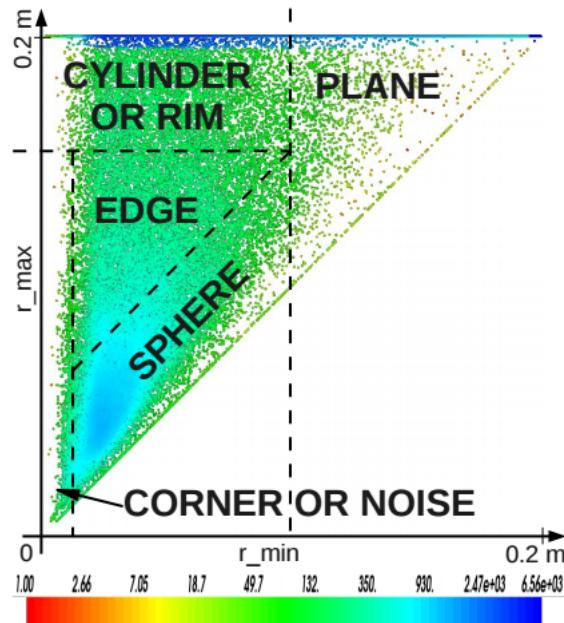


Figura 6.43: En el gráfico de densidad, se muestra un gráfico de número /densidad de puntos en un rango de 1cm para diferentes objetos, exemplificando la delimitación del tipo de superficie (plano,esfera,,cylinder,ruido) según el rango de radios mínimo y máximo.

Esta método cuenta con la ventaja de ser fácil de computar y aún así mantener su capacidad de descripción, y se emplea principalmente para la detección de puntos que pertenecen a distintas superficies.

GRSD consiste en generar agrupamiento de puntos (o voxels) en lugar de puntos individuales, donde cada voxel tiene un ancho de 2.5 cm, y se procede a computar los radios máximos y mínimos

entre p y un conjunto de vecinos y a etiquetar cada uno de los voxels según su valor de radio, siendo un plano si el $\text{radio_mínimo} > 0.1$, una superficie cilíndrica si no es un plano y $\text{radio_máximo} > 0.175$, un borde/esquina o ruido, si no es cilíndrico y $\text{radio_mínimo} < 0.015$, esférico si no es un borde y $\text{radio_máximo} - \text{radio_mínimo} < 0.05$ y otra superficie si no es ninguna de las anteriores. Una vez etiquetados todos los voxels, se computa un histograma global que describe las relaciones entre los clusters, en base a las intersecciones de cada superficie con el resto.

3.4. ESF

Este descriptor no emplea ningún tipo de pre-procesamiento, como las normales, sino que inicialmente emplea un conjunto de voxels de la superficie (voxel grid). Este algoritmo consiste en iterar a través de cada uno de los puntos de la nube y, de cada punto seleccionado, se eligen 3 puntos aleatorios y se computan las funciones de forma: D2, proporción D2 (D2 ratio), D3 y A3, donde cada función genera histogramas que describen la relación geométrica entre puntos de la figura, produciendo un total de 10 sub-histogramas cada uno de 64 divisiones, por lo que el tamaño del histograma final es de 640. A continuación se detallan las funciones de forma:

- La función D2, computa las distancias entre los 3 puntos elegidos, formando 3 pares distintos, y para cada par verifica si la línea que conecta ambos puntos yacen completamente dentro de la superficie, enteramente afuera de la figura (no formando parte del objeto) o, abarcando una porción del objeto y una porción del espacio libre. Dependiendo de esta condición, se asigna el valor de distancia a un histograma IN, OUT o MIXED respectivamente.

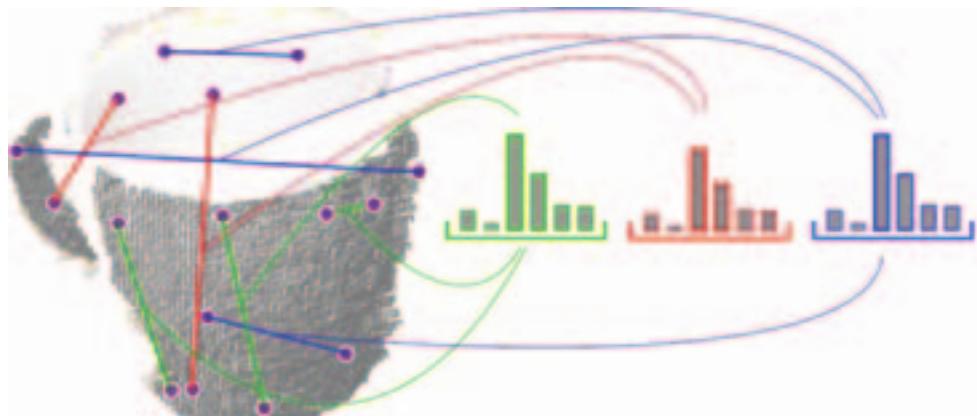


Figura 6.44: Representación gráfica de la función D2

- La proporción D2, consiste en generar un histograma que represente la proporción entre partes de la línea dentro de la superficie y fuera de ésta, donde el valor será cero si la línea está completamente afuera, uno si esta completamente adentro, y un valor intermedio si se encuentra tanto dentro como fuera.
- La función D3, computa la raíz cuadrada del área del triángulo formada por los 3 puntos, y es agrupado, al igual que D2, en 3 histogramas IN, OUT y MIXED independientes de los que emplea D2.

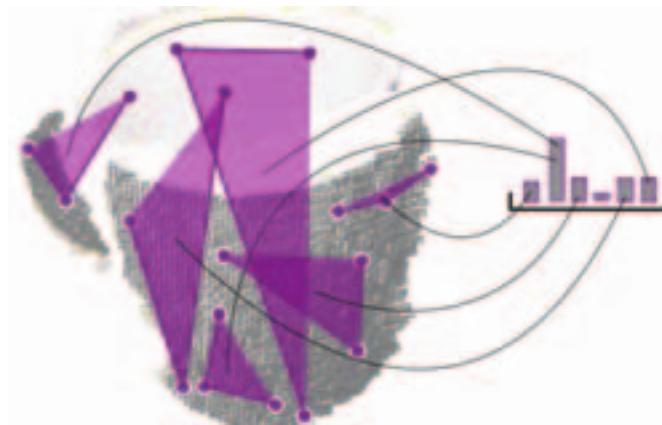


Figura 6.45: Representación gráfica de la función D3

- Finalmente, la función A3 computa el ángulo formado por los puntos del triángulo, y luego este valor es asignado a un histograma IN, OUT o MIXED, dependiendo de que superficie abarca la línea que se encuentra opuesta al ángulo calculado. Estos 3 histogramas son independientes de los que se emplean en D2 y D3.

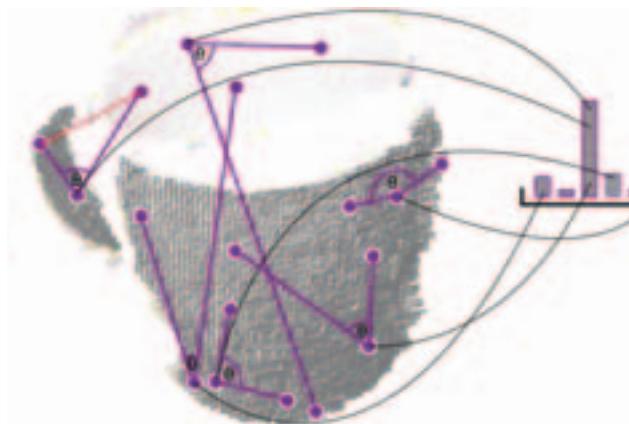


Figura 6.46: Representación gráfica de la función A3

4. Metodología de pre-procesado de muestras (Pipeline de Cropeado)

Debido a la cantidad numerosa de puntos que se encuentran en una captura realizada por el sensor (aproximadamente 300.000 puntos) y, a que se deseaba abstraer solo aquellas características propias de cada tipo de falla, se procedió a aplicar una serie de algoritmos como parte del pre-procesado de datos en machine learning o Pipeline de Cropeado, con el fin de reducir la cantidad de puntos de cada muestra y de sólo calcular el descriptor con los puntos principales de una falla. Este pipeline de cropeado, se compone de los siguientes pasos:

1 - Eliminación de ruido con Statistical Removal: Debido a que la densidad de puntos de una captura puede variar, bajo diversas condiciones tales como: La cantidad de luz solar presente o la posición del sensor con respecto al pavimento, es necesario eliminar para cada captura aquellos valores extremos (o outliers), que pueden interferir con la computación features de la muestra. Para ello, PCL ofrece un algoritmo de filtrado denominado Statistical Outlier Removal, el cual para cada punto en la nube de entrada computa la distancia media de éste hacia todos sus vecinos, y asumiendo que las distancias siguen una distribución estadística Gaussiana con una media y desvío estándar, elimina de la nube aquellos puntos cuyas distancias estén fuera del intervalo definido por la media y el desvío estándar de la distribución. |

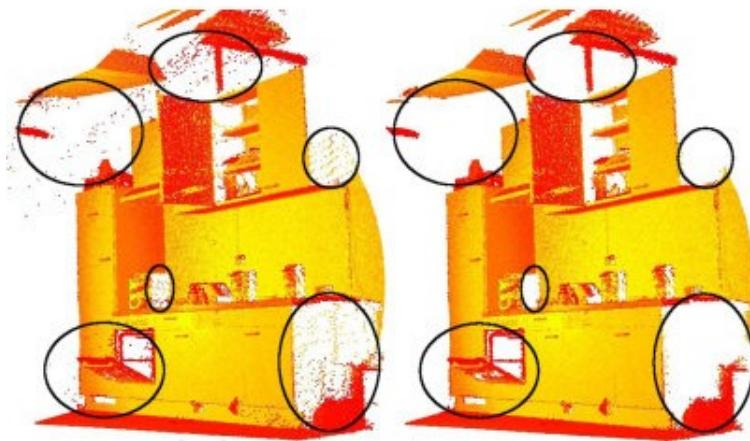


Figura 6.47: Izquierdo: Ejemplo de nube de puntos sin el filtro Statistical Outlier Removal. Derecha: Ejemplo de nube de puntos con el filtro de Statistical Outlier Removal.

2 - Downsampling con Voxel Grid (Extracción de Keypoints): Se asocia con el nombre de voxel a un conjunto de puntos que forman una mínima unidad cubica (grilla en 3D) de un objeto tridimensional, de la misma forma que un pixel es la mínima unidad en una imagen en 2D. El algoritmo de Voxel Grid en PCL, permite reducir la cantidad de elementos de una nube, realizando una división de una nube de puntos en voxels, y computando en base a éstos el centroide (centro del voxel grid) que se tomará como el punto que representa al resto de los puntos en el voxel grid. Estos puntos se denominan keypoints o puntos de interés y son aquellos puntos principales que aportan mayor información respecto de la estructura del pavimento a la SVM. Éstos se caracterizan por ser:

- Estables con respecto a interferencias locales y globales en el dominio de la imagen, como variaciones de iluminación y brillo.
- Distintivos para la caracterización efectiva de una superficie, y ricos en contenido en términos de color y textura.
- Tienen una posición claramente definida y se pueden obtener repetidamente con respecto a ruido y variaciones en el punto de visión.
- No es afectado por variaciones de escala, por lo que son ideales para procesamiento en tiempo real como también procesamiento en distintas escalas.

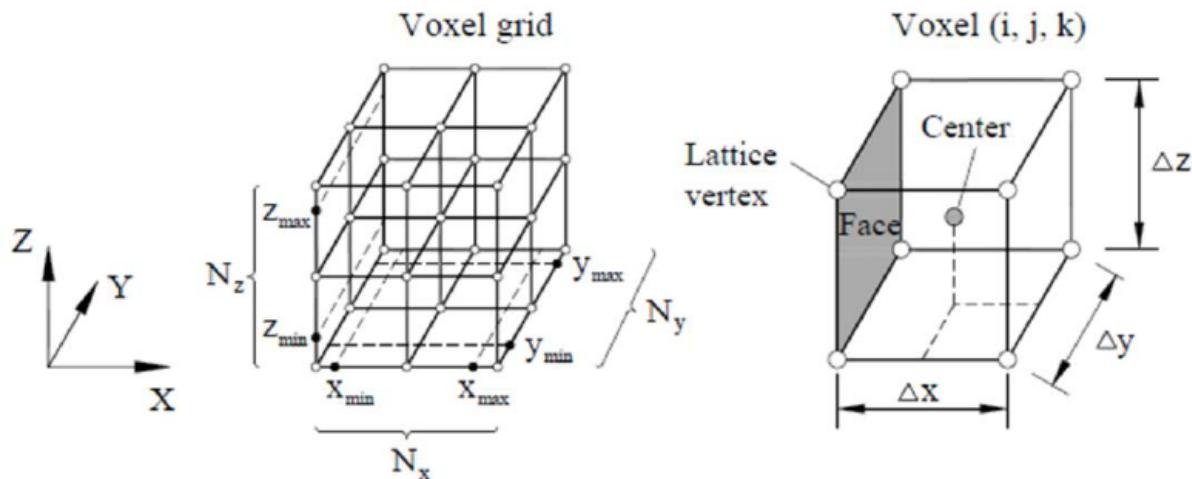


Figura 6.48: Estructura de un voxel y voxel grid en 3D

3 - Segmentación con algoritmo de Planar Segmentation: El algoritmo empleado para la segmentación en PCL fue RANSAC (Random Sample Consensus) configurado para el modelo de plano (PLANE), ya que se deseaba desplazar los planos asociados a las depresiones que representan a los tipos de fallas seleccionadas (baches y grietas profundos).

4 - Cálculo de curvaturas principales (Principal Curvatures Estimation): Una vez realizada la segmentación, se realiza el cálculo de curvaturas promedio para cada uno de los clusters aislados, de manera que se filtren solo aquellos que se ubican en un valor dentro del rango de las fallas, siendo estos valores establecidos a partir del análisis de valores de curvaturas para baches y grietas. PCL ofrece un algoritmo denominado Principal Curvatures Estimation (PCE) para calcular curvaturas principales mínimas y máximas de cada punto, empleando eigenvectores y eigenvalores asociados, en base a un conjunto de puntos y sus normales asociadas. Los eigenvectores (o vectores propios), son un concepto relacionado con el álgebra lineal, y son aquellos vectores no nulos tales que al ser transformados por un operador lineal, no modifican su escala o producen un vector múltiplo de si mismo, manteniendo su dirección; siendo el escalar que los multiplica λ el eigenvector asociado con este valor. Matemáticamente, dada una matriz A n dimensional, se dice que un vector v es un eigenvector y λ es un eigenvalor asociado al eigenvector, si se cumple la siguiente equivalencia:

$$A * v = \lambda * v \quad (6.14)$$

Así, las curvaturas principales se calculan como los eigenvalores para un eigenvector en un punto dado y permiten indicar el grado de depresión en una superficie para un punto establecido. Gráficamente, las curvaturas principales se pueden visualizar como: Si un punto p sobre una superficie dada y un vector unidad normal asociado, este contendrá un plano tangente que corta el punto y el vector normal unidad y, existirán diversos planos que contendrán al vector normal unidad y que cortarán a la superficie de manera distinta, lo que generará diversas curvas con distintos valores por plano. De esta forma, los valores de curvatura seleccionados serán aquellos máximos y mínimos que representen mayor grado de variación de ese conjunto.

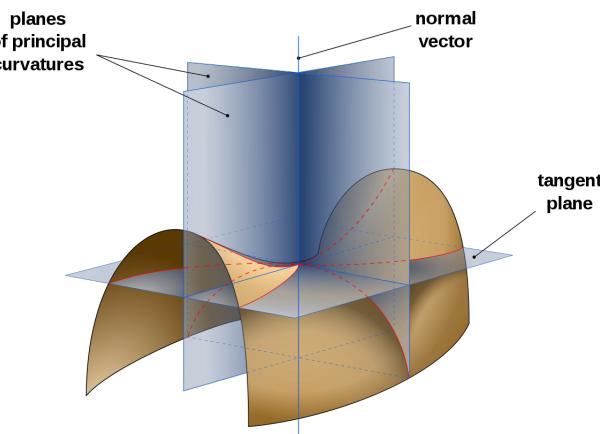


Figura 6.49: Representación gráfica de las curvaturas principales

Por lo tanto, el algoritmo de PCE en PCL para el plano tangente a la normal de un punto dado, aplica PCA sobre las normales de los puntos en un área dada (tomando k-vecinos del punto), siendo primero estas normales trasladadas al plano tangente, y finalmente retorna la curvatura principal (eigenvector del máximo eigenvalor), junto con los valores de curvatura mínimos y máximos (eigenvalores).

5. Metodología para el procesamiento de muestras con ML

Dado que PCL ofrece facilidades para emplear el mecanismo de SVM a través de la librería libsvm (implementada en C y con bindings a Python y compatibilidad con Scikit Learn), se optó por seleccionar este mecanismo en combinación con los descriptores producidos por los algoritmos de ML seleccionados, para las pruebas de clasificación de fallas (detalladas en la sección bitácora de pruebas). La metodología de trabajo para el procesamiento de muestras se dividió en dos fases:

- La fase de preparación del modelo, donde se debió realizar la conversión del descriptor de PCL y las características de la falla a un formato compatible con libsvm, el entrenamiento del modelo con dichos datos y el almacenamiento de éste para su posterior uso en la clasificación. Durante esta etapa, se realiza el entrenamiento de un modelo por cada tipo de descriptor probado.
- La fase de clasificación de muestras, donde se realiza el aislamiento de la muestra empleando el pipeline de cropeado y se emplea el modelo entrenado previamente para un descriptor para clasificar la muestra aislada previamente.

Con respecto a la fase de preparación del modelo, los pasos específicos para generar cada modelo en base a un descriptor consistieron en los siguientes:

1. Aplicar el pipeline de cropeado para cada muestra
2. Computación de descriptor (ESF | FPFH | VFH | GRSD | RIFT)
3. Extracción de features (valores del histograma) del descriptor seleccionado
4. Almacenamiento de las features en formato svmlight en archivo de training
5. Entrenamiento y almacenamiento del modelo entrenado con archivo de training

Luego de aplicar el pipeline de cropeado y computarse los descriptores de las muestras, se procede a realizar la conversión de las muestras a formato svmlight. Para la clasificación de muestras con svmlight, el formato consiste en especificar cada muestra como una combinación de un número que especifica la clase a la que pertenece la misma separado por un espacio en blanco <SPACE> de sus features <FEATURE_N> con sus respectivos valores <VALOR> y, separada de otras muestras por caracteres de nueva línea <NEW_LINE>:

<LABEL> <FEATURE_1>:<VALOR> <FEATURE_2>:<VALOR> ... <FEATURE_N>:<VALOR><NEW_LINE> <LABEL> <FEATURE_1>:<VALOR> <FEATURE_2>:<VALOR> ... <FEATURE_N>:<VALOR><NEW_LINE> ...”

Para el modo de clasificación, la clase a la que la muestra pertenece se especifica como un valor positivo (1) si la muestra pertenece a la clase del tipo de elementos que se busca clasificar o, negativo (-1) si ésta no pertenece a la clase del tipo de elementos que se desean clasificar. Los features se especifican como una sucesión de valores numéricos que representan las características propias de cada muestra, y que varía según el tamaño del histograma del descriptor que se emplee. Con el fin de realizar la conversión, se empleo un script de generación de muestras que por medio de un archivo de configuración (.cfg), genera los descriptores para cada muestra y lo acena en un archivo de testing o training según se haya especificado.

Una vez generados ambos archivos de training y testing, se procede a entrenar el modelo empleando el archivo de training, utilizando una de las utilidades provistas por svm-light (svm-train), que permite generar un modelo de salida para distintos tipos de kernel y distintos tipos de SVM según la tarea para la que se emplee la misma (regresión o clasificación). Debido a que se debe realizar una división de muestras entre clases preestablecidas, se empleó una SVM para clasificación de muestras (SVC) y debido a que el kernel que mejor precisión brindo fue Linear, éste fue empleado para generar el modelo, en combinación con distintos descriptores.

Con respecto a la etapa de clasificación, los pasos a seguir fueron los siguientes:

1. Aplicación del pipeline de cropeado a una muestra individual 
2. Lectura del modelo entrenado desde disco 
3. Computación de las dimensiones de la falla 
4. Generación del descriptor final, combinando el descriptor PCL y las dimensiones de la falla 
5. Conversión del descriptor final a formato svmlight 
6. Clasificación de la muestra (bache o grieta) empleando el descriptor final 
7. Almacenamiento en formato json de las propiedades de la falla 
8. Lectura y muestra de las propiedades obtenidas desde la aplicación web 

Luego de obtener los clusters válidos desde el pipeline de cropeado, se procede generar el descriptor final computando el descriptor seleccionado en PCL y, a calcular las dimensiones (alto-ancho y profundidad para baches y largo-grosor y profundidad para las grietas) en los ejes X,Y y Z por medio de la resta de las coordenadas mínimas y máximas en cada eje, obtenidas de la computación de una OBB que contiene a la falla. De esta forma, el descriptor final para cada cluster se compone del descriptor de PCL sumado a la diferencia entre alto y ancho y, posteriormente se adapta al formato que es utilizado por la SVC.

Una vez obtenida la muestra, se levanta el modelo entrenado desde disco, y se le asigna la muestra para su clasificación, obteniendo el tipo de ésta, el cual, se almacena junto con las dimensiones de la falla según corresponda y el nombre del cluster (generado en base al nombre de la muestra) en formato json. Éste, posteriormente es leído por la aplicación web, que mostrará dicha información en una sección a parte, donde se visualizan las propiedades de la falla.

6. Bitácora de pruebas para clasificación

Como primera medida, se procedió a realizar el cálculo de la cantidad de muestras que se dedicarán para training y testing del total de las muestras que se capturaron, siendo éste de 1000 muestras entre baches y grietas. Se decidió seleccionar un 76,75 % de las muestras para training (766) y el 33 % para testing (234). Una vez hecha la división, se decidió que se aplicaría un Pipeline de Cropeado que consistirá de varios pasos que abarcan desde la limpieza y aislamiento de la muestra hasta la clasificación, con el fin de disregar el tipo de falla del plano en el que ésta se encuentra y obtener sólo features inherentes a la falla.

Con respecto a la computación de features de baches y grietas, se optó por investigar cuales de los descriptores de PCL se enfocaban en capturar las diferencias entre distintos tipos de formas en superficies semejantes a planos, y debido al tamaño promedio de las nubes de puntos capturadas por el sensor, se seleccionaron aquellos que se definían por un histograma cuyas dimensiones no eran de una magnitud que prolongue el tiempo de procesamiento de manera excesiva.

Una vez aisladas todas las muestras de training, se comenzó con las pruebas de clasificación que consistieron en separar los descriptores FPFH del conjunto de training que emplea la SVM, tomando para este conjunto, como muestras positivas los baches y como muestras negativas las grietas, con el fin de intentar clasificar sólo entre baches y grietas. Una vez entrenada, la SVM se probó con diversos conjuntos de entrenamiento: Un conejo, un bache, una grieta y un conjunto de muestras mixto (que consistía de 7 baches y 28 elementos que no son baches). El resultado de esta prueba fue negativo, debido a que la muestra de bache no fue reconocida como tal, la del conejo resultó positiva y la del conjunto de training mixto proporcionó resultados positivos para muestras que no eran baches. Posteriormente, se aplicó la misma prueba para el descriptor VFH y GRSD, obteniéndose resultados positivos para muestras que no eran baches y negativos para baches, logrando un accuracy considerablemente inferior al esperado. Luego, se testeó escalando los valores de las features con el mismo dataset, la misma SVM y no se consiguió un aumento de precisión, para los 3 descriptores que emplean normales (FPFH,GRSD,VFH).

Dado que las diferencias entre los descriptores de los distintos tipos de muestra no eran significativas, se realizó una comparación gráfica de los descriptores pertenecientes al mismo conjunto de muestras, observando que el descriptor GRSD contenía mayor diferencia entre distintos tipos de muestra, por lo que se continuó experimentando solamente con este descriptor y se procedió a cambiar el enfoque, distinguiendo baches de planos y por otro lado, grietas y planos, necesitando clasificadores independientes. Con esta aproximación, la precisión aumentó considerablemente.

Dado la necesidad de utilizar dos clasificadores diferentes por cada clase de muestra, se hizo un análisis de los valores de las curvaturas (por medio del algoritmo de PCL Principal Curvatures Estimation) máximos y mínimos promedio por cada muestra, con el objetivo de encontrar un parámetro que, sumado al descriptor GRSD, permitiera la diferenciación entre ambos tipos de muestra empleando un único clasificador, y se pudo observar que el rango de curvatura promedio de las grietas estaba contenida dentro del rango de los baches, por lo que los baches contenían valores de curvatura mayores en general. Por esta razón, se decidió emplear el valor de curvatura para mejorar el segmentador y aislar sólo aquellas capturas cuya curvatura promedio se aproxime a la de un bache/grieta.

Luego se agregaron las features de curvatura máxima y mínima promedio de cada muestra al descriptor GRSD, y se entrenó una SVM con capacidad para multiclasificación (multi-labels), dividiendo las muestras utilizadas entre 3 diferentes clases: Baches, Grietas y Planos (utilizados solamente para este experimento). Se confeccionó el conjunto de training final con baches con histogramas GRSD similares, grietas y planos cropeados, (empleando como parámetros para un kernel RBF gamma -g 0.0008 y un costo -c 1) obteniendo una precisión del 55 % con un subconjunto de muestras del set de testing, aisladas con el segmentador mejorado, por lo que se observó que muchos de los baches se clasificaron como grietas, distinguiéndose así éstos de los planos, pero no de las grietas. Como la precisión obtenida con GRSD resultó ser muy baja, adicionalmente se probó con el descriptor local FPFH que calcula un histograma por punto, agregando los valores de curvatura y, al probarlo con las muestras de testing anteriores, se logró una precisión del 56,47 %, observando que el descriptor en combinación con la curvatura, no mejoraba satisfactoriamente la precisión.

Debido a esto, se decidió utilizar otro descriptor global conocido como Ensemble Shape of Functions (ESF) en una SVM multiclasificación, alcanzándose una precisión del 54.4444 % empleando el mismo set de testing, pudiendo conseguir que el clasificador distinguiera las grietas y baches de los planos, pero sin diferenciar baches de grietas, clasificando el resto de las muestras como grietas cuando en realidad eran baches.

Otra prueba realizada, consistió en computar y analizar el área y volumen de cada muestra de training, ya que si bien estos valores mostraban una diferencia inferior al feature de curvaturas, no era lo suficientemente ínfima para no lograr diferenciar baches y grietas. Al agregar estas características al descriptor GRSD, con SVM con kernel Linear se obtuvo una precisión del 52.94 % con el set de testing de baches y grietas, sin incluir planos. Además, se incluyeron aquellos atributos que son referentes a las dimensiones de las grietas y baches de training: ancho, alto, profundidad y volumen, y con éstos se realizó una comparación con el fin de obtener valores que permitieran diferenciar entre baches y grietas. Así, se

optó por emplear el descriptor GRSD con la diferencia en valor absoluto de ancho y alto de las fallas, clasificando por este límite a los baches que tienen diferencia $| \text{alto} - \text{ancho} | > 40$ como grietas y, los que tienen menor diferencia como baches. De esta forma, se reclasificaron las muestras según este valor y se realizaron las siguientes pruebas con el subconjunto de testing seleccionado obteniendo como resultado:

- Al agregar los valores de alto, profundidad y ancho, con el descriptor GRSD se obtuvo un accuracy de 79.8 %.
- Al agregar al GRSD la diferencia entre ancho y alto al descriptor GRSD se logró un accuracy de 100 %.
- Agregando volumen y profundidad al descriptor GRSD con la diferencia entre ancho y alto, se redujo el accuracy al 75 %.
- Al agregar al GRSD la diferencia entre ancho y alto y testeando únicamente con el descriptor GRSD, se logró un accuracy de 75 % kernel Linear y 87.5 con kernel RBF (con costo $-c$ 2 y gamma $-g$ 0.00000002).

Ya que al analizar la diferencia entre alto-ancho en el dataset de training de baches y grietas ésta era similar entre el mismo tipo de muestra, por lo que existían muestras (baches y grietas) que poseían una relación similar entre alto-ancho, se realizó una reclasificación de baches y grietas según esta característica. Luego al probar nuevamente la SVM entrenada con el subconjunto de testing incluyendo solamente los valores del descriptor GRSD y la diferencia entre alto-ancho, se consiguió una precisión del 87.5 % con kernel RBF y un 100 % con kernel Linear.

Al observar que la precisión incrementó reclasificando el dataset de training, se aplicó el mismo procedimiento para el dataset de testing completo y debido a que el ancho y alto calculados se basan en valores máximos y mínimos que son brindados por el mecanismo Oriented Bounding Box de PCL en los ejes X-Y, el cual se ajusta y se orienta al tamaño de la muestra, se eliminaron aquellas muestras que contenían outliers que introducían ruido en el cálculo de esta diferencia, filtrando con estos parámetros de un total de 1000 muestras, 806 muestras (753 para training y 53 para testing). Al analizar las estadísticas de dimensiones del dataset de fallas de training, se seleccionó un límite de diferencia entre alto y ancho para dividirlas según el tipo (grieta o bache) de 0.49, ya que las grietas contenían una longitud considerablemente mayor al grosor, situación que no ocurría en baches. Al ejecutar nuevamente las pruebas con dataset de training y testing divididos por este límite, se obtuvo 89 % de accuracy con kernel Linear y 71 % con kernel RBF (con gamma 0.0000002 y costo C 1500) empleando un cross validation de 5 iteraciones con GRSD. Nuevamente se procedió a experimentar con la diferencia alto-ancho, cambiando únicamente el descriptor con ESF y FPFH, obteniendo para los mismos parámetros y la misma cantidad de iteraciones los siguientes resultados:

- Con FPFH 63 % para un kernel Linear y 60 % para un kernel RBF.
- Con ESF 98 % para un kernel Linear y 54 % para un kernel RBF.

Finalmente, se realizó una comparación de las métricas de clasificación respecto de los distintos descriptores para la división original de muestras (53 en total), con el fin de contrastar la efectividad de clasificación de éstos y comprobar la superioridad de ESF respecto al resto. Para ello, se calcularon los valores de F1-Score y Recall para ambas clases y la matriz de confusión para exponer la cantidad de elementos efectivamente asignados a cada clase. Los valores de F1-Score y Recall para la partición del dataset inicial, con los kernels linear y RBF, se puede observar a continuación:

Tipo de muestra	Kernel Linear			Kernel RBF		
	Precision	Recall	F1-Score	Precision	Recall	F1-Score
Baches	1.0	1.0	1.0	0.0	0.0	0.0
Grietas	1.0	1.0	1.0	0.17	1.0	0.29
avg/total	1.0	1.0	1.0	0.03	0.17	0.05

Métricas para descriptor ESF con Kernel Linear-RBF

Tipo de muestra	Kernel Linear			Kernel RBF		
	Precision	Recall	F1-Score	Precision	Recall	F1-Score
Baches	0.83	1	0.91	0.00	0.00	0.00
Grietas	0.23	0.78	0.36	0.17	1.00	0.29
avg/total	0.80	0.53	0.58	0.03	0.17	0.05

Métricas para descriptor GRSD con Kernel Linear-RBF

Tipo de muestra	Kernel Linear			Kernel RBF		
	Precision	Recall	F1-Score	Precision	Recall	F1-Score
Baches	0.91	0.48	0.63	1.00	0.09	0.17
Grietas	0.23	0.78	0.36	0.18	1.00	0.31
avg/total	0.80	0.53	0.58	0.86	0.25	0.19

Métricas para descriptor FPFH con Kernel Linear-RBF

La matriz de confusión para cada uno de los descriptores empleados, con la partición de datos inicial, fue la siguiente:

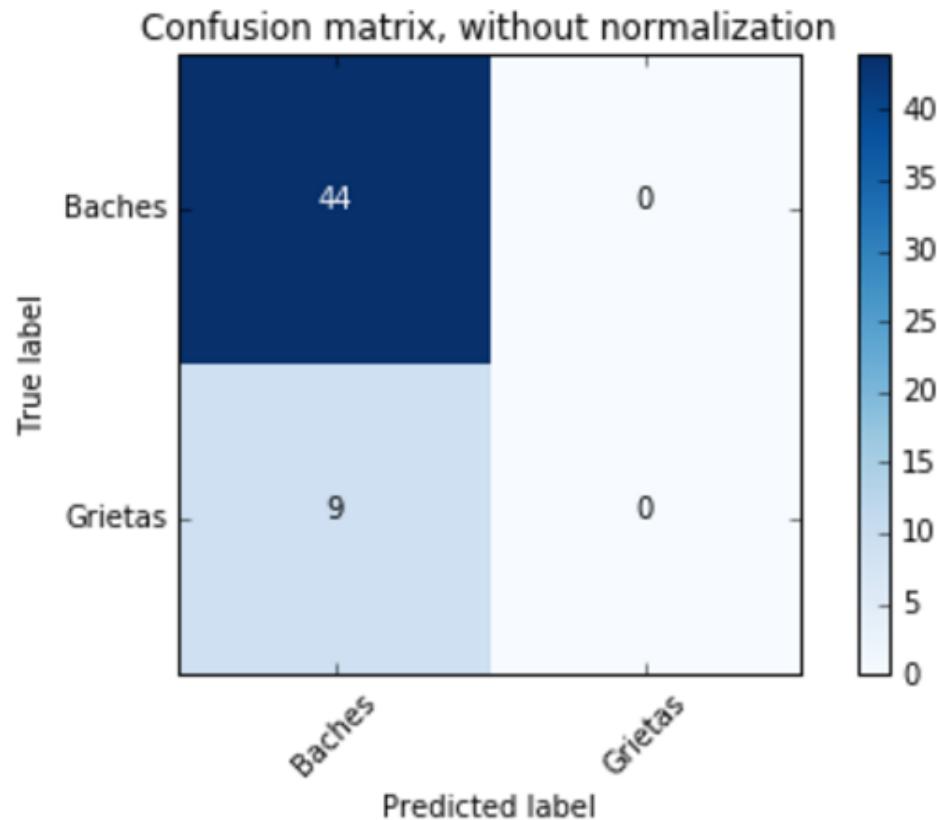


Figura 6.50: Matriz de confusión de SVM con descriptor GRSD

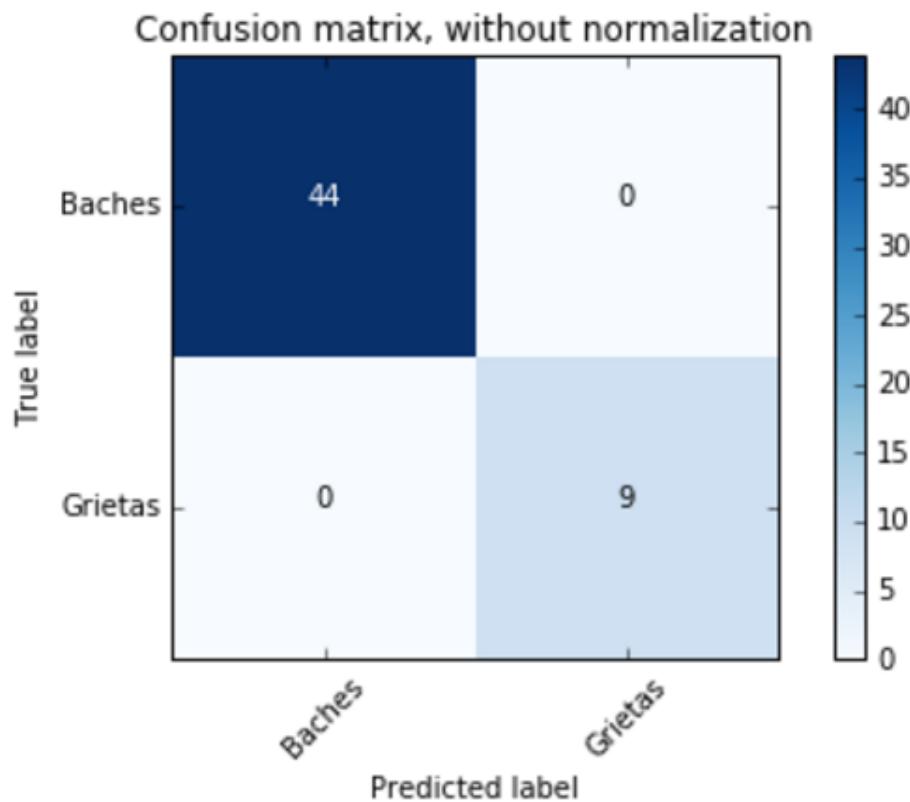


Figura 6.51: Matriz de confusión de SVM con descriptor ESF

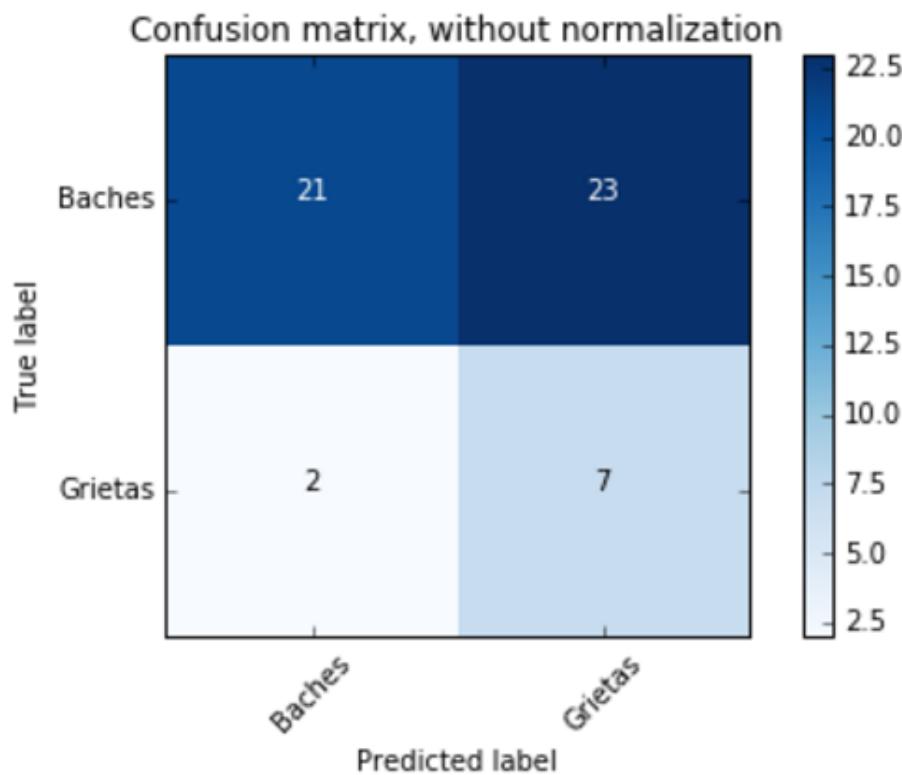


Figura 6.52: Matriz de confusión de SVM con descriptor FPFH

Finalmente, se realizó una comparación de la precisión promedio del k-folding de cada uno de los métodos con la precisión brindada por un clasificador Dummy, comprobando realmente que la eficiencia de clasificación del clasificador (con kernel linear) sobrepasa la de un clasificador aleatorio:

Tipo de descriptor	ESF		GRSD		FPFH	
Tipo clasificador	ESF	DummyESF	GRSD	DummyGRSD	FPFH	DummyFPFH
Precision	0.98	0.45	0.89	0.516	0.63	0.494

Bibliografía

- [1] Fundamentals of image processing. URL: <http://www.cs.dartmouth.edu/farid/downloads/tutorials/fip.pdf>.
- [2] Similarity measure. URL: https://en.wikipedia.org/wiki/Similarity_measure.
- [3] Spectral clustering. URL: https://en.wikipedia.org/wiki/Spectral_clustering.
- [4] -. -. URL: http://www.lcc.uma.es/~munozp/documentos/procesamiento_de_imagenes/temas/pi_cap6.pdf.
- [5] -. Color image processing slides. URL: <https://www.slideshare.net/lineking/10-color-image-processing-dip>.
- [6] _____. _____. URL: <http://civilgeeks.com/2012/03/27/juntas-en-el-concreto/>.
- [7] _____. _____. URL: https://www.ieca.es/Uploads/docs/Diseño_y_ejecución_de_juntas.pdf.
- [8] Jarret Webb, James Ashley. *Beginning Kinect programming with the Microsoft Kinect SDK*. Apress, -.
- [9] a,b , c, d, e,. Road condition monitoring using on-board three-axis acelerometer. 2011.
- [10] Jakob Eriksson ,Lewis Girod ,BretHull ,Ryan Newton ,Samuel Madden ,Hari Balakrishnan. The pothole patrol: using a mobile sensor network for road surface monitoring. *MobiSys*, 2008.
- [11] Radu Bodgan Rusu, Zoltan Csaba Marton, Nico Blodow, Michael Beetz. Learning informative point classes for the acquisition of object model maps. URL:.
- [12] Zoltan-Csaba Marton , Dejan Pangercic, Nico Blodow, Jonathan Kleinehellefort, Michael Beetz. General 3d modelling of novel objects from a single view. URL: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.391.7398&rep=rep1&type=pdf>.
- [13] Zoltan-Csaba Marton , Dejan Pangercic, Nico Blodow, Michael Beetz. Combined 2d-3d categorization and classification for multimodal perception systems. URL: https://ias.cs.tum.edu/_media/spezial/bib/marton11ijrr.pdf.
- [14] Depth Biomechanics. How the kinect works. URL: <http://www.depthbiomechanics.co.uk/?p=100>.
- [15] Bishop. *Pattern recognition and Machine Learning*. Springer, 2006.
- [16] AL Bovik. *Handbook of Image and Video Processing*. Academic Press, 2000.
- [17] AL Bovik. *The essential guide to Image Processing*. Academic Press, 2009.
- [18] Christian Korch , Ioannis Brikalis. Improving pothole recognition through vision tracking for autmated pavement assessment. *Proceedings of the 2011 EG-ICE International Workshop on Intelligent Computing in Engineering*, 2011.
- [19] Christian Korch , Ioannis Brikalis. Pothole detection in asphalt pavement images. *Advanced Engineering in Informatics*, 2011.
- [20] Harvey Rhody, Chester F. Carlson. Geometric image transformations. URL: <https://www.math.hkbu.edu.hk/~zeng/Teaching/math3615/ls.pdf>.

- [21] Ian Goodfellow, Yoshua Bengio, Aaron Courville. *Deep Learning*. MIT, 2016.
- [22] Ahmad Davudinasab. Kinect sensor. URL: https://www.researchgate.net/publication/268517966_Kinect_Sensor.
- [23] Pointclouds docs. Documentación de pcl. URL: http://docs.pointclouds.org/trunk/classpcl_1_1_principal_curvatures_estimation.html#details.
- [24] Mostafa Abdel, Bary EBRAHIM. *3D Laser Scanner's Techniques Overview*. International Journal of Science and Research (IJSR), 2015.
- [25] Albert Einstein. Zur Elektrodynamik bewegter Körper. (German) [On the electrodynamics of moving bodies]. *Annalen der Physik*, 322(10):891–921, 1905. doi:<http://dx.doi.org/10.1002/andp.19053221004>.
- [26] Andreas Eklund. *3D scanning With a Mobile Phone and Other Methods*. Arcada, 2016.
- [27] FayerWayer. Especificaciones técnicas de kinect. URL: <https://www.fayerwayer.com/2010/06/especificaciones-tecnicas-de-kinect/>.
- [28] Henrik Brink, Joseph W. Richards, Mark Fetherolf. *Real-World Machine Learning*. Mannnig Publications, 2017.
- [29] Aleksey Golovinsky, Thomas Funkhouser. Min-cut based segmentation of point clouds. URL: http://gfx.cs.princeton.edu/pubs/Golovinskiy_2009_MBS/paper_small.pdf.
- [30] Juan-Miguel Garcia. Valores y vectores propios de una matriz. URL: http://www.vc.ehu.es/campus/centros/farmacia/deptos-f/depme/apuntes/gracia/Curso_Actual/bolonia/matematicas/capitulo_1/espectral/ValoresVectoresPropiosPapel.pdf.
- [31] Research Gate. Voxel grid spanning a value in a 3d space bounded by (x min,x max)(y min, y max). URL: https://www.researchgate.net/figure/259053167_fig3_Figure-3-Voxel-grid-spanning-a-volume-in-a-3D-space-bounded-by-x-min-x-max-y-min.
- [32] PCL Github. Overview and comparison of features (algoritmos para obtencion de descriptores). URL: <https://github.com/PointCloudLibrary/pcl/wiki/Overview-and-Comparison-of-Features>.
- [33] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The \LaTeX Companion*. Addison-Wesley, Reading, Massachusetts, 1993.
- [34] Yao Wang, Zhu Liu, Onur Gulerzyuz. Image processing. geometric transformations,warping,registration,morphing. URL: http://eeweb.poly.edu/~yao/EL5123/lecture12_ImageWarping.pdf.
- [35] Tomoyuki Yamaguchi ,Shingo Nakamura ,Ryo Saegusa , Shuji Hashimoto. Image-based crack detection for real concrete surfaces. *Wiley InterScience*, 2008.
- [36] Eric Wahl, Ulrich Hillenbrand, Gerd Hirzinger. Surfel-pair-relation histograms: a statistical 3d-shape representation for rapid classification. URL:
- [37] Martin Hoffman. Support vector machines - kernels and the kernel trick. URL: http://www.cogsys.wiai.uni-bamberg.de/teaching/ss06/hs_svm/slides/SVM_Seminarbericht_Hofmann.pdf.
- [38] Martin Hofmann. Support vector machines - kernels and the kernel tick. URL: http://www.cogsys.wiai.uni-bamberg.de/teaching/ss06/hs_svm/slides/SVM_Seminarbericht_Hofmann.pdf.
- [39] HowStuffWorks. How microsoft kinect works. URL: <https://electronics.howstuffworks.com/microsoft-kinect2.htm>.
- [40] Emir Buza , Samir Omanovic , Alvin Huseinovic. Pothole detection with image processing and spectral cluterling. *Recent Advances in Computer Science and Networking*, 2013.
- [41] Abhijit Jana. *Kinect for Windows SDK Programming Guide*. PACKT Publishing, 2012.
- [42] Seing-Ku Ryu , Taehyong Kim , Young-Ro Kim. Image-based pothole detection system for its service and road management system. *Matematical problems in Engineering*, 2015.
- [43] Donald Knuth. Knuth: computers and typesetting. URL: <http://www-cs-faculty.stanford.edu/~uno/abcde.html>.
- [44] S Nieber , M Brooysen , R Kroon. Detection potholes using simple processing techniques and real-world footage. *Proceedings of the 34th Southern African Transport Conference*, 2015.

- LaserFocusWorld. Photonic frontiers: gesture recognition: lasers bring gesture recognition to the home. URL: <https://www.laserfocusworld.com/articles/2011/01/lasers-bring-gesture-recognition-to-the-home.html>.
- [46] Scikit Learn. Adjusted mutual information. URL: https://en.wikipedia.org/wiki/Adjusted_mutual_information.
- [47] Scikit Learn. Confusion matrix. URL: http://scikit-learn.org/stable/modules/model_evaluation.html#confusion-matrix.
- [48] Scikit Learn. Homogeneity, completeness and v-measure. URL: <http://scikit-learn.org/stable/modules/clustering.html#homogeneity-completeness>.
- [49] Scikit Learn. Model evaluation: quantifying the quality of predictions. URL: http://scikit-learn.org/stable/modules/model_evaluation.html.
- [50] Scikit Learn. Multiclass classification. URL: https://en.wikipedia.org/wiki/Multiclass_classification.
- [51] Scikit Learn. Mutual information based scores. URL: <http://scikit-learn.org/stable/modules/clustering.html#mutual-info-score>.
- [52] Scikit Learn. Mutual info score. URL: http://scikit-learn.org/stable/modules/generated/sklearn.metrics.mutual_info_score.html#sklearn.metrics.mutual_info_score.
- [53] Scikit Learn. Pca. URL: <http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html#sklearn.decomposition.PCA>.
- [54] Scikit Learn. Precision, recall and f-measures. URL: http://scikit-learn.org/stable/modules/model_evaluation.html#precision-recall-f-measure-metrics.
- [55] Scikit Learn. R² score, the coefficient of determination. URL: http://scikit-learn.org/stable/modules/model_evaluation.html#r2-score.
- [56] Scikit Learn. Support vector machines. URL: <http://scikit-learn.org/stable/modules/svm.html>.
- [57] Tongobo Chen, Hans-Peter Seidel, Hendrik P. A. Lensch. Modulated phase-shifting for 3d scanning. URL: <http://www.cs.virginia.edu/~mjh7v/bib/Chen08.pdf>.
- [58] Deon Joubert, Ayanda Tyatyantsi , Jeffry Mphahlele , Vivian Manchidi. Pothole tagging system. *4th Robotics and Mechatronics Conference of South Africa (RobMech 2011), CSIR International Conference Centre, Pretoria, 2011.* URL: <https://researchspace.csir.co.za/dspace/handle/10204/5384>.
- [59] Machine Learning Mastery. Classification and regression trees for machine learning. URL: <https://machinelearningmastery.com/classification-and-regression-trees-for-machine-learning/>.
- [60] Machine Learning Mastery. Decision tree - classification. URL: http://www.saedsayad.com/decision_tree.htm.
- [61] Machine Learning Mastery. Overfitting and underfitting with machine learning algorithms. URL: <https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/>.
- [62] Machine Learning Mastery. Visualize machine learning data in python with pandas. URL: <https://machinelearningmastery.com/visualize-machine-learning-data-python-pandas/>.
- [63] Matteo Munaro , Stefano Tonello, Edmond Wai Yan So, Emanuele Menegatti. Efficient completeness inspection using real-time 3d color reconstruction with a dual-laser triangulation system. URL: https://www.researchgate.net/publication/283108894_Efficient_Completeness_Inspection_Using_Real-Time_3D_Colour_Reconstruction_with_a_Dual-Laser_Triangulation_System.
- [64] Microsoft. Accelerometer. URL: <https://msdn.microsoft.com/en-us/library/jj663790.aspx>.
- [65] Microsoft. Audio stream. URL: <https://msdn.microsoft.com/en-us/library/jj131026.aspx>.
- [66] Microsoft. Color stream. URL: <https://msdn.microsoft.com/en-us/library/jj131027.aspx>.
- [67] Microsoft. Colorimagestream class. URL: <https://msdn.microsoft.com/en-us/library/microsoft.kinect.colorimagestream.aspx>.
- [68] Microsoft. Coordinate spaces. URL: <https://msdn.microsoft.com/en-us/library/hh973078.aspx>.

- [69] Microsoft. Depth stream. URL: <https://msdn.microsoft.com/en-us/library/jj131028.aspx>.
- [70] Microsoft. Face tracking. URL: <https://msdn.microsoft.com/en-us/library/jj130970.aspx>.
- [71] Microsoft. Getting started. URL: <https://msdn.microsoft.com/en-us/library/hh855354.aspx>.
- [72] Microsoft. Getting the next frame of data by polling or using events. URL: <https://msdn.microsoft.com/en-us/library/hh973076.aspx>.
- [73] Microsoft. Kinect fusion. URL: <https://msdn.microsoft.com/en-us/library/dn188670.aspx>.
- [74] Microsoft. Kinect for windows architecture. URL: <https://msdn.microsoft.com/en-us/library/jj131023.aspx>.
- [75] Microsoft. Kinect for windows programming guide. URL: <https://msdn.microsoft.com/en-us/library/hh855348.aspx>.
- [76] Microsoft. Kinect for windows sensor components and specifications. URL: <https://msdn.microsoft.com/en-us/library/jj131033.aspx>.
- [77] Microsoft. Kinectinteraction. URL: <https://msdn.microsoft.com/en-us/library/dn188671.aspx>.
- [78] Microsoft. Natural user interface for kinect for windows. URL: <https://msdn.microsoft.com/en-us/library/hh855352.aspx>.
- [79] Microsoft. Running a kinect-enabled application on a non-developer machine. URL: <https://msdn.microsoft.com/en-us/library/hh855357.aspx>.
- [80] Microsoft. What's new (sdk 1.8). URL: <https://msdn.microsoft.com/en-us/library/jj663803.aspx>.
- [81] Microsoft. *Human Interface Guidelines v1.8*. Microsoft, 2013.
- [82] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.
- [83] Senthil Mathavan, Mujib Rahman, K. Kamal, S. Usman, I. Moazzam. Motrology and visualization of potholes using the microsoft kinect sensor. *16th International IEEE Annual Conference on Intelligent Transportation Systems (ITSC 2013)*, 2013. URL: https://www.researchgate.net/publication/260293197_Metrology_and_Visualization_of_Potholes_using_the_Microsoft_Kinect_Sensor.
- [84] Andrew Nealen. An as-short-as-possible introduction to the least squares, weighted least squares and moving least squares methods for scattered data approximation and interpolation. URL: <http://www.nealen.de/projects/mls/asapmls.pdf>.
- [85] Michael Nelsen. Using neural nets to recognize handwritten digits. URL: <http://neuralnetworksanddeeplearning.com/chap1.html>.
- [86] The University of ADELAIE. Maths track. topic 3: vectors and applications. URL: https://www.adelaide.edu.au/mathslearning/bridging/resources/MT3VectorsBook_Feb2013.pdf.
- [87] Alasdair McAndrew. Victoria University of Technology. An introduction to digital image processing with matlab. URL: <https://www.math.hkbu.edu.hk/~zeng/Teaching/math3615/ls.pdf>.
- [88] OpenKinect. Getting started. URL: https://openkinect.org/wiki/Getting_Started.
- [89] OpenKinect. Main page. URL: https://openkinect.org/wiki/Main_Page.
- [90] OpenKinect. Pointcloud library. URL: https://en.wikipedia.org/wiki/Point_Cloud_Library.
- [91] OpenKinect. Roadmap. URL: <https://openkinect.org/wiki/Roadmap>.
- [92] OpenKinect. What's pcl? URL: <http://pointclouds.org/about/>.
- [93] OpenNI. About openni. URL: <http://openni.ru/about/index.html>.
- [94] OpenNI. Openni programmer's guide. URL: <http://openni.ru/openni-programmers-guide/index.html>.
- [95] J. R. Parker. *Algorithms for Image Processing and Computer Vision*. Wiley Publishing , Inc., 2011.
- [96] Jeff Kramer, Nicolas Burrus, Florian Echtler, Daniel Herrera C. , Matt Parker. *Hacking the Kinect*. Apress, 2012.

- [97] PCL. Downsampling a pointcloud using voxelgrid filter. URL: http://pointclouds.org/documentation/tutorials/voxel_grid.php.
- [98] PCL. Estimating surface normals in a pointcloud. URL: http://pointclouds.org/documentation/tutorials/normal_estimation.php.
- [99] PCL. Estimating vfh signatures for a set of points. URL: http://pointclouds.org/documentation/tutorials/vfh_estimation.php#vfh-estimation.
- [100] PCL. Fast point feature histograms (fpfh) descriptors. URL: http://pointclouds.org/documentation/tutorials/fpfh_estimation.php#fpfh-estimation.
- [101] PCL. Filtering a pointcloud using a passthrough filter. URL: <http://pointclouds.org/documentation/tutorials/passthrough.php>.
- [102] PCL. Getting started/basic structures. URL: http://pointclouds.org/documentation/tutorials/basic_structures.php.
- [103] PCL. How 3d features work in pcl. URL: http://pointclouds.org/documentation/tutorials/how_features_work.php#id2.
- [104] PCL. How to use random sample consensus model. URL: http://pointclouds.org/documentation/tutorials/random_sample_consensus.php#random-sample-consensus.
- [105] PCL. How to use a kdtree to search. URL: http://pointclouds.org/documentation/tutorials/kdtree_search.php.
- [106] PCL. Interest point detection. URL: https://en.wikipedia.org/wiki/Interest_point_detection.
- [107] PCL. Keypoints and features. URL: http://www.pointclouds.org/assets/uploads/cglibs13_features.pdf.
- [108] PCL. Min-cut based segmentation. URL: http://pointclouds.org/documentation/tutorials/min_cut_segmentation.php.
- [109] PCL. Model fitting(ransac). URL: [http://robotica.unileon.es/index.php/PCL/OpenNI_tutorial_3:_Cloud_processing_\(advanced\)#Model_fitting_.28RANSAC.29](http://robotica.unileon.es/index.php/PCL/OpenNI_tutorial_3:_Cloud_processing_(advanced)#Model_fitting_.28RANSAC.29).
- [110] PCL. Module keypoints. URL: http://docs.pointclouds.org/trunk/group__keypoints.html.
- [111] PCL. Moment of inertia estimation. URL: http://pointclouds.org/documentation/tutorials/moment_of_inertia.php.
- [112] PCL. Overview and comparison of features. URL: <https://github.com/PointCloudLibrary/pcl/wiki/Overview-and-Comparison-of-Features>.
- [113] PCL. Pcl goes mobile with ves and kiwi. URL: <http://https.www.pointclouds.org/news/2012/05/29/pcl-goes-mobile-with-ves-and-kiwi/>.
- [114] PCL. Pcl segmentation. URL: <http://www.pointclouds.org/assets/iros2011/segmentation.pdf>.
- [115] PCL. Plane model segmentation. URL: http://pointclouds.org/documentation/tutorials/planar_segmentation.php.
- [116] PCL. Point feature histograms (pfh) descriptors. URL: http://pointclouds.org/documentation/tutorials/pfh_estimation.php#pfh-estimation.
- [117] PCL. Principal curvatures estimation. URL: http://docs.pointclouds.org/trunk/classpcl_1_1_principal_curvatures_estimation.html#aa6051f3fa8fec96516c6236fd3f3b6bc.
- [118] PCL. Removing outliers using a conditional or radiusoutlier removal. URL: http://pointclouds.org/documentation/tutorials/remove_outliers.php.
- [119] PCL. Removing outliers using a statisticaloutlierremoval filter. URL: http://pointclouds.org/documentation/tutorials/statistical_outlier.php.
- [120] PCL. Spatial partitioning and search operations with octrees. URL: <http://pointclouds.org/documentation/tutorials/octree.php>.

- [121] PCL. The pcd (point cloud data) file format. URL: http://pointclouds.org/documentation/tutorials/pcd_file_format.php.
- [122] ROS PCL. URL: <http://wiki.ros.org/pcl>.
- [123] PCL Pointcloud. Documentacion de pcl. URL: <http://www.pointclouds.org/documentation/>.
- [124] PCL Pointcloud. Tutoriales de pcl. URL: <http://www.pointclouds.org/documentation/tutorials/>.
- [125] Radu B. Rusu. Principal curvatures estimation. URL: <http://www.pcl-users.org/Principal-curvatures-estimation-td2695839.html>.
- [126] Florida Laser Scanning. How does laser scanning work? URL: <http://floridalaserscanning.com/3d-laser-scanning/how-does-laser-scanning-work/>.
- [127] Jeff Schneider. Cross validation. URL: <https://www.cs.cmu.edu/~schneide/tut5/node42.html>.
- [128] Scikit. Underfitting vs overfitting. URL: http://scikit-learn.org/stable/auto_examples/model_selection/plot_underfitting_overfitting.html.
- [129] Nibcode Solutions. El algebra lineal y el procesamiento digital de imágenes. URL: <http://www.nibcode.com/es/blog/14/linear-algebra-and-digital-image-processing-part-III-affine-transformations>.
- [130] César Souza. Kernel functions for machine learning applications. URL: <http://crsouza.com/2010/03/17/kernel-functions-for-machine-learning-applications/#linear>.
- [131] César Souza. Radial basis function kernel. URL: https://en.wikipedia.org/wiki/Radial_basis_function_kernel.
- [132] Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani. *Introduction to Statistical Learning with applications in R*. Springer, 2013.
- [133] Federico Tombari. How does a good feature look like? URL: http://www.pointclouds.org/assets/icra2013/pcl_features_icra13.pdf.
- [134] Robotica Unileon. Kd-tree. URL: [http://robotica.unileon.es/index.php/PCL/OpenNI_tutorial_2:_Cloud_processing_\(basic\)#k-d_tree](http://robotica.unileon.es/index.php/PCL/OpenNI_tutorial_2:_Cloud_processing_(basic)#k-d_tree).
- [135] Robotica Unileon. Octree. URL: [http://robotica.unileon.es/index.php/PCL/OpenNI_tutorial_2:_Cloud_processing_\(basic\)#Octree](http://robotica.unileon.es/index.php/PCL/OpenNI_tutorial_2:_Cloud_processing_(basic)#Octree).
- [136] Robotica Unileon. Pcl/openni tutorial 0: the very basics. URL: http://robotica.unileon.es/index.php/PCL/OpenNI_tutorial_0:_The_very_basics.
- [137] Robotica Unileon. Pcl/openni tutorial 1: installing and testing. URL: http://robotica.unileon.es/index.php/PCL/OpenNI_tutorial_1:_Installing_and_testing.
- [138] Robotica Unileon. Pcl/openni tutorial 2: cloud processing (basic). URL: [http://robotica.unileon.es/index.php/PCL/OpenNI_tutorial_2:_Cloud_processing_\(basic\)](http://robotica.unileon.es/index.php/PCL/OpenNI_tutorial_2:_Cloud_processing_(basic)).
- [139] Robotica Unileon. Pcl/openni tutorial 2: cloud processing (basic). URL: [http://robotica.unileon.es/index.php/PCL/OpenNI_tutorial_2:_Cloud_processing_\(basic\)](http://robotica.unileon.es/index.php/PCL/OpenNI_tutorial_2:_Cloud_processing_(basic)).
- [140] Robotica Unileon. Pcl/openni tutorial 3: cloud processing (advanced). URL: [http://robotica.unileon.es/index.php/PCL/OpenNI_tutorial_3:_Cloud_processing_\(advanced\)](http://robotica.unileon.es/index.php/PCL/OpenNI_tutorial_3:_Cloud_processing_(advanced)).
- [141] Robotica Unileon. Pcl/openni tutorial 3: cloud processing (advanced). URL: [http://robotica.unileon.es/index.php/PCL/OpenNI_tutorial_3:_Cloud_processing_\(advanced\)](http://robotica.unileon.es/index.php/PCL/OpenNI_tutorial_3:_Cloud_processing_(advanced)).
- [142] Robotica Unileon. Pcl/openni tutorial 4: 3d object recognition (descriptros). URL: [http://robotica.unileon.es/index.php/PCL/OpenNI_tutorial_4:_3D_object_recognition_\(descriptors\)](http://robotica.unileon.es/index.php/PCL/OpenNI_tutorial_4:_3D_object_recognition_(descriptors)).
- [143] Robotica Unileon. Pcl/openni tutorial 4: 3d object recognition (descriptors). URL: [http://robotica.unileon.es/index.php/PCL/OpenNI_tutorial_4:_3D_object_recognition_\(descriptors\)](http://robotica.unileon.es/index.php/PCL/OpenNI_tutorial_4:_3D_object_recognition_(descriptors)).
- [144] Robotica Unileon. Pcl/openni tutorial 4: 3d object recognition(descriptors). URL: [http://robotica.unileon.es/index.php/PCL/OpenNI_tutorial_4:_3D_object_recognition_\(descriptors\)](http://robotica.unileon.es/index.php/PCL/OpenNI_tutorial_4:_3D_object_recognition_(descriptors)).
- [145] Robotica Unileon. Pcl/openni tutorial 5: 3d object recognition (pipeline). URL: [http://robotica.unileon.es/index.php/PCL/OpenNI_tutorial_5:_3D_object_recognition_\(pipeline\)](http://robotica.unileon.es/index.php/PCL/OpenNI_tutorial_5:_3D_object_recognition_(pipeline)).

- [146] Robotica Unileon. Phd-3d-object tracking. URL: <http://robotica.unileon.es/index.php/PhD-3D-Object-Tracking>.
- [147] Robotica Unileons. Esf. URL: [http://robotica.unileon.es/index.php/PCL/OpenNI_tutorial_4:_3D_object_recognition_\(descriptors\)#ESF](http://robotica.unileon.es/index.php/PCL/OpenNI_tutorial_4:_3D_object_recognition_(descriptors)#ESF).
- [148] Cerón ,Bermudez V.G. Evaluación y comparación de metodologías vizir y pci sobre el tramo de via en pavimento flexible y rígido de la vía: museo quimbaya-crq armenia quindío. 2006.
- [149] Ian T. Young , Jan J. Gerbrans , Lucas J. van Vliet. Fundamentals of image processing. URL: http://www.tnw.tudelft.nl/fileadmin/Faculteit/TNW/Over_de_faculteit/Afdelingen/Imaging_Science_and_Technology/Research/Research_Groups/Quantitative_Imaging/Education/doc/FIP2_3.pdf.
- [150] E. Boyer, A. M. Bronstein, M. M. Bronstein, B. Bustos, T. Darom, R. Horaud, I. Hotz, Y. Keller, J. Keustermans, A. Kovnatsky, R. Litman, J. Reininghaus, I. Sipirian, D. Smeets, P. Suetens, D. Vandermeulen. Shrec 2011:robust feature detection and description benchmark. URL: <https://arxiv.org/pdf/1102.4258.pdf>.
- [151] T.PRATHIBA , THAMARAISELVI.M , MOHANASUNDARI.M , VEERELAKSHMI.R. Pothole detection in road using image processing. *Interational Journal of Managment,Information*, 2015.
- [152] Analitics Vidhya. Understanding support vector machine algorithm from examples (along with code). URL: <https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/>.
- [153] Axel Smola, S.V.N Vishwanathan. *Introduction to Machine Learning*. Cambridge, 2010.
- [154] Wikipedia. 3d scanner. URL: https://en.wikipedia.org/wiki/3D_scanner.
- [155] Wikipedia. Análisis de componentes principales. URL: https://es.wikipedia.org/wiki/Análisis_de_componentes_principales.
- [156] Wikipedia. Análisis de componentes principales(pca). URL: https://es.wikipedia.org/wiki/Análisis_de_componentes_principales.
- [157] Wikipedia. Artificial neural network. URL: https://en.wikipedia.org/wiki/Artificial_neural_network.
- [158] Wikipedia. Biological neural network. URL: https://en.wikipedia.org/wiki/Biological_neural_network.
- [159] Wikipedia. Coeficient of determination. URL: https://en.wikipedia.org/wiki/Coefficient_of_determination.
- [160] Wikipedia. Computer vision. URL: https://en.wikipedia.org/wiki/Computer_vision.
- [161] Wikipedia. Correlation and dependence. URL: https://en.wikipedia.org/wiki/Correlation_and_dependence#Correlation_matrices.
- [162] Wikipedia. Covariance. URL: <https://en.wikipedia.org/wiki/Covariance>.
- [163] Wikipedia. Covariance matrix. URL: https://en.wikipedia.org/wiki/Covariance_matrix.
- [164] Wikipedia. Eigenvalue and eigenvector. URL: https://en.wikipedia.org/wiki/Eigenvalues_and_eigenvectors.
- [165] Wikipedia. Errors and residuals. URL: https://en.wikipedia.org/wiki/Errors_and_residuals.
- [166] Wikipedia. Field of view. URL: https://en.wikipedia.org/wiki/Field_of_view.
- [167] Wikipedia. Kd-tree. URL: https://en.wikipedia.org/wiki/K-d_tree.
- [168] Wikipedia. Kernel method. URL: https://en.wikipedia.org/wiki/Kernel_method.
- [169] Wikipedia. Lidar. URL: <https://en.wikipedia.org/wiki/Lidar>.
- [170] Wikipedia. Mean absolute error. URL: https://en.wikipedia.org/wiki/Mean_absolute_error.
- [171] Wikipedia. Mean squared error. URL: https://en.wikipedia.org/wiki/Mean_squared_error.
- [172] Wikipedia. Minimum bounding box. URL: https://en.wikipedia.org/wiki/Minimum_bounding_box.

- [173] Wikipedia. Minimum cut. URL: https://en.wikipedia.org/wiki/Minimum_cut.
- [174] Wikipedia. Moving least squares. URL: https://en.wikipedia.org/wiki/Moving_least_squares.
- [175] Wikipedia. Multiclass classification. URL: https://en.wikipedia.org/wiki/Multiclass_classification.
- [176] Wikipedia. Normal (geometry). URL: [https://en.wikipedia.org/wiki/Normal_\(geometry\)](https://en.wikipedia.org/wiki/Normal_(geometry)).
- [177] Wikipedia. OpenNI. URL: <https://es.wikipedia.org/wiki/OpenNI>.
- [178] Wikipedia. Otsu's method. URL: https://en.wikipedia.org/wiki/Otsu's_method.
- [179] Wikipedia. Ply (file format). URL: [https://en.wikipedia.org/wiki/PLY_\(file_format\)](https://en.wikipedia.org/wiki/PLY_(file_format)).
- [180] Wikipedia. Phase(waves). URL: [https://en.wikipedia.org/wiki/Phase_\(waves\)](https://en.wikipedia.org/wiki/Phase_(waves)).
- [181] Wikipedia. Point cloud. URL: https://en.wikipedia.org/wiki/Point_cloud.
- [182] Wikipedia. Polynomial kernel. URL: https://en.wikipedia.org/wiki/Polynomial_kernel.
- [183] Wikipedia. Precision and recall. URL: https://en.wikipedia.org/wiki/Precision_and_recall#Precision.
- [184] Wikipedia. Principal curvature. URL: https://en.wikipedia.org/wiki/Principal_curvature.
- [185] Wikipedia. Radial basis function. URL: https://en.wikipedia.org/wiki/Radial_basis_function_kernel.
- [186] Wikipedia. Random sample consensus. URL: https://en.wikipedia.org/wiki/Random_sample_consensus.
- [187] Wikipedia. Range imaging. URL: https://en.wikipedia.org/wiki/Range_imaging.
- [188] Wikipedia. Root mean square deviation. URL: https://en.wikipedia.org/wiki/Root-mean-square_deviation.
- [189] Wikipedia. Stl (file format). URL: [https://en.wikipedia.org/wiki/STL_\(file_format\)](https://en.wikipedia.org/wiki/STL_(file_format)).
- [190] Wikipedia. Structured light scanner. URL: https://en.wikipedia.org/wiki/Structured-light_3D_scanner.
- [191] Wikipedia. Support vector machine. URL: https://en.wikipedia.org/wiki/Support_vector_machine.
- [192] Wikipedia. Upsampling. URL: <https://en.wikipedia.org/wiki/Upsampling>.
- [193] Wikipedia. Validación cruzada. URL: https://es.wikipedia.org/wiki/Validación_cruzada.
- [194] Wikipedia. Vector propio y valor propio. URL: https://es.wikipedia.org/wiki/Vector_propio_y_valor_propio.
- [195] Wikipedia. Wavefront. URL: https://en.wikipedia.org/wiki/Wavefront_.obj_file.
- [196] WolframMathWorld. Normal vector. URL: <http://mathworld.wolfram.com/NormalVector.html>.
- [197] Rafael C. Gonzales , Richard E. Woods. *Digital image processing 3rd Edition*. Pretince Hall, 2002.
- [198] Sikai Xie. 3d pavement surface reconstruction and cracking recognition using kinect-based solution. 2015.
- [199] Richie Zirbes. Scientific visualization: volume surface rendering. URL: <http://johnrichie.com/V2/ricchie/isosurface/volume.html>.
- [200] Cesar Zousa. Kernel functions for machine learning applications. URL: <http://crsouza.com/2010/03/17/kernel-functions-for-machine-learning-applications/#linear>.
- [201] , , , , . Assessment of vehicular transportation quality via smartphones. 2013.
- [202] , , , , . Detection and localization of potholes in roadways using smartphones. 2014.
- [203] Jana Álvarez. Machine learning y support vector machines: porque el tiempo es dinero. URL: <http://www.analiticaweb.es/machine-learning-y-support-vector-machines-porque-el-tiempo-es-dinero-2/>.

First document. This is a simple example, with no extra parameters or packages included.